

GenSys®
Tcl Commands
Reference Guide

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

©2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>. All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

About This User Guide	ix
Customer Support.	ixii
1. Using the GenSys Tcl Commands	
Tcl Commands by Object Type	1-2
Tcl Commands Usage Concepts	1-6
Specifying Names	1-6
Specifying Vector Names.	1-7
Specifying Values in Tcl Commands	1-7
Specifying Values for Integer Arguments.	1-8
Specifying Live Formulae	1-9
Specifying Valid Range	1-9
Specifying Datasource Details for Structured Assembly	1-10
Retrieving the Default Datasource Details for Structured Assembly	1-10
Specifying Vendor and Version Details for Structured Assembly.	1-11
Understanding Bit-Alignment in Connections	1-12
Specifying Perl Callbacks in Tcl Interface	1-12
Using GenSys Special Attributes.	1-12
Tie-off Attributes	1-12
Getting Help on Commands	1-14
Creating ECO Ports.	1-15
Determining the Format and Directory to Save an Object by Using save/saveall	1-17
Determining the Format of Saving an Object	1-17
Determining the Directory to Save an Object	1-17
Generating Schematic in Batch Mode	1-18

Iterating Over a List or Table	1-18
Rerouting GenSys Connections through Via Points	1-19
Use Model for Rerouting GenSys Connections	1-19
Tcl Commands for Rerouting GenSys Connections	1-20
Examples of Rerouting GenSys Connections through Via Points	1-20
Setting GenSys Operating Modes	1-33
Setting an Operating Mode	1-33
Working with the RTL Mode	1-33
Working with the IP-XACT Mode	1-34
Working with the Mixed Mode	1-35
Auto Stop for Non Modified Hierarchies During RTL Design Read	1-35
Example	1-36
Usage Guidelines	1-36

2. GenSys Tcl Commands

add_attribute	2-2
Usage	2-2
Description	2-2
Arguments	2-2
Examples	2-3
add_autoconnect_rule	2-4
Usage	2-4
Description	2-5
Handling Hierarchy Separators while Creating Automatic Connections	2-5
Creating Name-Based Auto Connections	2-6
Arguments	2-6
Examples	2-14
Example 1	2-14
Example 2	2-15
Example 3	2-16
Cases of Autoconnect Rule Run	2-16
Example of Chaining	2-18
Example of the -export_hier Argument	2-20
Example of the -allow_pin_fanout Argument	2-21
add_config	2-21
Usage	2-21
Description	2-21

Arguments	2-21
Examples	2-22
add_connection	2-23
Usage	2-23
Description	2-24
Adding Connections across the Hierarchies	2-24
Arguments	2-25
Examples	2-32
Example 1	2-32
Example 2	2-32
Example 3	2-33
Example 4	2-33
Example 5	2-33
Example 6	2-33
Example 7	2-34
Example 8	2-34
Example 9	2-34
Example 10	2-34
Example 11	2-35
Example 12	2-36
Example 13	2-37
Example 14	2-38
add_datasource	2-41
Usage	2-41
Description	2-41
Arguments	2-41
add_file	2-42
Usage	2-42
Description	2-42
Arguments	2-42
add_file_set	2-44
Usage	2-44
Description	2-44
Arguments	2-45
add_hier_connection	2-45

Usage	2-45
Description	2-45
Arguments	2-45
Examples	2-46
add_info	2-46
Usage	2-46
Description	2-46
Arguments	2-47
Example	2-49
add_instance	2-49
Usage	2-49
Description	2-50
Arguments	2-50
Examples	2-53
Example 1	2-53
Example 2	2-53
Example 3	2-54
Example 4	2-55
Example 5	2-55
add_interface	2-56
Usage	2-56
Description	2-56
Arguments	2-57
Examples	2-61
add_interface_mapping	2-62
Usage	2-62
Description	2-62
Arguments	2-62
Examples	2-63
Example 1	2-63
Example 2	2-64
add_model_view	2-64
Usage	2-64
Description	2-65
Arguments	2-65

add_netname	2-66
Usage	2-66
Description.	2-66
Arguments	2-67
Examples.	2-68
add_parameter	2-69
Usage	2-69
Description.	2-69
Arguments	2-69
Examples.	2-73
Example 1	2-73
Example 2	2-73
Example 3	2-74
add_port	2-74
Usage	2-74
Description.	2-75
Arguments	2-75
Examples.	2-80
Example 1	2-80
Example 2	2-81
Example 3	2-81
Example 4	2-82
add_to_plane	2-82
Usage	2-82
Description.	2-83
Arguments	2-83
Examples.	2-84
Additional Details.	2-84
add_type	2-86
Usage	2-86
Description.	2-86
Arguments	2-87
Examples.	2-87
add_wrapper	2-88
Usage	2-88

Description	2-88
Arguments	2-88
Example	2-89
apply_rule	2-90
Usage	2-90
Description	2-90
Arguments	2-90
Example	2-91
check_rtl	2-91
Usage	2-91
Description	2-92
Arguments	2-92
clearlog	2-92
Usage	2-92
Description	2-92
clone_ip	2-93
Usage	2-93
Description	2-93
Arguments	2-93
Examples	2-94
close_object	2-96
Usage	2-96
Description	2-97
Arguments	2-97
Examples	2-97
closeall	2-97
Usage	2-97
Description	2-97
Examples	2-98
create_report	2-98
Usage	2-98
Description	2-99
Arguments	2-99
Design Differences Report Arguments	2-103

Examples	2-104
Example 1 - Generating a Design Summary Report	2-104
Example 2 - Generating an Interface Summary Report.	2-104
Example 3 - Generating a Connectivity Report	2-104
Example 4 - Generating a Design Differences Report.	2-104
Example 5 - Consolidated Report for All Macros of Imported RTL	2-106
create_report_ERC	2-106
Usage	2-107
Description.	2-107
Arguments	2-107
create_report_Packaging	2-109
Usage	2-109
Description.	2-109
Arguments	2-109
default_datasource	2-110
Usage	2-110
Description.	2-110
Arguments	2-110
default_match_rule	2-112
Usage	2-112
Description.	2-112
Arguments	2-112
del_library_obj	2-113
Usage	2-113
Description.	2-113
Arguments	2-113
delete_autoconnect_rule	2-114
Usage	2-114
Description.	2-114
Deleting Connections Created by an Autoconnect rule	2-115
Arguments	2-115
delete_autoconnections	2-115
Usage	2-116
Description.	2-116

Arguments	2-116
delete_by_datasource	2-117
Usage	2-117
Description.	2-117
delete_component	2-117
Usage	2-117
Description.	2-118
Arguments	2-118
Examples.	2-118
delete_connection.	2-119
Usage	2-119
Description.	2-119
Arguments	2-120
Examples.	2-123
Example 1	2-123
Example 2	2-124
Example 3	2-124
Example 4	2-124
Example 5	2-124
Example 6	2-125
Example 7	2-125
delete_export_interface	2-126
Usage	2-126
Description.	2-126
Arguments	2-126
Examples.	2-127
delete_hier_connection.	2-127
Usage	2-127
Description.	2-127
Arguments	2-127
Examples.	2-128
Command with [-remove_floating_ports FALSE] (default value)	2-128
Command with [-remove_floating_ports TRUE]	2-129
delete_instance.	2-130

Usage	2-130
Description	2-130
Arguments	2-130
Examples	2-130
delete_interface	2-130
Usage	2-130
Description	2-131
Examples	2-131
delete_logicalport_constraints	2-131
Usage	2-131
Description	2-131
Arguments	2-131
delete_parameter	2-132
Usage	2-132
Description	2-132
Arguments	2-132
delete_plane	2-133
Usage	2-133
Description	2-133
Arguments	2-133
delete_port	2-134
Usage	2-134
Description	2-134
Examples	2-134
delete_row	2-135
Usage	2-135
Description	2-135
Notes	2-136
Examples	2-136
diff	2-137
Usage	2-137
Description	2-137
Arguments	2-137
dump_perl_stats	2-138

Usage	2-138
Description	2-138
duplicate_multifanout_port	2-139
Usage	2-139
Description	2-139
Arguments	2-139
Example	2-140
elaborate	2-140
Usage	2-140
Description	2-140
Examples	2-141
export_all	2-141
Usage	2-141
Description	2-141
Arguments	2-142
Examples	2-142
export_data	2-142
Usage	2-143
Description	2-143
Arguments	2-143
Examples	2-144
Example 1	2-144
Example 2	2-145
export_interface	2-145
Usage	2-145
Description	2-145
Case 1	2-146
Case 2	2-146
Case 3	2-146
Arguments	2-146
Examples	2-150
Example 1	2-150
Example 2	2-151
export_pin	2-151

Usage	2-151
Description	2-152
Arguments	2-152
Examples	2-156
Example 1	2-156
Example 2	2-156
Example 3	2-157
export_unconnected	2-158
Usage	2-158
Description	2-158
Arguments	2-159
Example	2-161
get_glue	2-162
Usage	2-162
Description	2-162
Arguments	2-162
get_library_obj	2-162
Usage	2-163
Description	2-163
Arguments	2-163
get_netlist_dir	2-163
Usage	2-164
Description	2-164
get_proc_info	2-164
Usage	2-164
Description	2-164
get_row	2-164
Usage	2-164
Description	2-164
Example	2-165
get_tcltoxpath	2-165
Usage	2-165
Description	2-165
get_template_extension	2-166

Usage	2-166
Description	2-166
get_template_path	2-166
Usage	2-166
Description	2-166
get_xpathtotcl	2-166
Usage	2-166
Description	2-167
gload	2-167
Usage	2-167
Description	2-167
Arguments	2-167
Notes	2-168
group_design	2-169
Usage	2-169
Description	2-169
Arguments	2-170
Examples	2-177
Example 1	2-177
Example 2	2-178
Example 3	2-179
group_glue	2-180
Usage	2-180
Description	2-181
Arguments	2-181
help	2-183
Usage	2-183
Description	2-183
history	2-183
Usage	2-183
Description	2-183
import_data	2-183
Usage	2-184
Description	2-184

Arguments	2-184
incr_sch_add	2-184
Usage	2-184
Description	2-184
Arguments	2-185
incr_sch_print	2-185
Usage	2-185
Description	2-185
Arguments	2-186
incr_sch_set_zoom_factor	2-187
Usage	2-187
Description	2-187
Arguments	2-187
infer_interface_connections	2-188
Usage	2-188
Description	2-188
Arguments	2-188
infer_interfaces	2-189
Usage	2-189
Description	2-189
Arguments	2-190
Examples	2-193
insert_row	2-194
Usage	2-195
Description	2-195
Notes	2-195
load	2-196
Usage	2-196
Description	2-196
Arguments	2-197
Important Notes	2-197
Examples	2-199
load_rtl	2-199
Usage	2-199

Description	2-199
Arguments	2-199
Returns	2-203
Examples	2-204
make_rule	2-204
Description	2-204
Usage	2-204
Arguments	2-204
Example	2-205
match_interface	2-206
Usage	2-206
Description	2-207
Arguments	2-207
Examples	2-209
Example 1	2-209
Example 2	2-209
mod_sch_print	2-210
Usage	2-210
Description	2-210
Arguments	2-210
mod_sch_set_plane_view	2-211
Usage	2-212
Description	2-212
Arguments	2-212
mod_sch_set_zoom_factor	2-212
Usage	2-212
Description	2-213
Arguments	2-213
mod_sch_show	2-213
Usage	2-213
Description	2-213
Arguments	2-214
move_instance	2-214
Usage	2-214

Description	2-214
Arguments	2-215
Examples	2-217
Example 1	2-217
Example 2	2-217
Example 3	2-218
Example 4	2-218
new	2-220
Usage	2-220
Description	2-221
Arguments	2-221
Examples	2-223
optimize_ports	2-223
Usage	2-223
Description	2-224
Prerequisite	2-224
Type of Optimization	2-224
Exceptions	2-225
Arguments	2-225
Examples	2-226
Example 1 - Merge	2-226
Example 2 - Compress and Split	2-227
pop_default_datasource	2-228
Usage	2-228
Description	2-228
print_to_perf_log	2-229
Usage	2-229
Description	2-229
Arguments	2-229
Examples	2-229
push_default_datasource	2-230
Usage	2-230
Description	2-230
Arguments	2-230

refresh_library	2-231
Usage	2-231
Description	2-231
refresh_table	2-231
Usage	2-231
Description	2-232
refresh_ui	2-232
Usage	2-232
Description	2-232
rename	2-232
Usage	2-232
Description	2-233
Arguments	2-233
Examples	2-233
rename_instance	2-233
Usage	2-233
Description	2-233
Arguments	2-234
Examples	2-234
reroute_connection	2-234
Usage	2-234
Description	2-235
Arguments	2-235
Examples	2-236
Example 1	2-236
reroute_fanout_node	2-236
Usage	2-236
Description	2-237
Arguments	2-237
Examples	2-237
reset_rules	2-237
Usage	2-237
Description	2-238
Arguments	2-238

reset_visibility	2-238
Usage	2-238
Description.	2-238
restore_session	2-238
Usage	2-239
Description.	2-239
Arguments	2-239
Example.	2-239
restruct_rtl.	2-240
Usage	2-240
Description.	2-241
Arguments	2-241
Example.	2-241
run.	2-244
Usage	2-244
Description.	2-244
Arguments	2-244
Examples.	2-245
run_autoconnect.	2-245
Usage	2-245
Description.	2-245
Arguments	2-245
Example.	2-246
run_creategroup	2-247
Usage	2-247
Description.	2-247
Arguments	2-247
run_scr_checks.	2-248
Usage	2-248
Description.	2-248
Arguments	2-248
save	2-248
Usage	2-248
Description.	2-249

Saving in a Canonical XML Format	2-249
Saving in an IP-XACT Format	2-250
Limitations of Saving COM Objects in an IP-XACT Format.	2-250
Arguments	2-251
Examples	2-252
save_autoconnect_rules	2-252
Usage	2-252
Description.	2-253
Arguments	2-253
save_session	2-253
Usage	2-253
Description.	2-253
Arguments	2-253
Example.	2-254
saveall.	2-255
Usage	2-255
Description.	2-255
Arguments	2-255
Examples	2-255
sdi_end_update	2-255
Usage	2-256
Description.	2-256
sdi_start_update	2-256
Usage	2-256
Description.	2-256
sdi_update_row	2-256
Usage	2-256
Description.	2-256
select_component.	2-257
Usage	2-257
Description.	2-257
Arguments	2-257
Examples	2-258
select_design	2-258

Usage	2-258
Description.....	2-258
Arguments	2-258
Examples.....	2-259
select_instance.....	2-259
Usage	2-260
Description.....	2-260
Examples.....	2-260
select_interface.....	2-260
Usage	2-260
Description.....	2-260
Examples.....	2-260
select_interfaceinst.....	2-261
Usage	2-261
Description.....	2-261
Arguments	2-261
Example.....	2-261
select_interfaceLib	2-261
Usage	2-262
Description.....	2-262
Arguments	2-262
Examples.....	2-263
select_port	2-263
Usage	2-263
Description.....	2-263
Arguments	2-263
select_row.....	2-264
Usage	2-264
Description.....	2-264
Notes	2-265
Examples.....	2-265
select_subtable.....	2-266
Usage	2-266
Description.....	2-266

Examples	2-266
select_table	2-266
Usage	2-266
Description	2-267
Examples	2-267
select_terminal	2-267
Usage	2-267
Description	2-267
Arguments	2-267
Example	2-268
set_active_object	2-268
Usage	2-268
Description	2-268
Arguments	2-268
Example	2-269
set_attribute	2-270
Usage	2-270
Description	2-270
Arguments	2-270
set_autoconnect_options	2-270
Usage	2-270
Description	2-271
Arguments	2-271
Examples	2-273
Example 1	2-273
Example 2	2-273
Example 3	2-273
set_autoconnect_report_file	2-274
Usage	2-274
Description	2-274
Arguments	2-274
set_default_port_name	2-274
Usage	2-275
Description	2-275

Arguments	2-275
{Expr(<expression1>)}	2-275
{Expr(<expression2>)}	2-276
Example	2-277
Example 1	2-277
Example 2	2-278
set_default_save_dir	2-278
Usage	2-278
Description	2-278
Arguments	2-279
Examples	2-279
set_default_save_format	2-279
Usage	2-279
Description	2-280
set_default_tieoff_prefix	2-280
Usage	2-280
Description	2-280
set_default_vlv	2-281
Usage	2-281
Description	2-281
Arguments	2-281
set_elab_option	2-282
Usage	2-282
Description	2-282
Arguments	2-282
set_extends_vlnv	2-283
Usage	2-284
Description	2-284
Arguments	2-284
set_gensys_define	2-284
Usage	2-284
Description	2-284
Arguments	2-285
Example	2-285

Example 1	2-285
Example 2	2-285
set_gensys_options	2-286
Usage	2-286
Description	2-287
Arguments	2-287
Examples	2-292
Example 1	2-292
Example 2	2-293
TCL Script - test.tcl	2-293
Generated SGDC Constraints File - verify_connections.sgdc	2-293
Notes	2-294
set_glue	2-294
Usage	2-294
Description	2-294
Arguments	2-294
set_group	2-295
Usage	2-295
Description	2-295
Arguments	2-295
Examples	2-296
set_group_mi_module	2-297
Usage	2-297
Description	2-297
Conditions for Multi-Instance Grouping	2-299
Unsupported Scenarios	2-299
Error Scenarios	2-300
.....	
Unsupported Connection Scenarios	2-301
Supported Scenarios for Multi-Instance Grouping	2-304
Arguments	2-310
Example - Design Restructuring Using Multi-Instance Grouping	2-310
set_group_options	2-311
Usage	2-311
Description	2-311
Arguments	2-311

set_hierarchy_separator	2-317
Usage	2-317
Description	2-317
Examples	2-318
set_ifgenerate_options	2-318
Usage	2-318
Description	2-318
Arguments	2-318
Example	2-319
set_interface_port	2-319
Usage	2-319
Description	2-319
Notes	2-319
Arguments	2-320
Examples	2-324
Example 1 - Add Port	2-324
Example 2 - Mapping of a Single Logical Interface Port to Multiple Physical Ports	2-324
set_ipxact_busdef_vlnv	2-325
Usage	2-325
Description	2-325
Arguments	2-325
set_ipxact_version	2-326
Usage	2-326
Description	2-326
set_library_path	2-326
Usage	2-326
Description	2-326
Arguments	2-327
Additional Details	2-328
Case 1	2-328
Case 2	2-328
Case 3	2-329
Case 4	2-329
Examples	2-329
Example 1	2-329

set_log_dir	2-329
Usage	2-329
Description.	2-330
set_lsb_bit.	2-330
Usage	2-330
Description.	2-330
set_messaging_mode.	2-330
Usage	2-331
Description.	2-331
set_msg_severity	2-331
Usage	2-331
Description.	2-331
Arguments	2-331
set_netlist_dir	2-332
Usage	2-332
Description.	2-332
set_netname_option	2-332
Usage	2-332
Description.	2-332
Priority of Net Names.	2-333
Arguments	2-333
Examples.	2-335
Example 1	2-335
Example 2	2-335
set_output_dir.	2-336
Usage	2-336
Description.	2-337
Examples.	2-337
set_packaging_options.	2-337
Usage	2-337
Description.	2-337
Arguments	2-337
Examples.	2-338
Example 1	2-339

Example 2	2-339
Example 3	2-339
set_parameter	2-339
Usage	2-339
Description	2-340
Arguments	2-340
Examples	2-340
set_physical_units	2-341
Usage	2-341
Description	2-341
Arguments	2-341
set_port	2-341
Usage	2-341
Description	2-342
Arguments	2-342
Example	2-347
set_report_dir	2-347
Usage	2-347
Description	2-347
Examples	2-347
set_rtl_option	2-348
Usage	2-348
Description	2-348
Arguments	2-349
Examples	2-358
Example 1	2-358
Example 2	2-359
Example 3	2-359
Example 4	2-360
Example 5	2-361
Example 6	2-362
Example 7	2-363
Example 8	2-364
Example 9	2-367
Example 10	2-367

Example 11 (RTL Stub-Models)	2-368
Example 12 - preserve_floating_net_name	2-370
Example 13	2-370
set_rtlimport_option	2-371
Usage	2-371
Description	2-371
Arguments	2-371
Example	2-376
Example	2-377
set_save_template	2-377
Usage	2-378
Description	2-378
Arguments	2-378
Examples	2-379
Example 1	2-379
Example 2	2-379
set_strict_ipxact	2-379
Usage	2-379
Description	2-380
Example	2-380
set_template_extension	2-381
Usage	2-381
Description	2-381
Arguments	2-381
set_template_path	2-381
Usage	2-381
Description	2-382
Arguments	2-382
set_template_vars	2-382
Usage	2-382
Description	2-382
Arguments	2-382
set_ui_options	2-383
Usage	2-383

Description	2-383
set_verilog_dir	2-383
Usage	2-383
Description	2-383
Preference of a Default Directory for Storing Verilog Files	2-383
set_work_dir	2-384
Usage	2-384
Description	2-384
Examples	2-384
set_xml_dir	2-384
Usage	2-385
Description	2-385
Examples	2-385
set_xslt	2-385
Usage	2-385
Description	2-385
Arguments	2-386
split_struct_ports	2-386
Usage	2-386
Description	2-386
Arguments	2-387
Example	2-387
suppress_message	2-387
Usage	2-387
Description	2-387
Arguments	2-388
syncup_instance	2-388
Usage	2-388
Description	2-388
Arguments	2-388
templategen	2-389
Usage	2-389
Description	2-389
Arguments	2-389

transform_xml	2-390
Usage	2-390
Description	2-390
ungroup_design	2-390
Usage	2-390
Description	2-391
Arguments	2-391
uniquify_instance	2-393
Usage	2-393
Description	2-393
Arguments	2-393
unsuppress_message	2-394
Usage	2-394
Description	2-395
Arguments	2-395
update_design_configuration	2-395
Usage	2-395
Description	2-396
Arguments	2-396
update_logicalport_constraints	2-397
Usage	2-397
Description	2-397
Arguments	2-397
update_master	2-398
Usage	2-398
Usage 1:	2-398
Usage 2:	2-399
Description	2-399
Usage 1:	2-399
Usage 2:	2-400
Arguments	2-400
Examples	2-402
Example 1	2-402
Example 2	2-402

Example 3	2-402
Example 4	2-403
Example 5	2-403
update_master_interface	2-403
Usage	2-403
Description	2-404
Arguments	2-404
Examples	2-405
Example 1	2-405
Example 2	2-405
update_port	2-405
Usage	2-405
Description	2-406
Arguments	2-406
Example	2-406
update_row	2-408
Usage	2-408
Description	2-408
Examples	2-408
update_rule	2-409
Usage	2-409
Description	2-409
Arguments	2-409
validate_ipxactxml	2-410
Usage	2-410
Description	2-410
Arguments	2-410
write_formality_script	2-410
Usage	2-411
Description	2-411

3. GenSys Tcl APIs

Object-to-Methods Tree	3-3
Tcl Relation API Functions: Global API Functions	3-15

get_hierarchyseparator	3-15
Declaration	3-16
Returns	3-16
Example	3-16
get_isMultipleInstantiations	3-16
Declaration	3-16
Returns	3-16
Example	3-16
get_root	3-17
Declaration	3-17
Returns	3-17
Example	3-17
get_report_dir	3-17
Declaration	3-17
Returns	3-17
Example	3-17
get_work_dir	3-17
Declaration	3-17
Returns	3-18
Example	3-18
get_xml_dir	3-18
Declaration	3-18
Returns	3-18
Example	3-18
get_output_dir	3-18
Declaration	3-18
Returns	3-18
Example	3-18
Tcl Relation API Functions: GenSys Built-In Tcl APIs	3-18
Autoconnect Applications.	3-19
Object Retrieval Applications	3-19
connect_by_name	3-20
Declaration	3-20
Arguments	3-20
Examples	3-20
connect_by_name_hier	3-22
Declaration	3-22
Arguments	3-22
Example	3-22

connect_by_prefix	3-22
Declaration	3-22
Arguments	3-22
Example	3-23
connect_by_prefix_hier	3-23
Declaration	3-23
Arguments	3-23
Example	3-24
connect_by_postfix	3-24
Declaration	3-24
Arguments	3-24
Example	3-24
connect_by_postfix_hier	3-25
Declaration	3-25
Arguments	3-25
Example	3-25
connect_by_prefix_postfix	3-25
Declaration	3-25
Arguments	3-25
Example	3-26
connect_by_prefix_postfix_hier	3-26
Declaration	3-26
Arguments	3-26
Example	3-27
tie_high_unconn	3-27
Declaration	3-27
Arguments	3-27
Example	3-27
tie_high_unconn_hier	3-28
Declaration	3-29
Arguments	3-29
Example	3-29
tie_low_unconn	3-29
Declaration	3-29
Arguments	3-29
Example	3-29
tie_low_unconn_hier	3-30
Declaration	3-30
Arguments	3-30
Example	3-30

export_unconn_hier	3-30
Declaration	3-30
Arguments	3-30
Example	3-31
export_unconn_hier_inputs	3-32
Declaration	3-32
Arguments	3-32
Example	3-32
export_unconn_hier_outputs	3-32
Declaration	3-32
Arguments	3-32
Example	3-32
export_unconn_hier_matching_ports	3-33
Declaration	3-33
Arguments	3-33
Example	3-33
get_design_by_name	3-34
Declaration	3-35
Arguments	3-35
Example	3-35
get_instance_by_name	3-35
Declaration	3-35
Arguments	3-35
Example	3-35
get_port_by_name	3-35
Declaration	3-36
Arguments	3-36
Example	3-36
get_terminal_by_name	3-36
Declaration	3-36
Arguments	3-36
Example	3-36
get_all_unconnected_ports	3-37
Declaration	3-37
Arguments	3-37
Example	3-37
get_all_unconnected_ports_hier	3-37
Declaration	3-37
Arguments	3-37
Example	3-37

get_all_unconnected_terminals	3-38
Declaration	3-38
Arguments	3-38
Example	3-38
get_all_unconnected_terminals_hier	3-38
Declaration	3-39
Arguments	3-39
Example	3-39
get_all_connected_ports	3-39
Declaration	3-39
Arguments	3-39
Example	3-39
get_all_connected_ports_hier	3-40
Declaration	3-40
Arguments	3-40
Example	3-40
get_all_connected_terminals	3-40
Declaration	3-40
Arguments	3-40
Example	3-41
get_all_connected_terminals_hier	3-41
Declaration	3-41
Arguments	3-41
Example	3-41
get_all_tied_ports	3-42
Declaration	3-42
Arguments	3-42
Example	3-42
get_all_tied_ports_hier	3-42
Declaration	3-42
Arguments	3-42
Example	3-42
get_all_tied_terminals	3-43
Declaration	3-43
Arguments	3-43
Example	3-43
get_all_tied_terminals_hier	3-43
Declaration	3-43
Arguments	3-43
Example	3-44

get_port_connections	3-44
Declaration	3-44
Arguments	3-44
Example	3-44
get_terminal_connections	3-44
Declaration	3-45
Arguments	3-45
Example	3-45
get_port_tieoff_connections	3-45
Declaration	3-45
Arguments	3-45
Example	3-46
get_terminal_tieoff_connections	3-46
Declaration	3-46
Arguments	3-46
Example	3-46
GenSys Tcl Methods	3-46
do_elaborate	3-46
Applies to	3-47
Arguments	3-47
Description	3-47
Returns	3-47
Example	3-47
do_unelaborate	3-47
Applies to	3-47
Arguments	3-47
Description	3-47
Returns	3-47
Example	3-48
get_actdir	3-48
Applies to	3-48
Arguments	3-48
Description	3-48
Returns	3-48
Example	3-48
get_active_object	3-48
Applies to	3-48
Arguments	3-49
Returns	3-49
Example	3-49

get_aconnects	3-49
Applies to	3-49
Arguments	3-49
Returns	3-49
Example	3-49
get_align	3-50
Applies to	3-50
Arguments	3-50
Returns	3-50
Example	3-50
get_attributes	3-50
Applies to	3-50
Arguments (optional)	3-50
Returns	3-50
Example	3-50
get_autofillschema	3-51
Applies to	3-51
Arguments	3-51
Returns	3-51
Example	3-51
get_backref	3-51
Applies to	3-51
Arguments	3-51
Returns	3-51
Example	3-51
get_category	3-52
Applies to	3-52
Arguments	3-52
Returns	3-52
Example	3-52
get_cell	3-52
Applies to	3-52
Arguments	3-52
Example	3-52
get_cellenums	3-53
Applies to	3-53
Arguments	3-53
Example	3-53
get_celltype	3-53
Applies to	3-53

Arguments	3-53
Returns	3-53
Example	3-53
get_children	3-53
Applies to	3-54
Arguments	3-54
Returns	3-54
Example	3-54
get_clockrate	3-54
Applies to	3-54
Arguments	3-54
Return type	3-54
Example	3-54
get_colnames	3-54
Applies to	3-55
Arguments	3-55
Return type	3-55
Example	3-55
get_columndata	3-55
Applies to	3-55
Arguments	3-55
Return type	3-55
Example	3-55
get_columnschema	3-55
Applies to	3-56
Arguments	3-56
Returns	3-56
Example	3-56
get_columnschemas	3-56
Applies to	3-56
Arguments	3-56
Example	3-56
get_comcomponent	3-56
Applies to	3-56
Arguments	3-57
Returns	3-57
Example	3-57
get_command	3-57
Applies to	3-57
Arguments	3-57

Returns	3-57
Example	3-57
get_components	3-57
Applied to	3-57
Arguments (optional)	3-58
Returns	3-58
Example	3-58
get_controlclock	3-58
Applies to	3-58
Arguments	3-58
Example	3-58
get_controlreset	3-59
Applies to	3-59
Arguments	3-59
Example	3-59
get_creationdate	3-59
Applies to	3-59
Arguments	3-59
Returns	3-59
Example	3-59
get_data	3-59
Applies to	3-60
Arguments	3-60
Returns	3-60
Example	3-60
get_datasource	3-60
Applies to	3-60
Arguments	3-60
Returns	3-60
Example	3-60
get_datetime	3-60
Applies to	3-61
Arguments	3-61
Returns	3-61
Example	3-61
get_default	3-61
Applies to	3-61
Arguments	3-61
Returns	3-61
Example	3-61

get_deltadelay	3-61
Applies to	3-62
Arguments	3-62
Returns	3-62
Example	3-62
get_dependcolumns	3-62
Applies to	3-62
Arguments	3-62
Example	3-62
get_description	3-62
Applies to	3-62
Arguments	3-63
Returns	3-63
Example	3-63
get_designs	3-63
Applies to	3-63
Arguments	3-63
Returns	3-63
Example	3-63
get_design_objects	3-64
Applied to	3-64
Arguments (optional)	3-64
Returns	3-64
Example	3-64
get_dir	3-64
Applies to	3-64
Arguments	3-64
Returns	3-65
Example	3-65
get_end	3-65
Applies to	3-65
Arguments	3-65
Returns	3-65
Example	3-65
get_endian	3-65
Applies to	3-65
Arguments	3-66
Returns	3-66
Example	3-66
get_enumchoices	3-66

Applies to	3-66
Arguments	3-66
Returns	3-66
Example	3-66
get_expr	3-66
Applies to	3-66
Arguments	3-67
Returns	3-67
Example	3-67
get_file	3-67
Applies to	3-67
Arguments	3-67
Example	3-67
get_first	3-67
Applies to	3-67
Arguments	3-67
Returns	3-68
Example	3-68
get_frozen	3-68
Applies to	3-68
Arguments	3-68
Returns	3-68
Example	3-68
get_functiontype	3-68
Applies to	3-68
Arguments	3-69
Returns	3-69
Example	3-69
get_hdl_type	3-69
Applies to	3-69
Arguments	3-69
Returns	3-69
Example	3-69
get_hdl_type_lsb	3-70
Applies to	3-70
Arguments	3-70
Returns	3-70
Example	3-70
get_hdl_type_msb	3-70
Applies to	3-70

Arguments	3-70
Returns	3-70
Example	3-70
get_hdl_type_language	3-71
Applies to	3-71
Arguments	3-71
Returns	3-71
Example	3-71
get_hdl_type_library	3-71
Applies to	3-71
Arguments	3-71
Returns	3-71
Example	3-71
get_hdl_type_package	3-72
Applies to	3-72
Arguments	3-72
Returns	3-72
Example	3-72
get_help	3-72
Applies to	3-72
Arguments	3-72
Returns	3-72
Example	3-72
get_hiddencolumns	3-73
Applies to	3-73
Arguments	3-73
Example	3-73
get_hiername	3-73
Applies to	3-73
Arguments	3-73
Example	3-73
get_hiernames	3-73
Applies to	3-73
Arguments	3-74
Returns	3-74
Example	3-74
get_iconnects	3-74
Applies to	3-74
Arguments	3-74
.	3-74

Returns	3-74
Example	3-74
get_ifinstance.	3-74
Applies to	3-74
Arguments	3-75
Returns	3-75
Example	3-75
get_ifinstances.	3-75
Applies to	3-75
Arguments	3-75
Returns	3-75
Example	3-75
get_interface	3-75
Applies to	3-76
Arguments	3-76
Returns	3-76
Example	3-76
get_interfaceGroup	3-76
Applies to	3-76
Arguments	3-76
Returns	3-76
Example	3-76
get_instance	3-76
Applies to	3-77
Arguments	3-77
Returns	3-77
Example	3-77
get_instances.	3-77
Applies to	3-77
Arguments (optional)	3-77
Returns	3-77
Example	3-77
get_intdata	3-78
Applies to	3-78
Arguments	3-78
Returns	3-78
Example	3-78
get_interfacedef.	3-78
Applies to	3-78
Arguments	3-78

Returns	3-78
Example	3-79
get_interfacedefs	3-79
Applied to	3-79
Arguments (optional)	3-79
Returns	3-79
Example	3-79
get_interfaces	3-79
Applies to	3-79
Arguments (optional)	3-79
Returns	3-80
Example	3-80
get_isautofill	3-80
Applies to	3-80
Arguments	3-80
Returns	3-80
Example	3-80
get_isbasetable	3-80
Applies to	3-80
Arguments	3-80
Returns	3-81
Example	3-81
get_isdesign	3-81
Applies to	3-81
Arguments	3-81
Returns	3-81
Example	3-81
get_is_elab_generated	3-81
Applies to	3-81
Arguments	3-81
Returns	3-81
Example	3-82
get_iseval	3-82
Applies to	3-82
Arguments	3-82
Returns	3-82
Example	3-82
get_isevalallowed	3-82
Applies to	3-82
Arguments	3-82

Returns	3-82
Example	3-83
get_isevalcolpresent	3-83
Applies to	3-83
Arguments	3-83
Returns	3-83
Example	3-83
get_isextension	3-83
Applies to	3-83
Arguments	3-83
Returns	3-83
Example	3-83
get_isextensiontable	3-84
Applies to	3-84
Arguments	3-84
Returns	3-84
Example	3-84
get_isflattable	3-84
Applies to	3-84
Arguments	3-84
Returns	3-84
Example	3-84
get_is_exported	3-85
Applies to	3-85
Arguments	3-85
Returns	3-85
Example	3-85
get_ishidden	3-85
Applies to	3-85
Arguments	3-85
Returns	3-85
Example	3-85
get_isinternal	3-86
Applies to	3-86
Arguments	3-86
Returns	3-86
Example	3-86
get_iskey	3-86
Applies to	3-86
Arguments	3-86

Returns	3-86
Example	3-86
get_ismirror	3-87
Applies to	3-87
Arguments	3-87
Returns	3-87
Example	3-87
get_is_multi_dimension	3-87
Applies to	3-87
Arguments	3-87
Returns	3-87
Example	3-87
get_isopen	3-88
Applies to	3-88
Arguments	3-88
Returns	3-88
Example	3-88
get_is_open	3-88
Applies to	3-88
Arguments	3-88
Returns	3-88
Example	3-88
get_isreadonly	3-89
Applies to	3-89
Arguments	3-89
Returns	3-89
Example	3-89
get_is_scalar	3-89
Applies to	3-89
Arguments	3-89
Returns	3-89
Example	3-89
get_istab	3-90
Applies to	3-90
Arguments	3-90
Returns	3-90
Example	3-90
get_istable	3-90
Applies to	3-90
Arguments	3-90

Returns	3-90
Example	3-90
get_istemp	3-91
Applies to	3-91
Arguments	3-91
Returns	3-91
Example	3-91
get_is_term_or_port.	3-91
Applies to	3-91
Arguments	3-91
Returns	3-91
Example	3-91
get_isvalidate.	3-92
Applies to	3-92
Arguments	3-92
Returns	3-92
Example	3-92
get_isvalidcolpresent.	3-92
Applies to	3-92
Arguments	3-92
Returns	3-92
Example	3-92
get_isvalidateallowed.	3-93
Applies to	3-93
Arguments	3-93
Returns	3-93
Example	3-93
get_key	3-93
Applies to	3-93
Arguments	3-93
Example	3-93
get_keycolumns.	3-93
Applies to	3-94
Arguments	3-94
Example	3-94
get_language.	3-94
Applies to	3-94
Arguments	3-94
Returns	3-94
Example	3-94

get_library	3-94
Applies to	3-94
Arguments	3-95
Returns	3-95
Example	3-95
get_lname	3-95
Applies to	3-95
Arguments	3-95
Returns	3-95
Example	3-95
get_line	3-95
Applies to	3-95
Arguments	3-96
Example	3-96
get_longlongintdata	3-96
Applies to	3-96
Arguments	3-96
Example	3-96
get_lports	3-96
Applies to	3-96
Arguments (optional)	3-96
Returns	3-96
Example	3-97
get_lsb	3-97
Applies to	3-97
Arguments	3-97
Returns	3-97
Example	3-97
get_macro_visibility	3-97
Applies to	3-97
Arguments	3-97
Returns	3-97
Example	3-98
get_master	3-98
Applies to	3-98
Arguments	3-98
Returns	3-98
Example	3-98
get_masterrtlcomponent	3-98
Applies to	3-98

Arguments	3-98
Returns	3-98
Example	3-99
get_maxdatawidth	3-99
Applies to	3-99
Arguments	3-99
Returns	3-99
Example	3-99
get_maxvalue	3-99
Applies to	3-99
Arguments	3-99
Returns	3-99
Example	3-100
get_memories	3-100
Applies to	3-100
Arguments (optional)	3-100
Returns	3-100
Example	3-100
get_method	3-100
Applies to	3-100
Arguments	3-100
Returns	3-100
Example	3-101
get_memory	3-101
Applies to	3-101
Arguments	3-101
Returns	3-101
Example	3-101
get_minvalue	3-101
Applies to	3-101
Arguments	3-101
Returns	3-102
Example	3-102
get_morebackref	3-102
Applies to	3-102
Arguments	3-102
Returns	3-102
Example	3-102
get_msb	3-102
Applies to	3-102

Arguments	3-102
Returns	3-103
Example	3-103
get_name	3-103
Applies to	3-103
Arguments	3-103
Returns	3-103
Example	3-103
get_ncols	3-103
Applies to	3-103
Arguments	3-103
Returns	3-104
Example	3-104
get_nrows	3-104
Applies to	3-104
Arguments	3-104
Returns	3-104
Example	3-104
get_numrows	3-104
Description	3-104
Applies to	3-105
Arguments	3-105
Example	3-105
get_offset	3-105
Applies to	3-105
Arguments	3-105
Returns	3-105
Example	3-105
get_optional	3-105
Applies to	3-105
Arguments	3-106
Returns	3-106
Example	3-106
get_other_end	3-106
Applies to	3-106
Arguments	3-106
Returns	3-106
Example	3-106
get_parameters	3-107
Applies to	3-107

Arguments (optional)	3-108
Returns	3-108
Example	3-108
get_parent	3-108
Applies to	3-108
Arguments	3-108
Returns	3-108
Example	3-108
get_parentrtlcomponent.	3-109
Applies to	3-109
Arguments	3-109
Returns	3-109
Example	3-109
get_person.	3-109
Applies to	3-109
Arguments	3-109
Returns	3-109
Example	3-109
get_partition.	3-110
Applies to	3-110
Arguments	3-110
Returns	3-110
Example	3-110
get_partitions	3-110
Applies to	3-110
Arguments	3-110
Returns	3-110
Example	3-110
get_path.	3-111
Applies to	3-111
Arguments	3-111
Returns	3-111
Example	3-111
get_plane.	3-111
Applies to	3-111
Arguments	3-111
Returns	3-111
Example	3-111
get_port	3-112
Applies to	3-112

Arguments	3-112
Returns	3-112
Example	3-112
get_port_connected	3-112
Applies to	3-112
Arguments	3-112
Returns	3-112
Example	3-113
get_portname	3-113
Applies to	3-113
Arguments	3-113
Returns	3-113
Example	3-113
get_ports	3-114
Applies to	3-114
Arguments (optional)	3-114
Returns	3-114
Example	3-114
get_powerdomain	3-114
Applies to	3-115
Arguments	3-115
Returns	3-115
Example	3-115
get_ptable	3-115
Applies to	3-115
Arguments	3-115
Returns	3-115
Example	3-115
get_ptablecolname	3-115
Applies to	3-116
Arguments	3-116
Returns	3-116
Example	3-116
get_ptablerownum	3-116
Applies to	3-116
Arguments	3-116
Returns	3-116
Example	3-116
get_reason	3-116
Applies to	3-116

Arguments	3-117
Returns	3-117
Example	3-117
get_reset	3-117
Applies to	3-117
Arguments	3-117
Returns	3-117
Example	3-117
get_resetmask	3-117
Applies to	3-117
Arguments	3-118
Returns	3-118
Example	3-118
get_rowdata	3-118
Applies to	3-118
Arguments	3-118
Returns	3-118
Example	3-118
get_rowtype	3-118
Applies to	3-118
Arguments	3-119
Returns	3-119
Example	3-119
get_rtlfiles	3-119
Applies to	3-119
Arguments	3-119
Returns	3-119
Example	3-119
get_rtlinstances	3-119
Applies to	3-119
Arguments	3-120
Returns	3-120
Example	3-120
get_rtlports	3-120
Applies to	3-120
Arguments	3-120
Returns	3-120
Example	3-120
get_second	3-120
Applies to	3-120

Arguments	3-121
Returns	3-121
Returns	3-121
Example	3-121
get_scalarports	3-121
Applies to	3-121
Arguments	3-121
Returns	3-121
get_scalarterminals	3-121
Applies to	3-122
Arguments	3-122
Returns	3-122
get_selfinstances	3-122
Applies to	3-122
Arguments	3-122
Returns	3-122
Example	3-122
get_shortcode	3-122
Applies to	3-122
Arguments	3-122
Returns	3-123
Example	3-123
get_size	3-123
Applies to	3-123
Arguments	3-123
Returns	3-123
Example	3-123
get_splices	3-123
Applies to	3-123
Arguments	3-123
Returns	3-124
Example	3-124
get_start	3-124
Applies to	3-124
Arguments	3-124
Returns	3-124
Example	3-124
get_subtable	3-124
Applies to	3-124
Arguments	3-124

Returns	3-125
Example	3-125
get_table	3-125
Applies to	3-125
Arguments	3-125
Returns	3-125
Example	3-125
get_tables	3-125
Applies to	3-125
Arguments (optional)	3-125
Returns	3-126
Example	3-126
get_tableschema	3-126
Applies to	3-126
Arguments	3-126
Returns	3-126
Example	3-126
get_tconnects	3-126
Applies to	3-126
Arguments	3-127
Returns	3-127
Example	3-127
get_terminals	3-127
Applies to	3-127
Arguments (Optional)	3-127
Returns	3-127
Example	3-127
get_tid	3-127
Applies to	3-128
Arguments	3-128
Returns	3-128
Example	3-128
get_tieoffs	3-128
Applies to	3-128
Arguments	3-128
Returns	3-128
Example	3-128
get_type	3-128
Applies to	3-129
Arguments	3-129

Returns	3-129
Example	3-129
get_validate	3-129
Applies to	3-129
Description	3-130
Arguments	3-130
Example	3-130
get_value	3-130
Applies to	3-130
Arguments	3-130
Returns	3-130
Example	3-130
get_vendor	3-130
Applies to	3-130
Arguments	3-131
Returns	3-131
Example	3-131
get_version	3-131
Applies to	3-131
Arguments	3-131
Returns	3-131
Example	3-131
get_vnlv	3-131
Applies to	3-131
Arguments	3-132
Returns	3-132
Example	3-132
get_voltage	3-132
Applies to	3-132
Arguments	3-132
Returns	3-132
Example	3-132
get_width	3-132
Applies to	3-132
Arguments	3-133
Returns	3-133
Example	3-133
get_xmlfile	3-133
Applies to	3-133
Arguments	3-133

Returns 3-133

Example 3-133

Preface

This preface includes the following sections:

- [About This User Guide](#)
- [Customer Support](#)

About This User Guide

Tcl support is provided by GenSys® for interactive command entry and for command scripting. The GenSys Tcl Commands Reference Guide describes GenSys Tcl commands and usage concepts.

Related Publications

For additional information about GenSys, see the documentation on SolvNet at the following address:

<https://solvnet.synopsys.com/DocsOnWeb>

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *GenSys Release Notes* on the SolvNet site.

To see the *GenSys Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

<https://solvnet.synopsys.com/DownloadCenter>

2. Select GenSys, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</code>
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Reporting an Error

The Technical Publications team welcomes your feedback and suggestions on this publication. Provide specific feedback and, if possible, attach a snapshot. Send your feedback at spyglass_support@synopsys.com.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

<https://solvnet.synopsys.com>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at <https://solvnet.synopsys.com>, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at <https://www.synopsys.com/support/global-support-centers.html>
- Telephone your local support center.

- Call (800) 245-8005 from within North America.
- Find other local support center telephone numbers at
<https://www.synopsys.com/support/global-support-centers.html>

1

Using the GenSys Tcl Commands

Tcl support is provided by GenSys® for interactive command entry and for command scripting, as is common in many EDA tools. Therefore, there is a command-box at the bottom of the GenSys Main Window for entering the Tcl commands. Commands from the command-line are echoed in the log window. A copy from the log window to the command line is also supported.

The primary purpose of Tcl command support is to help simplify repetitive tasks and also to provide a mechanism to import data from other sources into the tool by using a simple format. For example, you can enter register information for an IP by formatting that register information into a script and then running the script in GenSys.

This mechanism provides an alternate path to load large amounts of legacy data rather than requiring all the data to be entered manually by using the GUI controls. This mechanism also provides a convenient way to simplify other repetitive commands through user-defined Tcl scripts. For example, you can enter eight registers with one command, when register names and other characteristics are somehow determinable in a script.

GenSys also provides you the flexibility of creating your own user-defined Tcl commands. The user-defined Tcl commands can be used to add, delete, or update data in the extension tables. Based on the generic Tcl commands schema, GenSys internally creates these Tcl commands. These Tcl commands can be used in GenSys just like any other standard Tcl commands. For more details, refer to *Creating User-Defined Tcl Commands* section of *GenSys Schema User Guide*.

Tcl Commands by Object Type

The following tree lists the available Tcl commands under their respective object types:

- `setup::`
 - `set_library_path`
 - `set_log_dir`
 - `set_hierarchy_separator`
 - `set_output_dir`
 - `set_report_dir`
 - `set_log_dir`
 - `set_xml_dir`
- `design::`
 - `new`
 - `gload`
 - `load`
 - `select_design`
 - `del_library_obj`
 - `add_config`
 - `add_info`
 - `elaborate`
 - `rename`
 - `set_elab_option`
 - `get_library_obj`
 - `unsuppress_message`
 - `update_master`
- `::Ports`
 - `add_port`
 - `delete_port`
- `::Instance`
 - `add_instance`

[add_parameter](#)

[select_instance](#)

[set_parameter](#)

[delete_instance](#)

[refresh_library](#)

[group_design](#)

[ungroup_design](#)

- `::Interface`

[add_interface](#)

[select_interface](#)

[delete_interface](#)

[set_interface_port](#)

[get_netlist_dir](#)

[set_netlist_dir](#)

- `::RTL Connectivity`

[add_connection](#)

[delete_connection](#)

[export_interface](#)

[delete_export_interface](#)

[export_pin](#)

[make_rule](#)

[get_glue](#)

[load_rtl](#)

[set_rtl_option](#)

- `::Partitions`

[create_report](#)

[set_physical_units](#)

[set_hierarchy_separator](#)

- `component::`

[new](#)

- gload
- load
- select_component
- del_library_obj
- delete_component
- add_config
- add_info
- add_port
- rename
- get_library_obj
- unsuppress_message
- ::Instance
 - get_netlist_dir
 - set_netlist_dir
- ::Interface
 - add_interface
 - select_interface
 - delete_interface
 - set_interface_port
 - get_netlist_dir
 - set_netlist_dir
- ::Parameters
 - add_type
 - add_parameter
 - set_parameter
- interface:::
 - new
 - gload
 - load
 - select_interfaceLib

`del_library_obj`
`add_config`
`add_info`
`get_netlist_dir`
`set_netlist_dir`
`rename`
`get_library_obj`
`unsuppress_message`

- `::Interface Ports`

`set_interface_port`

- `table::`

`select_table`
`select_row`
`insert_row`
`delete_row`
`export_all`
`load`
`select_subtable`
`clearlog`
`export_all`
`write_formality_script`
`update_row`
`clearlog`
`add_type`

- `utility::`

`refresh_library`
`run`
`create_report`
`create_report`
`closeall`

[save](#)[save_autoconnect_rules](#)[help](#)[history](#)[clearlog](#)

Tcl Commands Usage Concepts

Specifying Names

All table/column names specified with the Tcl commands must be MRS names.

You can specify any table/column names available in the current scope:

- The name of a column of the same table is specified as `$<col-name>`. For example, `$MSB`, or `$Size`. The name of a column of a different table is specified as `$<table-name>.<col-name>`.
- Depending on the relative position of the column to be specified, the format is as follows:

Location	Format
Same row, same table	<code>\$<col-name></code>
Different column, same table	<code>./\$<col-name></code>
Different column, different table (in the same table hierarchy)	<code><hier>/\$<table-name>.<col-name></code> Where <i><hier></i> is the relative path of the column as in the following example: <code>../\$attributes.Name</code> Each <code>..</code> takes you one level up the table hierarchy.
Column in an unrelated table	Specify the hierarchical column name with respect to the top as in the following example: <code>::DES1::CINST1::Name</code> The above specification indicates the Name column of the component instance CINST1 in design DES1.

Location	Format
Column with specified value in a row	<p>Specify as in the following example:</p> <p><code>\$attributes[Name=A1].Value</code></p> <p>The above specification indicates value in the Value column of the attributes table corresponding to the row where the value of the Name column (of the same attributes table) is A1.</p>

Note:

Names are case-sensitive.

Specifying Vector Names

You may need to specify a portbit or a termbit of a vector port/pin in a Tcl command. Then, you must escape the square brackets in the portbit/termbit using the backslash (\) character as in the following example that indicates C1[1]:

```
-add_connection .. -port C1\[1\] ...
```

Specifying Values in Tcl Commands

GenSys now supports Decimal, C-Hex, C-Octal, Verilog-Binary, Verilog-Hex, Verilog Decimal, Verilog-Octal, VHDL-Hex and VHDL-Octal, VHDL-Binary data representations.

Value	Example
Decimal values	123
C-Hex values	0xA24
C-Octal values	0123
Verilog-Binary values	4'b0101 or 'b0101 or s0101
Verilog-Hex values	4'hAF20
Verilog-Decimal values	4'd2340
Verilog-Octal	7777

Value	Example
VHDL-Hex	x"FF"
VHDL-Octal	o"77"
VHDL-Binary	"10110"

Note:

The default and tieoff values can be specified as Verilog binary values prefixed with s (in the s<value> format) or as -1. Values specified as s<value> are considered as binary values and are sign-extended to fill the field, if required. If the width of <value> is less than width of field, <value> will be sign-extended to fill the field (i.e. LSB of the value will be copied into low order bit(s) and the MSB of the value will be copied into all higher bits of the field). For example, for a 4-bit field, value s10 would be expanded to 1110. Value specified as -1 expands to as many 1's as are required to fill the field. No other negative value is allowed.

Specifying Values for Integer Arguments

You can specify values of the Integer arguments in the following ways:

Value	Example
An integer number	-lsb 0 -lsb 7
Simple expressions based on other table values or parameters	-lsb {[celldata MSB]-8} -lsb {[p PARAM1]-8} -msb {[log2 [p PARAM1]]- 1}
Formula-based	See Specifying Live Formulae .

Note:

Names are case-sensitive.

Integer values greater than 32-bit are now supported in GenSys. Such integer specifications can be given in:

- Tie-off values/default values. These can take any arbitrary length of integers.

- Address space specification (used in `add_register` and `add_address_map` commands), you can specify integer length up to 64 bits.
- Perl specifications.

Specifying Live Formulae

You can specify a live formula as the Integer argument value in the following format:

```
{=[cellldata <table-hier-name>(<key-col-name>).<col-name>]}
```

Consider the following example:

```
add_port
  -name CLK1
  -direction IN
  -lsb 0
  -msb {[cellldata \
component.attributes(wait_state).Value]}
```

Here, the command adds a port named CLK1 to the current component as input port with LSB as 0 and MSB as the value in the Value column corresponding to the row with value `wait_state` of the attributes table of the current component.

Specifying Valid Range

Some Tcl commands such as the [add_parameter](#) command have the `-validrange` argument that enables you to specify the range of the object values. You can specify the range value as one of the following:

Range Type	Example
Single range	<p><code>-validrange 4:6</code></p> <p>Indicates all possible values of the specified type from 4 to 6</p> <p>You can also specify the range as <code>-validrange 4-6</code></p> <p><code>-validrange 6:</code></p> <p>Indicates all possible values of the specified type from 6 onwards</p>

Range Type	Example
Multiple range	<p><code>-validrange 2-4,6</code></p> <p>Indicates all possible values of the specified type from 2 to 4 and value 6</p> <p><code>-validrange 2:4,6:8</code></p> <p>Indicates all possible values of the specified type from 2 to 4 and from 6 to 8</p> <p>You can specify any number of ranges.</p> <p><code>-validrange 2-4,8-</code></p> <p>Indicates all possible values of the specified type from 2 to 4 and from 8 onwards</p>

Note:

Both : and - can be used as separators to specify a valid range.

Note:

It is invalid to specify parenthesis or space in the value of the `-validrange` argument.

Specifying Datasource Details for Structured Assembly

GenSys allows you to save datasource details such as creation date, change date, change reason, or person name. To add these datasource details, use the [add_datasource](#) and [default_datasource](#) Tcl commands.

The `default_datasource` Tcl command enables or disables the addition of datasources to all the COM objects created after the execution of this command. If enabled, all the default datasource details are added to the COM objects. To add any specific datasource details for a currently active object, execute the `add_datasource` command.

Retrieving the Default Datasource Details for Structured Assembly

You can retrieve the default datasource details by using the `DataSourceGetPersonName`, `DataSourceGetCreationDate`, `DataSourceGetChangeReason`, `DataSourceGetChangeType`, `DataSourceGetChangeMethod`, `DataSourceGetChangeDateTime`, and `DataSourceGetDatalsFrozen` APIs.

For more information on these API functions, refer to *GenSys CustomizationGuide*.

Consider the following example in which the default datasource details is being set using the `default_datasource` Tcl command:

```
default_datasource -name myDataSrc \  
-creationdate 06.01.09\  
-changereason "to test"\  
-changetype "type 100"\  
-changemethod "method jff3"\  
-changedate "06.01.09 second half"\  
-datafrozen FALSE
```

To retrieve the person name field of the default datasource, perform the following steps:

- Get the default datasource by using the `GetDefaultDataSource` API, as shown below:

```
my $datasource = GetDefaultDataSource();
```

The above command captures the default datasource from the top of the default datasource stack.

- Get the person name of the default datasource, as shown below:

```
my $datasource_personName =  
DataSourceGetPersonName($datasource);
```

The above command captures the person name, `myDataSrc`, of the default datasource.

Similarly, you can retrieve the other fields of the default datasource.

Specifying Vendor and Version Details for Structured Assembly

Some Tcl commands like the [add_instance](#) command have two arguments (`-vendor` and `-version`) that allow you to specify the exact master file in the library when the library has multiple master files of different vendors/versions.

If you specify both arguments, GenSys searches the specified library by the specified vendor, master name, and version (VNV) and uses the first found matching master file.

If you specify only the `-vendor` argument, GenSys searches the specified library by the specified vendor and master name (VN) and uses the first found matching master file.

If you specify only the `-version` argument, GenSys searches the specified library by the specified master name and version name (NV) and uses the first found matching master file.

If you do not specify both these arguments, GenSys searches the specified library by the specified master name (N) and uses the first found matching master file.

Understanding Bit-Alignment in Connections

The [add_connection](#) command has an argument named `-align` that specifies the bit-level connectivity for the connection being made. Similarly, the [add_port](#) command also has the `-align` argument that specifies the same whenever the port being created is connected.

The allowed values of the `-align` argument are MSB (default) and LSB.

Under the MSB mode (default), the bits of two vector objects are connected starting with the MSB of both objects. Similarly, under the LSB mode, the bits of two vector objects are connected starting with the LSB of both objects.

When the two objects have the same number of bits being connected, identical connections are produced in both MSB and LSB modes. However, when the two objects do not have the same number of bits being connected, the connections produced are different. For example, you are connecting P1[7:5] with P2[9:8]. Under the MSB mode, the connections are P1[7]<>P2[9] and P1[6]<>P2[8] with P1[5] left unconnected. Under the LSB mode, the connections are P1[5]<>P2[8] and P1[6]<>P2[9] with P1[7] left unconnected.

Note:

If you want the default alignment to be LSB, use the [set_elab_option](#).

Specifying Perl Callbacks in Tcl Interface

You can register Perl callback functions for Tcl commands by using the `RegisterPreCommand` and `RegisterPostCommand` API functions. For more information on these API functions, refer to *GenSys Customization Guide*.

Using GenSys Special Attributes

Tie-off Attributes

Tie-offs often need to be connected to TIELO or TIEHI cells rather than logical 1/0, to allow for ECOs to configure modules and to prevent undesired optimizations.

This has been implemented through special attributes TIEHI and TIELO.

The attributes TIEHI and TIELO on a port, interface, component, or design will indicate the tieoff cell and signal names that should be used for 0 (TIELO) or 1 (TIEHI). If no attribute is present, or if the attributes are set to 0 (TIELO) and 1 (TIEHI) then the constants 0 and 1 will be used in RTL generation. The attribute specifies the cell name and signal name, separated by a . (dot). For example, `cell_name.signal_name`. The signal name will be the actual signal (port) name.

It is possible to override these special attributes. The order of precedence for the TIELO and TIEHI attributes is given below (item 1 being the highest priority, item 5 being the lowest priority):

1. Port
2. Interface
3. Component
4. Design
5. Default to constants 0, 1.

If the value of the special attribute is explicitly specified for the port, then the same value will be used for the port. However, if the value is not explicitly set for the port, then the value of the attribute set for the interface instance will be used for the port.

The following example illustrates the use of the special attributes:

Consider the following scenario of the use of attribute TIELO on the component and its interfaces and/or ports.

On component X:

```
add_attribute -name TIELO -value lowcell.TIELO
```

On interface A on X:

```
add_attribute -name TIELO  
              -value otherlowcell.TIELO
```

On port ABA on interface A on component X:

```
add_attribute -name TIELO -value 0
```

Similarly, on component Y:

...

On interface C on Y:

...

On port CCC on interface C on Y:

```
add_attribute -name TIELO -value 0
```

The value of the special attributes in the above cases would be:

Port ABA on interface A of X (tied to 0): 0

The value 0 is the last specified value of TIELO on the interface port in the interface instance.

Port BBB on interface A of X (tied to 0): otherlowcell.TIELO

For port BBB, the value is picked from the interface as it has not been explicitly set for this port.

Port ABA on interface B of X (tied to 0): lowcell.TIELO

For port ABA on interface B, the value is picked from the component as the value has not been explicitly set.

Port CCC on interface C of Y (tied to 0): 0

The value of port CCC is explicitly set as 0.

Getting Help on Commands

To get help on all the Tcl commands, type `help` at the Tcl Command window as shown:

```
help
```

The `help` command also supports regular expressions. For example, consider the following specifications:

```
help -d help
help add*
help *load
```

To get help on a particular command *<command-name>*, type the following at the Tcl Command window:

```
help <command-name>
```

or

You can also specify the `-help` option after the command name as shown below:

```
<command-name> -help
```

Note:

You can give the `-help` option after the command name or after any number of options specified for the command.

Note:

The *<command>* `-help` specification does not support regular expressions.

Creating ECO Ports

GenSys provides the partitioning capability for hierarchy manipulation in RTL generation. Connections between the instances that cross various hierarchical boundaries results in the generation of some additional ports on the hierarchical boundaries that the connection crosses.

However, after the RTL boundary freeze, you may want to add some additional ports on the hierarchy boundaries and make some connection through these ports. GenSys enables you to do so through ECO ports. ECO ports are the additional ports that can be added to the hierarchy boundaries, and all the incremental connections (after the RTL boundary freeze) are routed through these ECO ports, as specified by the user. Since a connection can cross multiple hierarchies from source to the sink, user can control on the ECO ports on all the intervening hierarchies.

You can create a partition hierarchy on which ECO ports would be added by using the `-hierarchy` argument of the [add_port](#) Tcl command. For example, consider the following commands:

```
add_port -name eco_port_0 -msb 7 -lsb 0 -direction IN
-hierarchy VD1
add_port -name eco_port_1 -msb 7 -lsb 0 -direction IN
-hierarchy PD1
add_port -name eco_port_0 -msb 7 -lsb 0 -direction
OUT -hierarchy PD2
```

The above commands add ECO ports on partitioned hierarchies specified by using the `-hierarchy` argument. GenSys does not process these port connections (so that RTLHC, etc., would not work on them) till RTL netlisting. During RTL generation, GenSys process these `add_port` commands on the RTL object model.

Once you have created the partition hierarchy and ECO ports by using the `add_port` Tcl command, you can make additional connections through these ECO ports after the RTL freeze by using the [add_connection](#) Tcl command. In this Tcl command, you need to provide explicit hierarchy port names (specified by using the `-pin` argument) at each partition hierarchy which the connection crosses. For a hierarchy where the port name is not provided, it defaults to GenSys generated port names.

Such connections are shown in RTLHC, and other generators can query these connections like any other connections. The hierarchy/pin information is stored as additional information on these connections. Partitioning uses this information if the connection actually crosses the mentioned hierarchy, and uses the specified port name or creates its own port name if the information is not present.

Consider the following example:

```
new component compl
add_port -name R1 -lsb 0 -msb 1 -direction OUT
save compl
```

```

new component comp2
add_port -name R2 -direction IN -lsb 0 -msb 1
save comp2
new design top
add_instance -name I1 -master comp1
add_instance -name I2 -master comp2
add_connection -instance I1 -pin R1 -instance I2 -pin
R2
create_partition -name P1
set_partition -instance I1 -hierarchy H1
add_port -name P1 -direction OUT -lsb 0 -msb 1
-hierarchy H1

```

In the above example, the I1 instance is placed inside the hierarchy, H1. In addition, the P1 port has been added on the hierarchy, H1, and the connection is specified between I1.R1 and I2.R2 is through the newly added port, H1.P1[1:0]. In this case, the generated RTL is as follows:

```

module top ();
wire [1:0] H1_I1_R1;
    comp2    I2 (
        .R2(H1_I1_R1)    //I:2);
    H1 U0_H1 ( .P1(H1_I1_R1)
        //O:2 Port, P1, is created on
        //hierarchy module. )
endmodule
module H1 (P1);
output [1:0] P1;
    comp1    I1 (
        .R1(P1)    //O:2 I1.R1 is connected
        //through H1.P1 which in
        //turn is connected to I2.R2
    );endmodule

```

If you do not specify the commands highlighted in bold in the above example, the generated RTL would be as follows:

```

module top ();
wire [1:0] H1_I1_R1;
    comp2    I2 (
        .R2(H1_I1_R1)    //I:2
    );
    H1 U0_H1 (
        .H1_I1_R1_out(H1_I1_R1)    //O:2
    );
endmodule
module H1 (H1_I1_R1_out);
output [1:0] H1_I1_R1_out;
    comp1    I1 (
        .R1(H1_I1_R1_out)    //O:2
    );
endmodule

```


Such connections are shown in RTLHC, and other generators can query these connections like any other connections. The hierarchy/pin information is stored as additional information on these connections. Partitioning uses this information if the connection actually crosses the mentioned hierarchy, and uses the specified port name or creates its own port name if the information is not present.

Note:

ECO ports support is not available for IPXACT dump/restore.

Determining the Format and Directory to Save an Object by Using `save/saveall`

When you use the `save/save_autoconnect_rules` Tcl command, GenSys decides the format and the directory for that object by using certain algorithms.

Determining the Format of Saving an Object

When you run the `save` Tcl command, GenSys determines the format of the object being saved in the following manner:

1. By default, the `save` and `save_autoconnect_rules` Tcl commands save objects in their original format. That is, if an existing object has an IP-XACT 1.4 format, it will be saved in the same format.
2. When you create new objects, GenSys saves them in the format specified by the `set_default_save_format` Tcl command.
3. If you specify the `save -as tcl <dir-name>` command, GenSys saves the object in the Tcl format in the specified directory, `<dir-name>`.

Similarly, you can specify a format by using the `export_all` Tcl command. Running this command exports objects in the directory specified by this command with the specified format.

Determining the Directory to Save an Object

When you run the `save` Tcl command, GenSys determines the directory for the object being saved in the following manner:

1. If an existing object has been loaded, GenSys saves the object in its original location.
2. When you create new objects, they are saved in the directory specified by `set_default_save_dir` Tcl command.
3. If you specify the `save -as tcl <dir-name>` command, GenSys saves the object in a Tcl format in the specified directory.

4. If you have not run the [set_default_save_dir](#) Tcl command, GenSys saves the object in the directory specified by the [set_xml_dir](#) Tcl command.
5. If you have not executed the [set_xml_dir](#) Tcl command, GenSys saves the object in the current working directory.

Generating Schematic in Batch Mode

While running GenSys in batch mode, you can specify required settings for generating schematic by using various schematic-related Tcl commands.

To enable the schematic-related Tcl commands in batch mode, specify the `-batchSch` command-line option while running GenSys.

You can then specify the following Tcl commands to specify the required schematic-related settings:

- [incr_sch_add](#): Populates incremental schematic with specified ports and instances.
- [incr_sch_print](#): Prints incremental schematic.
- [incr_sch_set_zoom_factor](#): Sets a zoom factor for incremental schematic.
- [mod_sch_print](#): Prints modular schematic.
- [mod_sch_set_plane_view](#): Sets a plane view for the modular schematic.
- [mod_sch_set_zoom_factor](#): Sets a zoom factor for the modular schematic.
- [mod_sch_show](#): Populates and shows modular schematic.

Iterating Over a List or Table

When iterating over a list or a table, you should not delete the objects inside the list or table at the same time. For example, when deleting rows from a table, it is important to note that the table shrinks on each deletion, and therefore a simple approach as given below does not work:

```
foreach my $row (0 .. $table->nrows-1)
{
    delete...
}
```

A safer way is to do the deletion starting from the highest relevant row:

```
for ( my $row = $table->nrows-1; $nrows > -1 ;
      $nrows-- )
{
    delete...
```

Rerouting GenSys Connections through Via Points

Rerouting connections/nets is about ripping out existing nets between two end points and routing them separately for physical reasons.

GenSys provides the capability to reroute nets between a hierarchical source and hierarchical sink through the specified instances or instance ports. Such instances or instance ports through which rerouting occurs are known as *via points*.

When a net is rerouted through another instance, additional ports are created and a feed-through connection is made for that particular instance. However, no additional ports are created if you reroute nets through instance ports rather than instances.

This section contains the following topics related to rerouting GenSys connections through via points:

- [Use Model for Rerouting GenSys Connections](#)
 - [Modifying Existing Connections](#)
 - [Making New Connections](#)
 - [Rerouting SystemVerilog Interfaces](#)
- [Tcl Commands for Rerouting GenSys Connections](#)
- [Examples of Rerouting GenSys Connections through Via Points](#)

Use Model for Rerouting GenSys Connections

There are two use models for rerouting GenSys connections through via points:

- [Modifying Existing Connections](#)
- [Making New Connections](#)

Modifying Existing Connections

For modifying existing connections, specify the source and destination pins/ports, as the case may be, and the through instances or instance ports. The following applies to this use model:

1. New input and output ports are created for the specified instances. No port is created if you specify instance ports.
2. If there is a common hierarchy, the pins are preserved, as far as possible. For example, if there is a connection from top::instA::instB::instC.pinA to top.outA and the connection needs to be routed through top::instA::instD, the pins at top::instA are maintained. Relevant sections of the existing connection are deleted.

3. A direct feed-through net is created for all the through instances.

Making New Connections

For making new connections, specify the source and destination pins/ports, as the case may be, and the through instances or instance ports. The following applies to this use model:

1. A new connection is made via the through instances or instance ports.
2. New input and output ports are created for the specified instances. No port is created if you have specified instance ports.
3. A direct feed-through net is created for all the through instances or instance ports.

Rerouting SystemVerilog Interfaces

GenSys considers SystemVerilog interfaces as end points of connections to be rerouted.

GenSys reroutes the entire SystemVerilog interface across via instances, creates SystemVerilog interfaces, and maintains the interface abstraction during rerouting.

Use the `-intf_group` argument of the [reroute_connection](#) Tcl command to control interface modport to be uses in the via hierarchy.

Tcl Commands for Rerouting GenSys Connections

Use the following Tcl commands to reroute GenSys connections though via points:

- [reroute_connection](#)
- [reroute_fanout_node](#)
- [add_hier_connection](#)

Examples of Rerouting GenSys Connections through Via Points

Example 1

The following Tcl commands illustrate how to reroute connections.

```
# Use the hierarchy separator TCL command to provide
# the delimiter which will be used for specifying hierarchical
# design objects. This has a default value # of "::".
set_hierarchy_separator "::"

# This takes in an existing connection between two
# hierarchical end points (instance top::i0::i1::i2,
# pin P1 for this instance and range 7:6 for this pin
# and instance top::i0::i4, pin P2 on this instance and
# range 3:2 for this pin), delete the existing
```

```

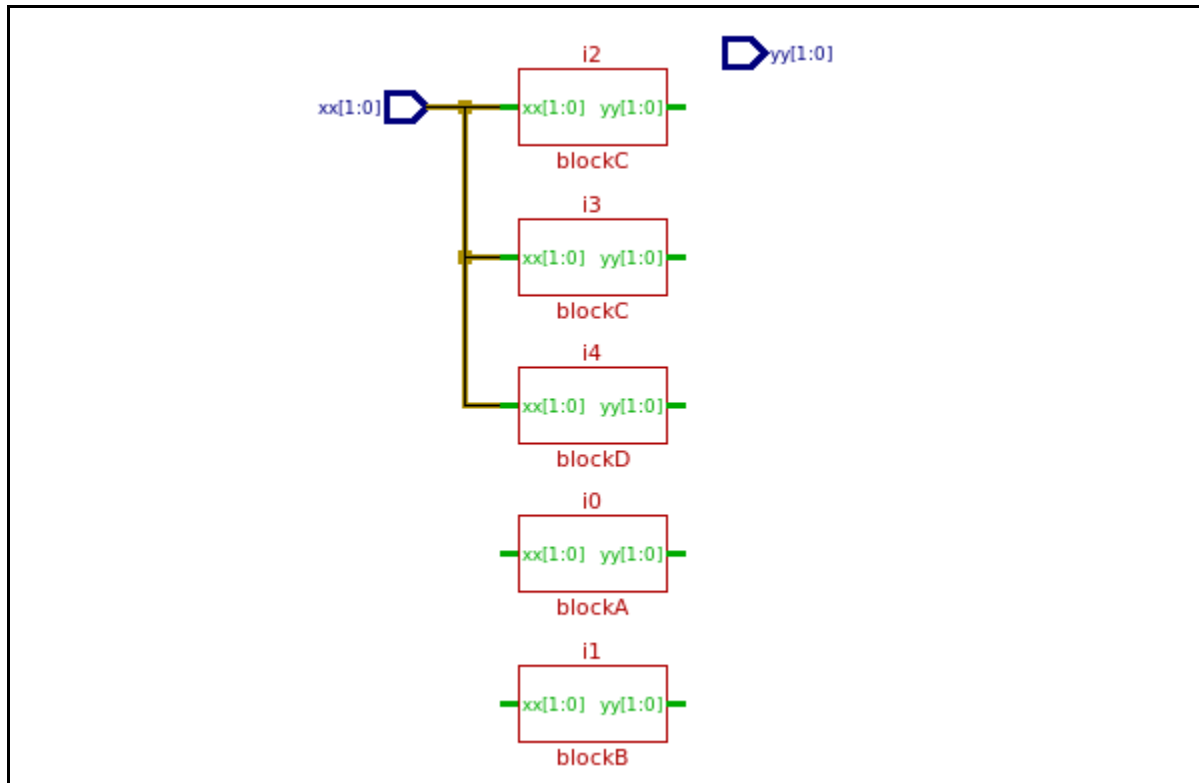
# connection between these two end points and route the
# connection through the given hierarchical instances.
reroute_connection -source top::i0::i1::i2::P1[7:6]
-dest top::i0::i4::P2[3:2] -through {top::i0::i5 top::i7 top::i8::i9}
reroute_connection -source top::i0::i1::i2::P3 \
-dest top::i5::i4::P4 \
-through {top::i0::i5 top::i7 top::i8::i9}
set_hierarchy_separator "."
reroute_connection -source top.i0.i1.i2.P1 \
-dest top.P2[7:4] \
-through {top.i0.i5 top.i7 top.i8.i9}

# Create a hierarchical connection between two
# different end-points (instance top::i0::i1::i2, pin
# P1 for this instance and range 7:6 for this pin and
# instance top::i0::i4, pin P2 on this instance and
# range 3:2 for this pin) while ensuring that the new
# connection passes through the provided via instances.
add_hier_connection -source top::i0::i1::i2::P1[7:6]
-dest top::i0::i4::P2[3:2]
-through {top::i0::i5 top::i7 top::i8::i9}
add_hier_connection
-source top::i0::i1::i2::P1[7:6]
-dest top::i0::i4::P2[3:2]

```

Example 2 - Rerouting Fan-Out Connections

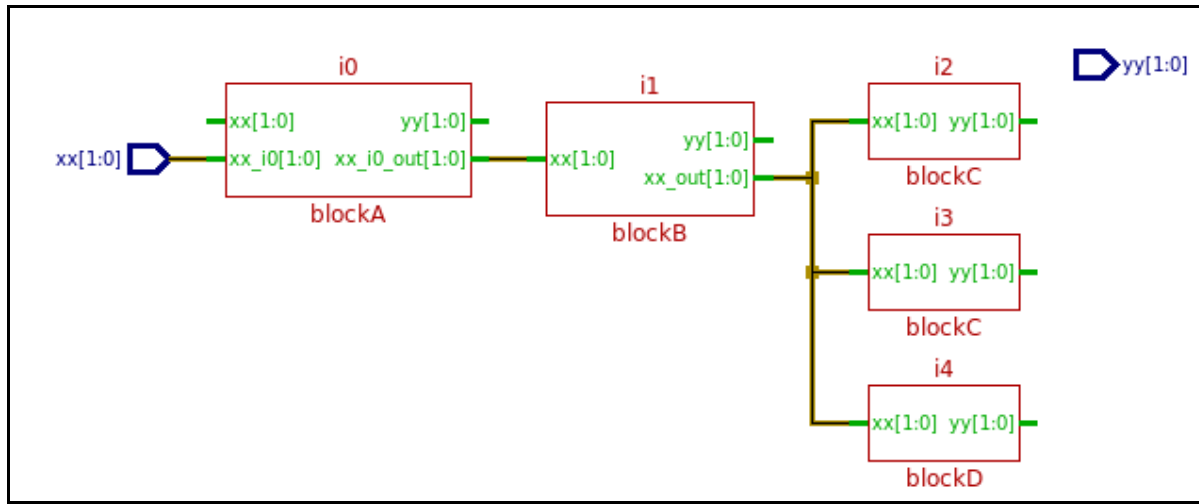
This example illustrate how to reroute fan-out connections. Consider the scenario shown in the following figure:



In the above scenario, to reroute the fan-out net through the i0 instance and the xx pin of the i1 instance, specify the following Tcl command:

```
reroute_connection -source top.xx \
-through {top.i0 top.i1.xx}
```

On running the above Tcl command, the following result is achieved:

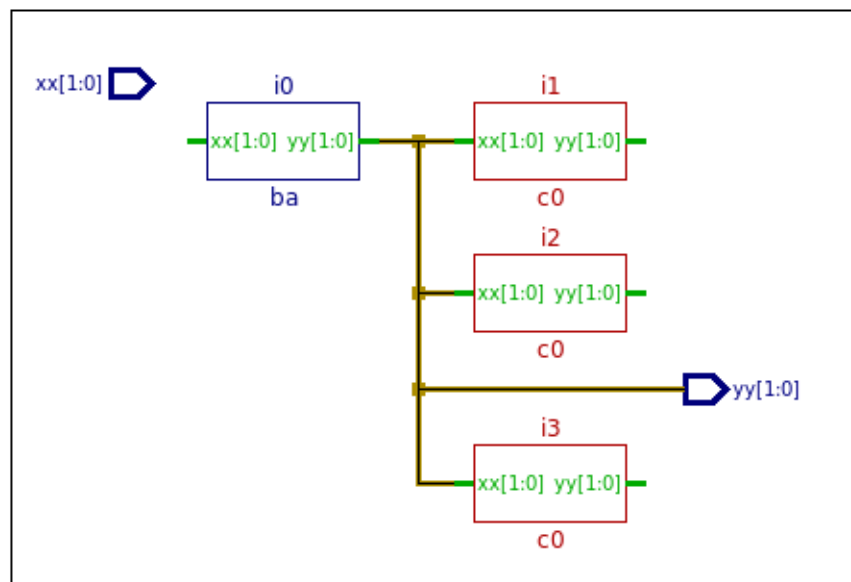


GenSys does the rerouting through pins, if specified. Else, GenSys creates new ports. The naming of ports is done based on the [set_default_port_name](#) command specification.

Example 3 - Using the reroute_fanout_node Tcl Command

This example shows the usage of the [reroute_fanout_node](#) Tcl command.

Consider the scenario shown in the following figure:

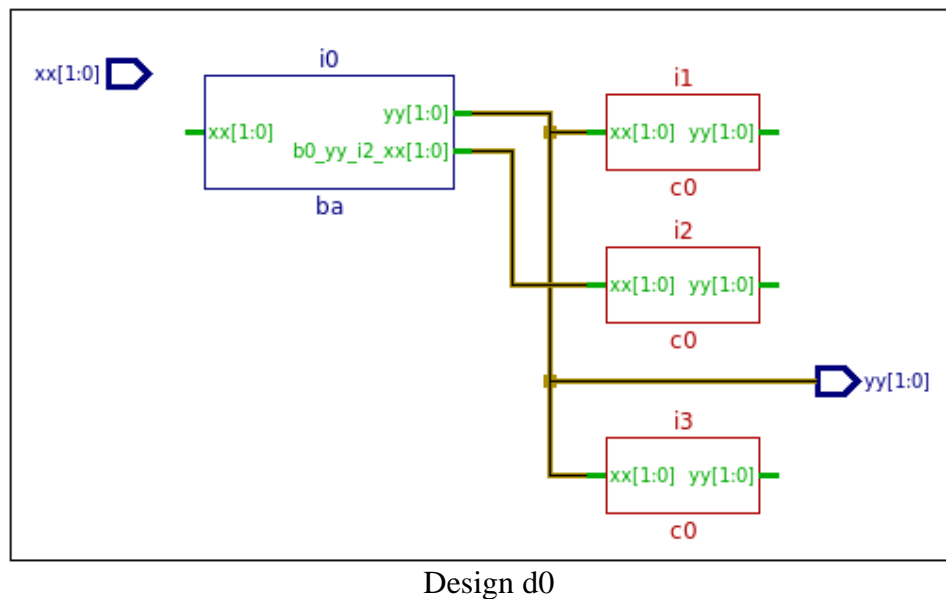


Design d0

Step 1: Run the following Tcl command:

```
reroute_fanout_node -source top.d0.i0.yy \
-dest top.d0.i2.xx
```

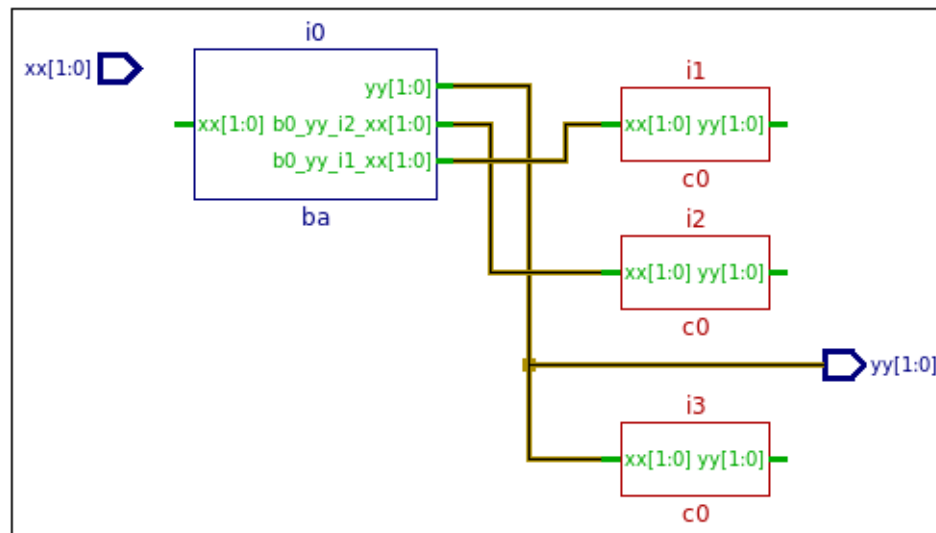
On running the above Tcl command, the following result is achieved:



Step 2: Run the following Tcl command:

```
reroute_fanout_node -source top.d0.i0.yy \
-dest top.d0.i1.xx
```

On running the above Tcl command, the following result is achieved:

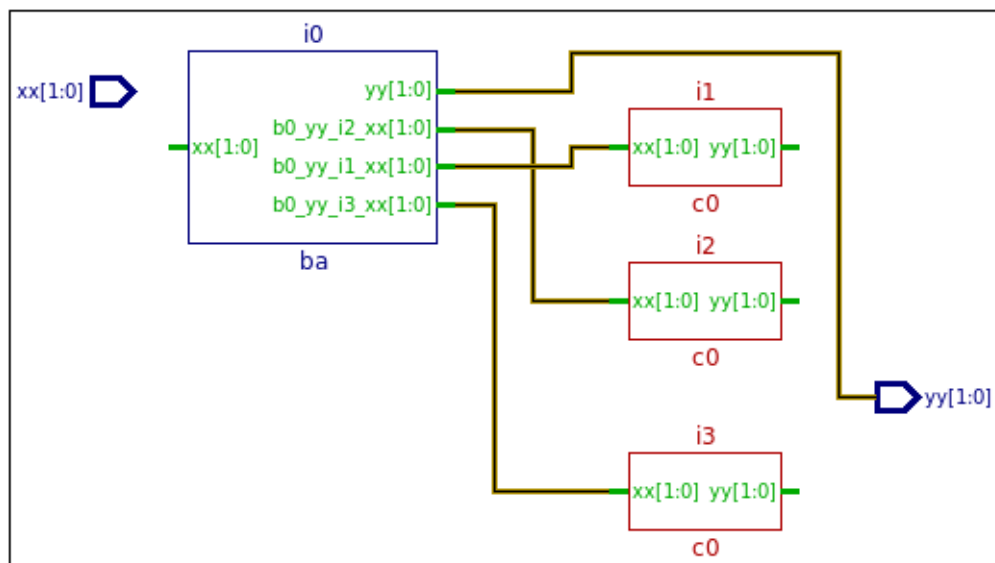


Design d0

Step 3: Run the following Tcl command:

```
reroute_fanout_node -source top.d0.i0.yy \
-dest top.d0.i3.xx
```

On running the above Tcl command, the following result is achieved:

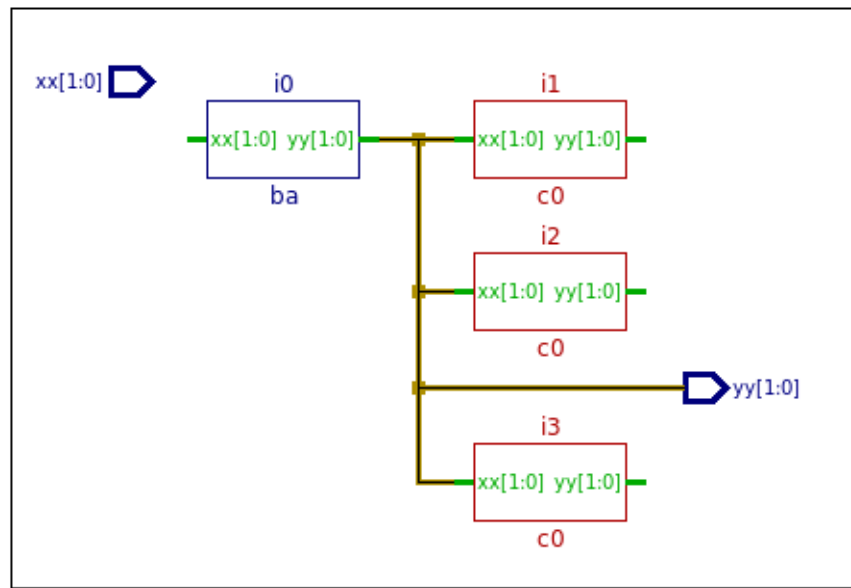


Design d0

Example 4 - reroute_fanout_node Tcl Command without -dest Argument

This example shows the usage of the `reroute_fanout_node` Tcl command when the `-dest` argument is not specified.

Consider the scenario shown in the following figure:

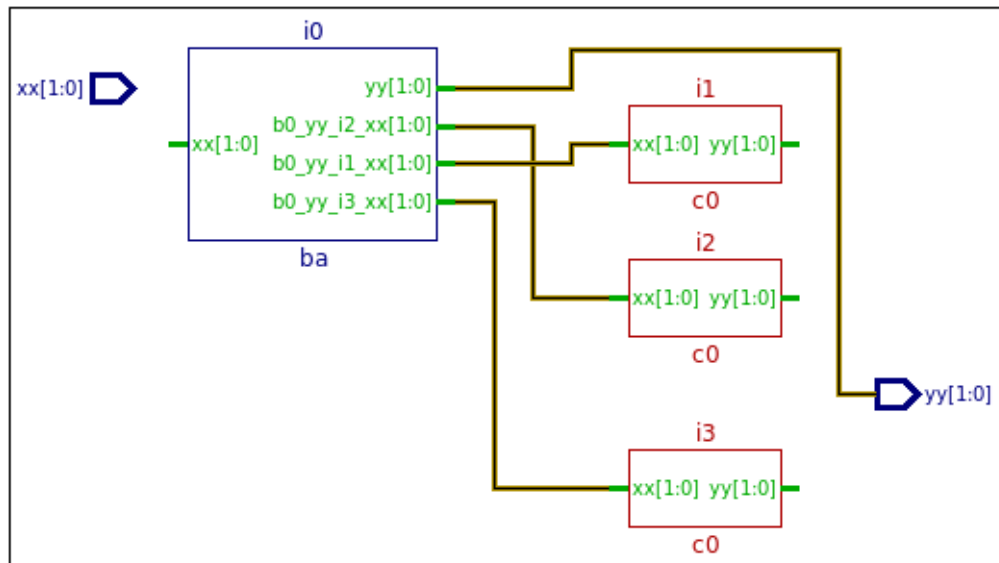


Design d0

Now consider that you run the following Tcl command:

```
reroute_fanout_node -source top.d0.i0.yy
```

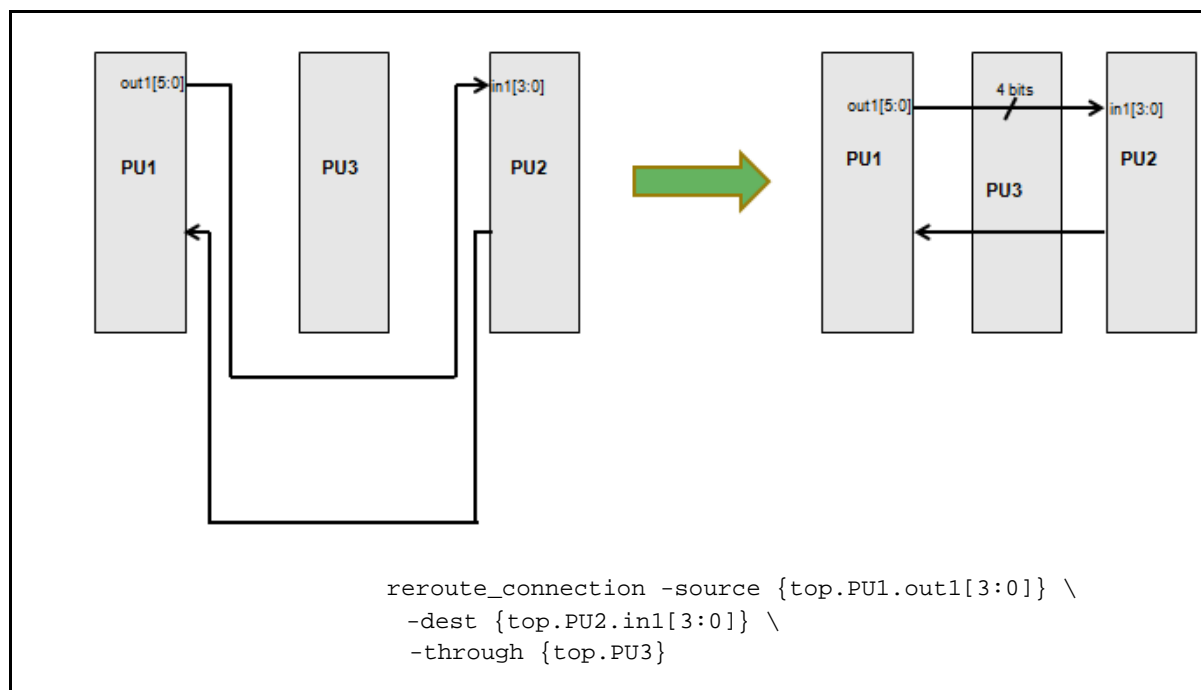
On running the above Tcl command, the following result is achieved:



Design d0

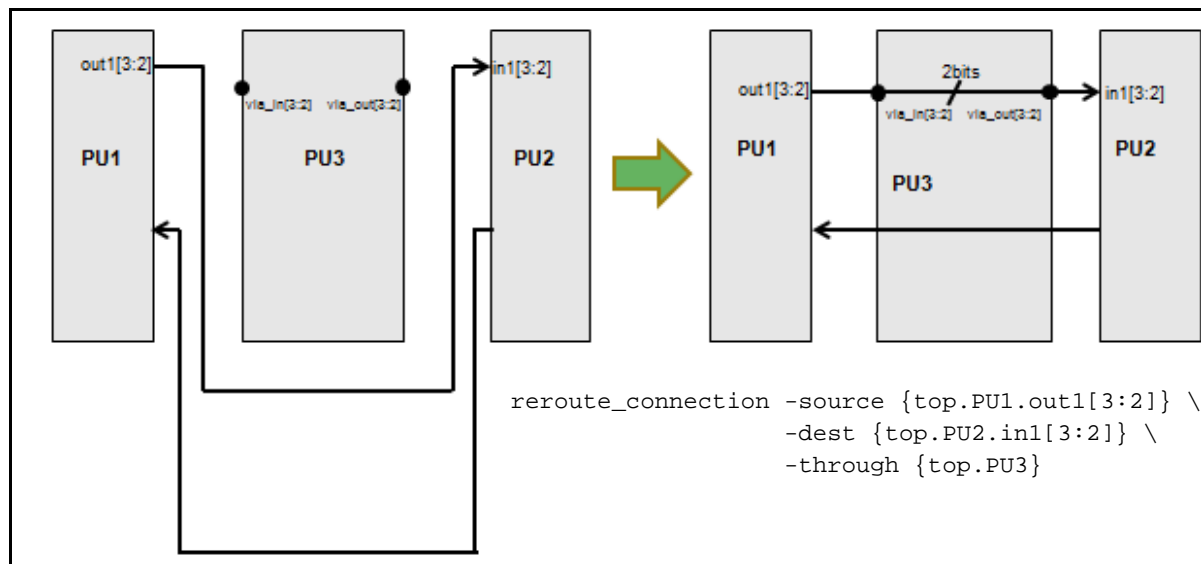
Example 5 - Rerouting Vector Part Select

The following figure shows the example of rerouting vector part select:



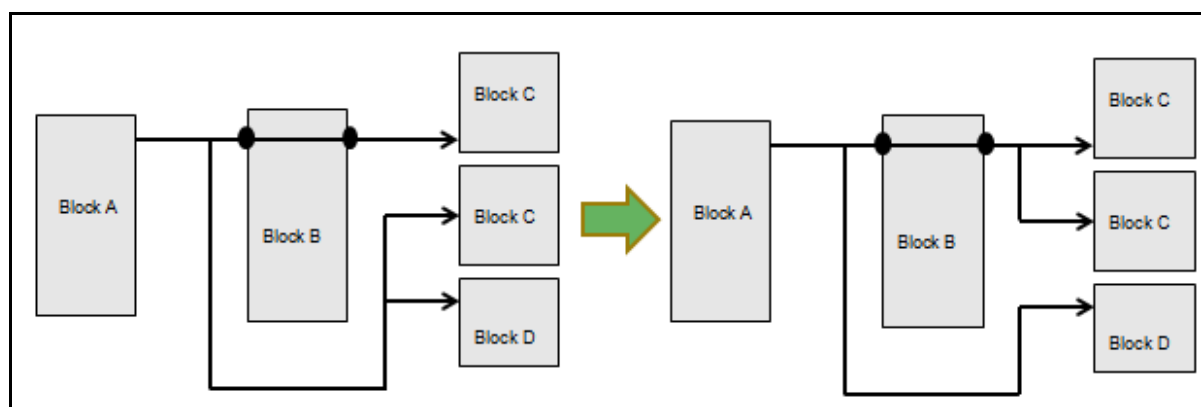
Example 6 - Rerouting Vector Part Select Using Existing Ports on Via Instances

The following figure shows the example of rerouting vector part select using existing ports on via instances:



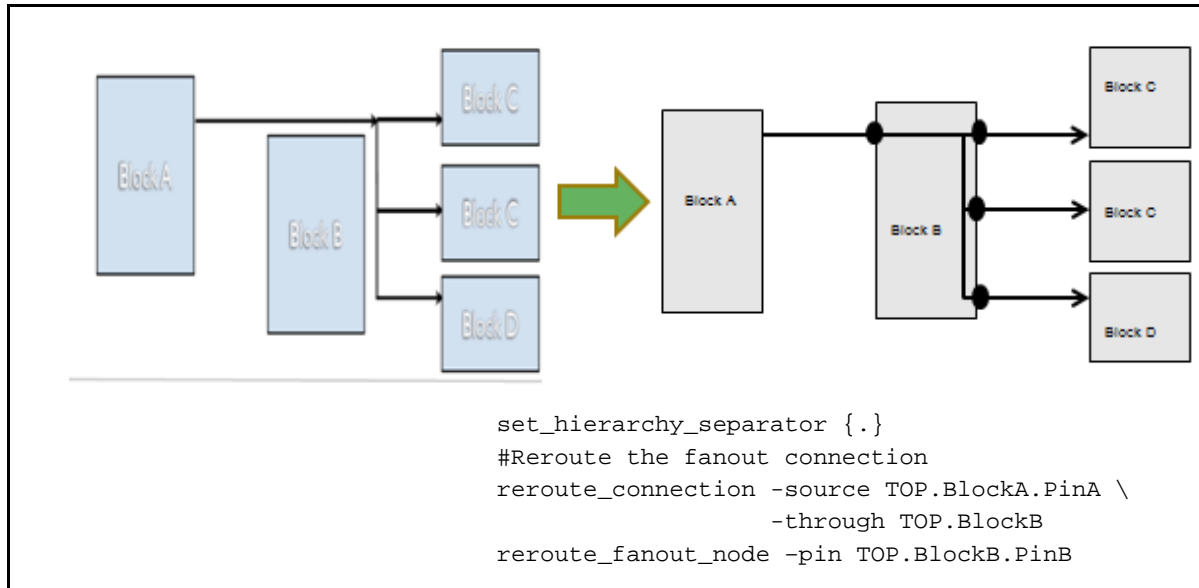
Example 7 - Rerouting using Existing Feed-through Connections on Via Instances

The following figure shows the example of rerouting using existing feed-through connections on via instances:



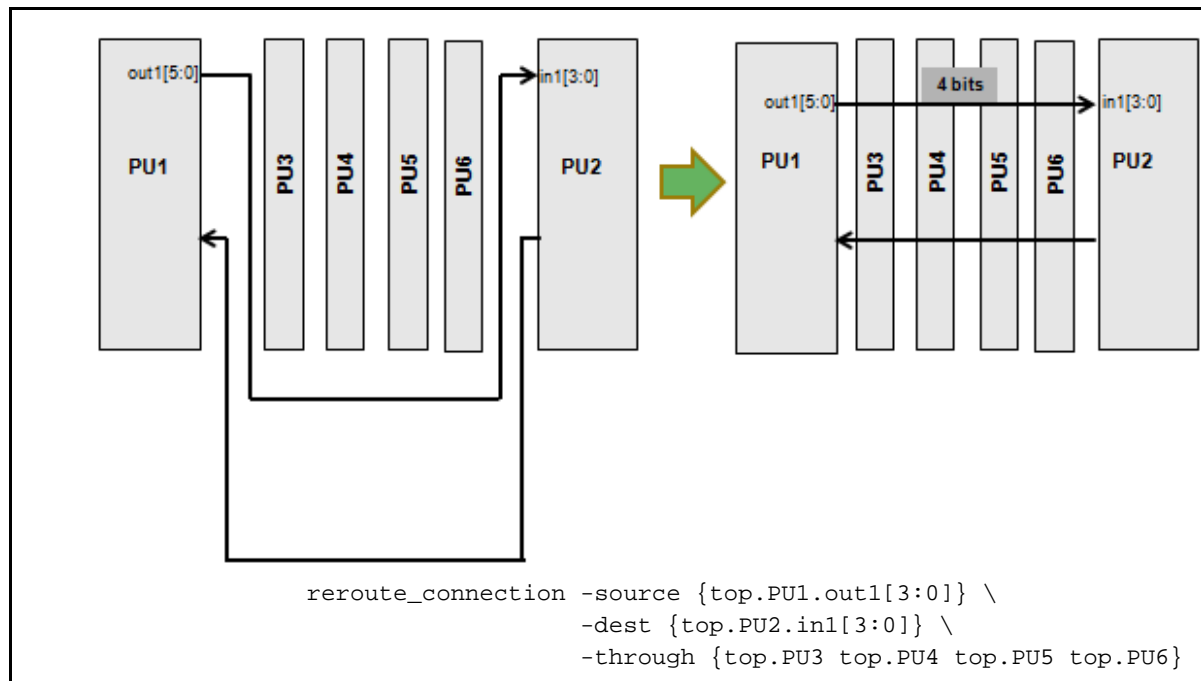
Example 8 - Fan-out Scenario with an Internal Fan-out Node

The following figure shows the example of a fan-out scenario with an internal fan-out node:



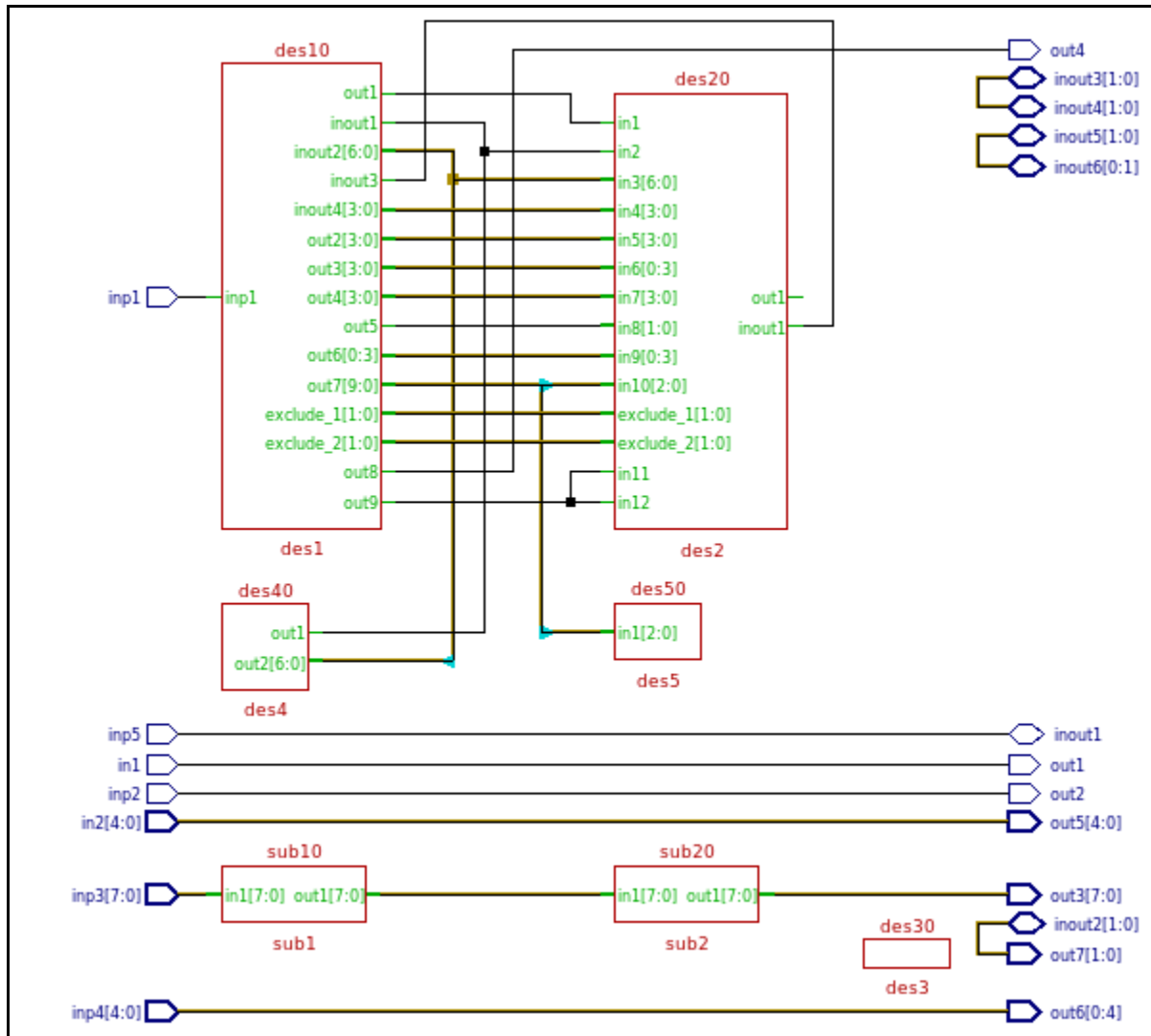
Example 9 - Reroute Vector Part-Select Through Multiple Via Instances

The following figure shows the example of reroute vector part-select through multiple via instances:



Example 10 - Rerouting Connections After Specifying Excluded Source and Destination Pins

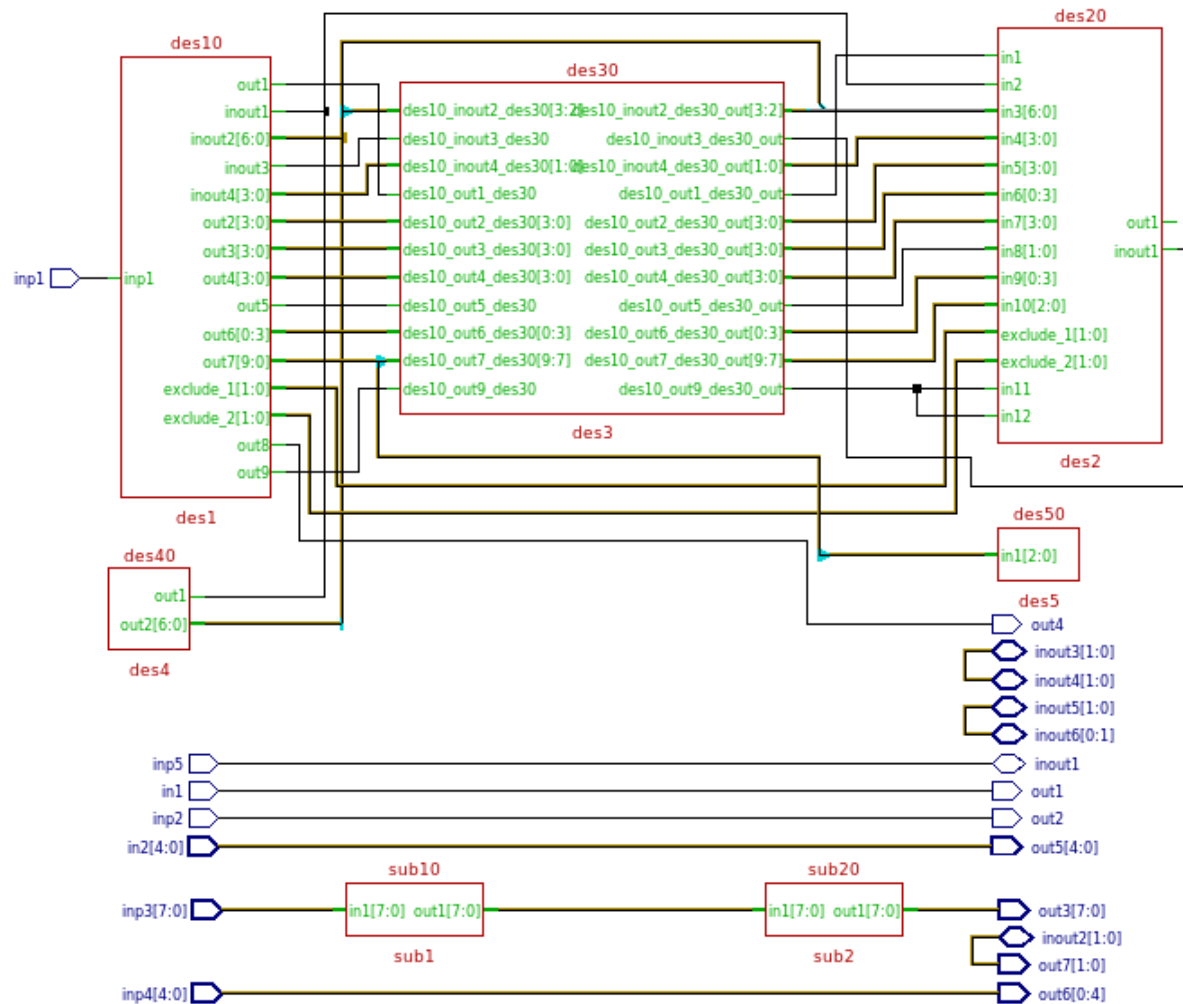
Consider the following design top:



In the above design, to reroute the connection from des10 (source) to des20 (destination) through des30 after excluding the exclude_1 and exclude_2 pins of each source and destination, specify the following Tcl command:

```
reroute_connection -source mid0::des0::des10 -dest mid0::des0::des20
-through top::mid0::des0::des30 -exclude_src { exclude_1 exclude_2 }
-exclude_dest { exclude_1 exclude_2 }
```

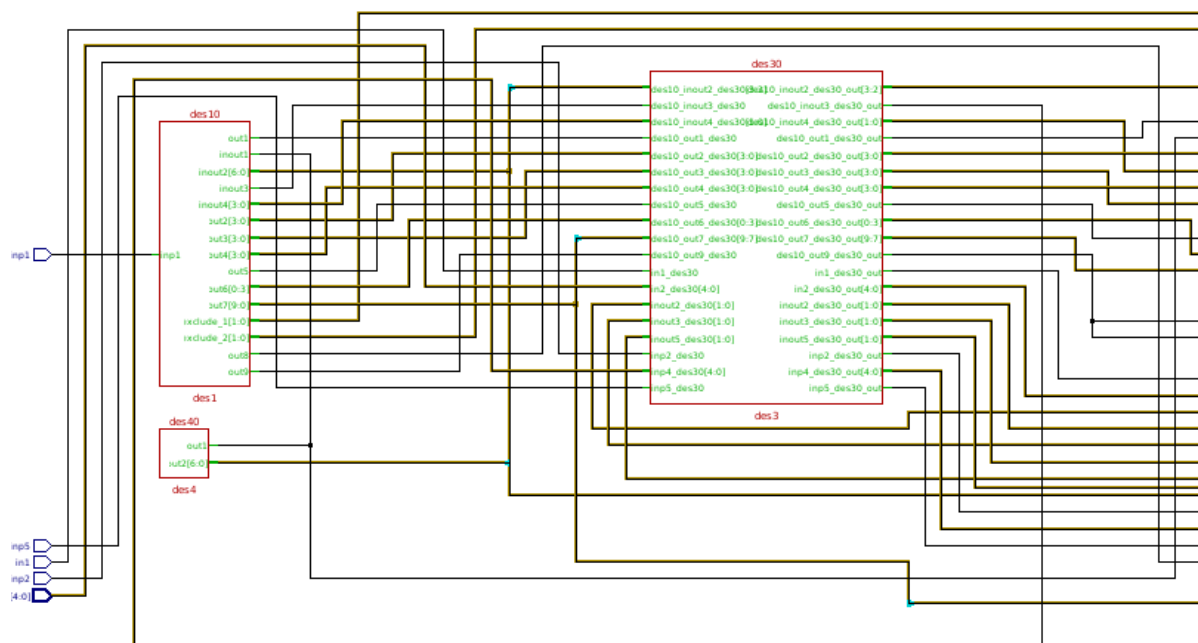
After running the above Tcl command, the connection in the design top changes to the following:



Now, for the above modified design, consider that you specify the following command:

```
reroute_connection -source mid0::des0 \
  -dest mid0::des0 -through top::mid0::des0::des30 \
  -source_port TRUE -dest_port TRUE
```

On specifying the above command, GenSys reroutes the feed-through connections of the des design. Here, `-source_port TRUE` means that for the source specified by `-source`, design ports are considered. Similarly, `-dest_port TRUE` means that for the destination specified by `-dest`, design ports are considered. Therefore, the modified rerouted connections appear as follows:



Setting GenSys Operating Modes

You can specify modes, such as RTL (default operating mode), IPXACT or MIXED, under which GenSys should operate.

Setting an Operating Mode

Specify the operating mode by using the `-use_model` argument of the [set_gensys_options](#) Tcl command:

```
set_gensys_options \
-use_model {RTL | IPXACT | | MIXED}
```

Working with the RTL Mode

In the RTL mode, the following arguments are set to the specified value:

- `-preserve_comments` argument of [set_rtl_option](#) is set to FALSE (default) unless it is explicitly set to TRUE earlier.
- `-preserve_pragmas` argument of [set_rtlimport_option](#) is set to TRUE (default) unless it is explicitly set to FALSE earlier.

- -preserve_configurations argument of [set_rtlimport_option](#) is set to FALSE (default) unless it is explicitly set to TRUE earlier.
- -load_from_library_browser argument of [set_rtlimport_option](#) is set to FALSE (default) unless it is explicitly set to TRUE earlier.
- -preserve_unsynth_constructs argument of [set_gensys_options](#) is set to FALSE (default) unless it is explicitly set to TRUE earlier.

The following holds true in the RTL mode:

- RTL views of all the IPs and sub-systems available.
- Save always happens in the MRS format.
- Export can happen in any format.
- Adding an instance or an interface of the IP-XACT type results in an error indicating that IP-XACT type of instance/interface cannot be added in a design when the use model is RTL.
- The [load](#) command reports errors when it loads any component of the IP-XACT type.

If you set any of the above arguments to a value that is not recommended in that particular use model, GenSys reports a warning and change the value to the correct value.

Working with the IP-XACT Mode

In this mode, the following holds true:

- The -preserve_pragmas argument of the [set_rtlimport_option](#) command is set to FALSE even if it is set to TRUE before.
- The -preserve_configurations argument of the [set_rtlimport_option](#) command is set to FALSE even if it is set to TRUE before.
- The -preserve_unsynth_constructs argument of the [set_gensys_options](#) command is set to FALSE even if it is set to TRUE before.
- The IP-XACT views of all the IPs and sub-systems are available.
- Save always happens in the IPXACT format without *Atrenta Vendor Extensions*.
- Export can happen in any format.
- Adding an instance or an interface of MRS type results in an error indicating that MRS type of instance/interface cannot be added in a design when the use model is IP-XACT.
- The [load](#) command reports errors when it loads any component of MRS type.

- If there is any *Atrenta Vendor Extension* present in IP-XACT load, an error message is reported indicating to regenerate IP-XACT without *Atrenta Vendor Extension*.
- If you perform a [load_rtl](#) in the IP-XACT mode, GenSys checks if there are some constructs that are not representable in IP-XACT. If yes, GenSys reports an error indicating to move to the RTL/MIXED flow.
- If you load an IP-XACT saved prior to GenSys 5.3 release, which has *Atrenta Vendor Extension*, GenSys does sanity checking and reports errors. If there are some constructs that are not representable in IP-XACT, GenSys reports errors indicating to move to the RTL/MIXED flow.
- Localparam's: Localparams in the exported IP-XACT are generated as modelParameters with the additional attribute spirit:configGroups="LOCAL_PARAM" which is used during IP-XACT read to restore it as a local param.
- Constants and Macros: Generating RTL by loading an IP-XACT results in macro/constant's replaced with their evaluated value.

Working with the Mixed Mode

In this mode, the following holds true:

- RTL and IP-XACT views of sub-systems and IPs are available.
- IP-XACT views cannot be re-structured.
- Save always happens in the MRS format.
- IP-XACT components or sub-systems are not changed in the mixed mode. Therefore, IP-XACT save for such components is not required.
- GenSys makes the IP-XACT views in this mode as read-only so that you cannot edit such sub-systems/components.
- GenSys does not allow connectivity between an IP-XACT interface and a Gensys interface (one that is in MRS format).

Auto Stop for Non Modified Hierarchies During RTL Design Read

When you specify the `-stop_untouched_hierarchy` command-line option, GenSys parses all design modification commands specified in Tcl files and populates a list of modified hierarchies. During RTL design read specified in the command-line Tcl files, GenSys applies stop to all hierarchies, which are not present in the list of modified hierarchies populated earlier.

With this option, you need to provide a Tcl file (or files) in the command line (either with `-tcl` option or directly). The Tcl file should capture design modification commands.

This command-line option should be used in batch mode, but can also be used in non-interactive UI mode when all design modifications are known and are captured in the Tcl files specified in the command line.

A `stop.f` file is generated in default report directory which can be used to subsequent runs with same commands. For subsequent runs, user can remove `-stop_untouched_hierarchy` option and use the `stop.f` file generated during first run.

For more information, see the following:

- [Example](#)
- [Usage Guidelines](#)

Example

This example shows the impact of using the `-stop_untouched_hierarchy` command-line option.

Consider the following Tcl file called `test.tcl`:

TCL Script - `test.tcl`

```
load_rtl -f filelist.f -enableSV
move_instance -instance demo_svunit1 \
-from demo_svunit_top::demo_svunit12 \
-to demo_svunit_top::demo_svunit12::demo_svunit2
move_instance -instance demo_svunit2_ireg \
-from demo_svunit_top::demo_svunit12::demo_svunit2 \
-to demo_svunit_top
```

Command-line Option

Specify the command-line option as follows:

```
gensys -batch -stop_untouched_hierarchy -tcl test.tcl
```

Result

In this example, the modified hierarchies are `demo_svunit_top`, `demo_svunit12`, and `demo_svunit2`. All other hierarchies are black boxed during design read.

Usage Guidelines

GenSys reports a warning message and does not perform the Auto Stop feature, if the following guidelines are not followed:

- When you specify the `-stop_untouched_hierarchy` command-line option, you must also specify a Tcl file.
- Do not use this command-line option in the interactive UI mode.
- Do not use a Tcl variable, such as `${variable}`, as a place holder for hierarchy and instance names in the GenSys command.
- Do not use a GenSys Tcl procedure affecting multiple hierarchies.
- Do not specify multiple `load_rtl` commands in the Tcl file

2

GenSys Tcl Commands

This chapter describes the Tcl commands currently available in GenSys. You can run these Tcl commands from the command prompt in the Message Window in the GenSys GUI.

add_attribute

Adds the attribute to the active object

Usage

```
add_attribute
  -name <attr-name>
  [ -description <description> ]
  [ -value <attr-value> ]
  [ -category <category> ]
  [ -validrange <range> ]
  [ -type INT | STRING | BOOL | HEX | ENUM <enum-list> ]
```

Description

The add_attribute command adds the specified attribute with its value, if specified, to the active object.

Arguments

The add_attribute command has the following arguments:

-name <attr-name>

Specifies the name (string) of the attribute. You can specify any name for the attribute apart from the following GenSys reserved attribute:

REQUIRE_REPAIR: REQUIRE_REPAIR is used to mark a connection as invalid.

-value <attr-value>

(Optional) Specifies the attribute value (string).

If you do not specify the -value argument, no value is attached to the attribute. You can also specify expressions (with parameters) as values.

-type

(Optional) Specifies the attribute type as one of the following:

Value	Indicates...
INT	Integer type attribute

Value	Indicates...
STRING	(Default) String type attribute
BOOL	Boolean type attribute
HEX	Hexadecimal type attribute
ENUM <enum-list>	Enumerated attribute <enum-list> is the comma-separated list of enum values enclosed in brackets

Note:

The attribute types given above are shown as ATTR_INT, ATTR_STRING, ATTR_BOOL, ATTR_BOOL, ATTR_HEX, or ATTR_ENUM in the GenSys GUI.

`-description <description>`

(Optional) Specifies the description of the attribute being added.

This argument also accepts unicode characters.

`-category <category>`

(Optional) Specifies the category of the attribute being added.

`-validrange <range>`

(Optional) Specifies the attribute range for Integer type and Hexadecimal type attributes.

See [Specifying Valid Range](#) for more details.

Examples

1. The following example adds the attribute with name s and value v:

```
add_attribute -name s -value v
```

2. The following example specifies the -validrange argument for attribute atrb1:

```
add_attribute -name atrb1 -value 4 -type INT -validrange 2-5,8
```

Here, the valid values of attribute A1 would be 2, 3, 4, 5, 8.

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case1 directory.

3. The following example creates a new design named des using the new command. Next, the add_attribute command creates an attribute named N3 of type ENUM with enum choices 0, 2, and 4 in the design des (which is the active object):

```
new design des
add_attribute -name N3 -type ENUM (0,2,4)
```

add_autoconnect_rule

Adds a new rule based on which connections are automatically generated

Usage

```
add_autoconnect_rule
  -name <rule_name>
  [ -src_inst <src-inst> ]
  [ -src_pin <src-pin> | -src_intf <src-intf> ]
  [ -src_lsb <src-lsb> ]
  [ -src_msb <src-msb> ]
  [ -src_property <src-property> ]
  [ -dst_inst <dst-inst> ]
  [ -dst_pin <dst-pin> | -dst_intf <dst-intf> ]
  [ -dst_lsb <dst-lsb> ]
  [ -dst_msb <dst-msb> ]
  [ -dst_property <dst-property> ]
  [ -consider_order <TRUE | FALSE> ]
  [ -active <TRUE | FALSE> ]
  [ -priority <priority> ]
  [ -align <LSB | MSB> ]
  [ -ignore_case <TRUE | FALSE> ]
  [ -plane <sch-plane-name> ]
  [ -tieoff
    {1|0| <decimal-number> | s<decimal-number> | 'b<binary-string> }
    | -open <TRUE | FALSE> | -export <TRUE | FALSE>
  ]
  [ -temporary <TRUE | FALSE> ]
  [ -hierarchical <TRUE | FALSE> ]
  [ -export_hier <TRUE | FALSE> ]
  [ -allow_pin_fanout <TRUE | FALSE> ]
  [ -allow_interface_fanout <TRUE | FALSE> ]
  [ -allow_loopback <TRUE | FALSE> ]
  [ -exclude_inst <instances-list> ]
  [ -exclude_pins <instance-pin-list> ]
  [ -exclude_intf <instance-interface-list> ]
  [ -exclude_component
    { -name <comp-name> -vendor <comp-vendor> -version
      <comp-version> -library <comp-library> }
  ]*
  [ -exclude_intflib
    { -name <intflib-name> -vendor <intflib-vendor>
```

```

    -version <intflib-version>
    -library <intflib-library> }
]*
[ - update ]
[ -strict <TRUE | FALSE> ]
[ -netname <string> ]
[ -export_name <exported-name> ]
[ -match_top_port <TRUE | FALSE> ]

```

Description

The `add_autoconnect_rule` command adds a new rule based on which connections are automatically generated. Use this command for pin-to-pin adhoc connections and not for interface connections.

When you run this Tcl command, GenSys adds a new row in the Autoconnect table of the currently loaded design.

Based on the rules executed, GenSys dumps the status of connections in the `AutoconnectReport.log` file. You can specify a different name for this file by using the [set_autoconnect_report_file](#) Tcl command.

Note:

Only one of tieoff, open, and export can be used in a single rule.

Handling Hierarchy Separators while Creating Automatic Connections

If a hierarchy separator (specified by the [set_hierarchy_separator](#) command) is used as a special character in a regular expression, use a backslash (\) before using that hierarchy separator.

For example, consider that you changed the instance hierarchy separator to '.' (dot) by using the following command:

```
set_hierarchy_separator {.}
```

Now consider that you need to export all the pins of the `usbf_unit_0` instance inside the `usb_group` subsystem. In this case, use a backslash while specifying the dot hierarchy separator, as shown in the following command (note the text in bold):

```
add_autoconnect_rule -name RULE_EXPORT_ALL -src_inst
{usb_group\.usbf_unit_0} -src_pin {.*} -export_hier TRUE
```

Creating Name-Based Auto Connections

If you want to connect the pins of the same name, specify the following Tcl command:

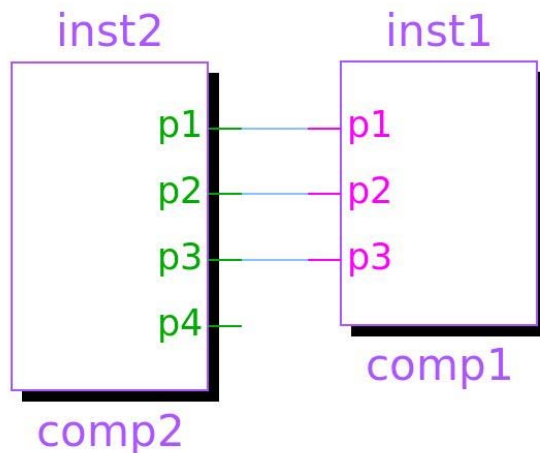
```
add_autoconnect_rule -name CONNECT_BY_NAME
```

In the above case, you do not need to specify any other argument to the `add_autoconnect_rule` Tcl command.

For example, consider `inst1` having the pins `p1(in)`, `p2(in)`, and `p3(in)` and `inst2` having the pins `p1(out)`, `p2(out)`, `p3(out)`, and `p4(out)`. In this case, if you specify `add_autoconnect_rule -name CONNECT_BY_NAME`, the following connections will be created:

`p1 -> p1`, `p2->p2`, and `p3 -> p3`

The above connections are shown in the following figure:



Arguments

The `add_autoconnect_rule` command has the following arguments:

`-name <rule-name>`

Specifies the rule name.

The specified rule name should be unique. If the specified name matches with an existing rule name, GenSys reports an error and does not insert that rule.

Note:

The rule name should contain only alphanumeric characters. Underscores are also allowed in the rule name.

`-src_inst <src-inst>`

(Optional) Specifies the name of the source instance.

You can use regular expressions with this argument.

`-src_pin <src-pin>`

(Optional) Specifies the name of the source pin.

You can use regular expressions with this argument.

`-src_intf <src-intf>`

(Optional) Specifies the name of the source interface.

You can use regular expressions with this argument.

`-src_lsb <src-lsb>`

(Optional) Specifies the lsb for the source side pin/port.

You can use an integer value and simple arithmetic expressions (+, -, *, /) with this argument. The captured text can also be accessed here using \$1...\$n, but regular expressions are not allowed in this argument.

`-src_msb <src-msb>`

(Optional) Specifies the msb for the source side pin/port.

You can use an integer value and simple arithmetic expressions (+, -, *, /) with this argument. The captured text can also be accessed here using \$1...\$n, but regular expressions are not allowed in this argument.

`-src_property <src-property>`

(Optional) Specifies the source property, which can be used for selecting some specific pins and interfaces.

You can specify any of the following properties:

Port Properties

attribute s	port_type	direction	lsb
msb	align	control_clock	control_reset

voltage	power_domain	interfaces	net_name
net_prefix	isPortTLMType	description	Short_Name
Comment	Visibility	default_value	
Interface Properties			
IsMirrored	interfaceLib_Name	control_clock	control_reset
voltage	power_domain	address_bus_width	access_size
open	description	Short_Name	Visibility

The above property names are not case-sensitive. For example, specifying MSb and MSB is correct.

GenSys evaluates expressions specified by this command based on priorities.

Note:

The following operators are supported to join two property expressions:

&&		!
----	--	---

Note:

The following operators are supported inside expressions:

==	!=	>=	>	<=	<
----	----	----	---	----	---

For details on using this argument, see [Examples](#).

`-dst_inst <dst-inst>`

(Optional) Specifies the name of the destination instance.

You can use regular expressions with this argument.

`-dst_pin <dst-pin>`

(Optional) Specifies the name of the destination pin.

You can use regular expressions with this argument.

`-dst_intf <dst-intf>`

(Optional) Specifies the name of the destination interface.

You can use regular expressions with this argument.

`-dst_lsb <dst-lsb>`

(Optional) Specifies the lsb for the destination side pin/port.

You can use an integer value and simple arithmetic expressions (+, -, *, /) with this argument. The captured text can also be accessed here using \$1...\$n, but regular expressions are not allowed in this argument.

`-dst_msb <src-msb>`

(Optional) Specifies the msb for the destination side pin/port.

You can use an integer value and simple arithmetic expressions (+, -, *, /) with this argument. The captured text can also be accessed here using \$1...\$n, but regular expressions are not allowed in this argument.

`-dst_property <dst-property>`

(Optional) Specifies the destination property, which can be used for selecting some specific pins and interfaces.

Usage of this argument is similar to the `-src_property` argument of this Tcl command.

`-consider_order <TRUE | FALSE>`

(Optional) Specifies whether the order should be considered while making connections from a rule. This switch is effective for specific type of regular expressions, such as ([a|b|c] or [0-5]).

The default value is TRUE. If this argument is set to FALSE, no order is considered for connection. However, if this argument is set to TRUE, connections are made in the same order as present in the regular expression.

`-active <TRUE | FALSE>`

(Optional) Specifies whether particular rule is active or not. By default all the rules are active. To disable any rule set this argument to FALSE. If any rule is not active then that rule will not run while executing [run_autoconnect](#) Tcl command.

`-priority <priority>`

(Optional) Specifies the rule priority.

This value determines the order in which rules are executed. A high priority rule is executed first.

For example, consider that R1 rule has the priority 10 and the R2 rule that has the priority 20. In this case, GenSys first runs the R1 rule and then the R2 rule.

Rules for which no priority is defined are run last. The order in which such rules should run is determined by the order in which they are specified in the Autoconnect table of the design.

`-align <LSB | MSB>`

(Optional) Specifies the alignment of the connections being made by this rule.

The default value of this argument is MSB.

`-ignore_case <TRUE | FALSE>`

(Optional) Specifies case sensitive or insensitive search.

Specifies case sensitive or insensitive search. The default value of this argument is FALSE, which means the search is case sensitive while considering pins, interfaces, and instances. Set the value to TRUE to execute case insensitive search.

`-plane <sch-plane-name>`

(Optional) Specifies the plane name for a particular rule.

For each connection, one end is the driver side and the other end is the driven side. When you specify this argument, the driven side of all connections generated by running an auto-connect rule is assigned the specified plane.

If you do not specify this argument, GenSys considers the rule name as the plane name.

`-tieoff {1 | 0 | <decimal number> | s<decimal number> | 'b<binary string>`

(Optional) Specifies the tieoff value for the selected pins or ports.

For example, consider the following command:

```
add_autoconnect_rule -name rule1 -src_pin ^p.*$ -tieoff 0
```

When you run the above command, all ports matching the specified regular expression are assigned the tieoff value of 0.

`-open <TRUE | FALSE>`

(Optional) Specifies if the specified pins or interfaces should be made intentionally open.

Set this argument to FALSE if you do not want them to be intentionally open.

`-export <TRUE | FALSE>`

(Optional) Specifies if all the selected pins or interfaces should be exported to the next upper design.

Set this argument to FALSE if you do not want to perform this export operation.

`-temporary <TRUE | FALSE>`

(Optional) Specifies if all connections should be made temporary tieoff (if the `-tieoff` argument is specified) or temporary open (if the `-open` argument is specified).

Note:

You can specify this argument only with the `-tieoff` or `-open` arguments.

`-hierarchical <TRUE | FALSE>`

(Optional) Specifies if all the auto-connect rules should be run on sub-designs also.

Set this argument to FALSE if you do not want the rules to run on sub-designs.

`-export_hier <TRUE | FALSE>`

(Optional) Specifies whether the pin or interface mentioned in the rule should be exported across multiple levels of hierarchy to the top design.

The default value is FALSE. Set this argument to TRUE to export pin or interface till top design.

Refer to [Example of the -export_hier Argument](#).

`-allow_pin_fanout <TRUE | FALSE>`

(Optional) Specifies whether particular auto connect rule may allow a port to drive multiple ports.

The default value is FALSE. Set this argument to TRUE to allow a port to drive multiple ports.

Refer to [Example of the -allow_pin_fanout Argument](#).

`-allow_interface_fanout <TRUE | FALSE>`

(Optional) Enables an interface to drive multiple interfaces while generating automatic connections.

By default, this argument is set to FALSE, and an interface is allowed to drive only one interface. Set this argument to TRUE to enable an interface to drive multiple interfaces.

`-allow_loopback <TRUE | FALSE>`

(Optional) Enables the *Automatic Connect* feature to permit loopback connections.

By default, this argument is set to FALSE and loopback connections are not generated.

`-exclude_inst {instances-list}`

(Optional) Specifies a space-separated list of instances that you want to exclude from the auto-connect rule.

You can use regular expressions with this argument.

The following example shows the usage of this argument:

```
add_autoconnect_rule -name rule1 -src_pin p2
-dst_inst inst.* -dst_pin p.* -exclude_inst inst2
add_autoconnect_rule -name rule2 -src_pin p2
-dst_inst inst.*::in.* -dst_pin p.* -exclude_inst
inst1::inst11.*
```

`-exclude_pins {instance-pin-list}`

(Optional) Specifies a space-separated list of instance pins that you want to exclude from the auto-connect rule.

You can use regular expressions with this argument.

You should specify instance pins in the following format:

<instance-name>@<pin-name>

The following example shows the usage of this argument:

```
add_autoconnect_rule -name rule1 -src_pin p2
-dst_inst inst.* -dst_pin p.*
-exclude_pins { inst1@p2 inst2@p2 }
```

In the above example, the p2 pin of the inst1 instance and the pin p2 pin of the inst2 instance is not considered for creating connection when the rule1 is run. You can achieve the same effect by using regular expressions, for example:

`-exclude_pins {inst[1|2]@p2}`

`-exclude_intf {instance-interface-list}`

(Optional) Specifies a space-separated list of interfaces that you want to exclude from the auto-connect rule.

You can use regular expressions with this argument.

You should specify interfaces in the following format:

<instance-name>@<interface-name>

The following example shows the usage of this argument:

```
add_autoconnect_rule -name rule2 -src_intf .*
-dst_inst inst.* -dst_intf .*
-exclude_intf { m0 inst1@m0 inst1@m1 inst2@m3 }
```

In the above example, the m0 design interface, the m0 interface of the inst1 instance, the m1 interface of the inst1 instance, and the m3 interface of the inst2 instance is not considered while creating connections when the rule2 rule is run. You can achieve the same effect by using regular expressions, for example:

```
-exclude_intf {m0 inst1@m[0|1|3]}-exclude_component { -name <comp-name>
-vendor <comp-vendor> -version <comp-version> -library <comp-library> }
```

(Optional) Specifies the VLNV of master of component instances that you do not want to consider while creating connections.

You can specify this argument multiple times in a single add_autoconnect_rule command.

Note:

Usage of regular expressions is not allowed for this argument.

```
-exclude_intflib { -name <intflib-name> -vendor <intflib-vendor> -version
<intflib-version> -library <intflib-library> }
```

(Optional) Specifies the VLNV of the parent of interfaces that you do not want to consider while creating connections.

You can specify this argument multiple times in a single add_autoconnect_rule command.

Note:

Usage of regular expressions is not allowed for this argument.

```
-update
```

(Optional) Updates an existing auto-connect rule.

Consider the following example in which you create the r1 rule:

```
add_autoconnect_rule -name r1 -src_pin p1.*
-dst_pin p2.*
```

Now, if you want to change the destination pin for the above rule, specify the new value in the -dst_pin argument for the r1 rule and specify -update, as shown in the following example:

```
add_autoconnect_rule -name r1 -dst_pin p3.* -update
```

In this case, the value of source pin remains the same as mentioned in the first add_autoconnect_rule command for the r1 rule. However, the value of the destination pin is updated from p2.* to p3.*.

Note:

This argument updates all details of an auto-connect rule, except the rule name.

`-strict <TRUE | FALSE>`

(Optional) Specifies if automatic connections should be generated only between ports/pins of the same bit width.

By default, this argument is set to FALSE and GenSys connects pins or ports even if their bit width does not match.

Set this argument to TRUE so that GenSys does not connect pins or ports of different bit widths.

`-netname <string>`

(Optional) Sets the name for connections generated by the autoconnect feature.

When you specify a string to this argument, GenSys appends an index number in the `_<index-num>` format to the string for each automatically-generated connection. For details, see [Example 2](#).

`-export_name <exported-name>`

(Optional) Creates an exported port with the name specified by this argument. This will have highest precedence over the default port naming convention and the normal counter based naming provided it does not lead to multiple driver scenarios.

`-match_top_port <TRUE | FALSE>`

(Optional) By default, this argument is set to FALSE. If you set it to TRUE, it restricts automatically exporting unconnected pins from specified/matched source instance to top-level, if the instance pin names match exactly with the top-level port.

For details, see [Example 3](#).

Examples

Example 1

The following example shows the usage of the `-src_property` and `-dst_property` arguments of the `add_autoconnect_rule` Tcl command:

```
add_autoconnect_rule -src_intf intf1 \  
-src_property "interfacelib_Name master" -dst_intf intf2 \  
-dst_property "type SLAVE" -name rule2  
  
add_autoconnect_rule -src_pin lop1.* \  
-src_property "port_type == (CORE_INPUT|DFT) && direction IN" \  
-dst_pin lop2.* -dst_property "port_type DFT && direction == OuT" -name  
rule2
```

```

add_autoconnect_rule -src_pin r1.* \
-src_property "port_type != CORE_INPUT  && lsb 0 || msb 100 "
-dst_pin r2.* -dst_property " direction == OuT && msb != 200" \
-name rule8

```

Example 2

Consider that you run the following Tcl commands:

```

new component comp
add_port -name p1 -direction IN
add_port -name p2 -direction IN
add_port -name p3 -direction OUT
new design dsgn2
add_port -name p1 -direction IN
add_port -name p2 -direction IN
add_port -name p3 -direction OUT
add_instance -name inst -master comp
add_autoconnect_rule -src_pin p.* -dst_inst inst
-dst_pin p.* -name rule1 -name rule1 -netname NNN
run_autoconnect
run GenerateVerilog

```

On running the above commands, the following Verilog file is generated:

```

module dsgn2 (p1,
              p2,
              p3);
input        p1;
input        p2;
output       p3;
// inst wires
wire         NNN_2;
wire         NNN_0;
wire         NNN_1;
    comp     inst ( //Instance
                    .p1(NNN_0), //In[1]
                    .p2(NNN_1), //In[1]
                    .p3(NNN_2)  //Out[1]
                  );
    assign NNN_0 = p1;
    assign NNN_1 = p2;
    assign p3 = NNN_2;
endmodule

```

In the above generated Verilog, notice the name of the automatically-generated connections (in bold).

The NNN string, which is specified by the `-netname` argument of the `add_autoconnect_rule` command is appended with a unique index number to name multiple connections.

Example 3

The following specification would export all those unconnected pins of matched instances to the top-level only if a port with the same name exists on top:

```
add_autoconnect_rule -src_inst {.*} -src_pin {.*}  
-export_hier {TRUE} -allow_pin_fanout {TRUE}  
-match_top_port {TRUE} -name new_pull_up_rule
```

The following specification would export all the unconnected pins of matched instances to the top level without checking for the port with the same name on top:

```
add_autoconnect_rule -src_inst {.*} -src_pin {.*}  
-export_hier {TRUE} -allow_pin_fanout {TRUE}  
-match_top_port {FALSE} -name pull_up_rule
```

Cases of Autoconnect Rule Run

The following points discuss various cases of Autoconnect rule run:

- Case 1

When one side has only one pin/interface that was selected for connection, and the other side has one or multiple (may M) pins/interface that were selected.

In this case, M connections are generated by the autoconnect rule. That one pin is connected to all other pins that got selected on the other side.

- Case 2

When both sides have got multiple pins/interfaces selected.

Consider that one side contains M selections, other side contains N selections, and K is the minimum of M and N. (M is not equal to N).

In this case, K connections are generated.

If the `CONNECT_ORDER` attribute is present on the selected pins/interfaces, connections are generated by using them. That is, pins/interface of the source side are connected to the pin/interface on the destination side that has the same connect order attribute.

If enough CONNECT_ORDER attributes are not present, some connections (say K1 connections) are generated by using this equal CONNECT_ORDER, and the rest of the connections (that is K - K1 connections) are generated by simply selecting a pin from the source side that is still not connected and the first pin from the destination side that has still not connected. GenSys continues this way until K1 connections are made.

If there is not a single pin/interface with the CONNECT_ORDER attribute, K connections are generated by choosing the first K pins /interfaces from the source side and first K pins/interfaces from the destination side and connecting them one by one.

For adding the CONNECT_ORDER attribute on design ports, user can simply use commands after selecting the pin/interface.

```
add_attribute -name CONNECT_ORDER -value <value>
```

Example:

```
add_attribute -name CONNECT_ORDER -value 2
```

However, for giving the CONNECT_ORDER attribute to instances pins/interfaces, you should first select them by using the following Tcl commands:

- select_terminal -instance <instance name> -pin <pin-name>

Use the above command to select instances pins.

- select_interfaceinst -instance <instance name> -interface <interfaceInst-name>

Use the above command to select instance interfaces.

You can then use the same Tcl command add_attribute to add the CONNECT_ORDER attribute to the selected pins/interfaces.

- Case 3

When both sides have same number of pins/interfaces selected (strictly greater than one on both sides, that is, $M = N$)

If instances that get selected on both sides are exactly same, GenSys makes chained connections if each selected instance has some pins that can act as a driver and some pins that can act as driven.

Chaining means that instances are connected in the following manner:

I1 --- >> I2 -- >> I3 -- >> I4 --- ... so on IN .. connected back to I1

Therefore, driver pin of I1 drives the driven pin of I2, driver pin of I2 drives the driven pin of I3, and so on. Finally, the driver pin of IN connects to the first chaining instance I1 driven pin.

Note:

Multiple chains can be made by a single rule.

In addition, directions of all pins selected on one side should be IN or INOUT. It can be any side, source or destination. Similarly, directions of all pins selected on other side should be OUT or INOUT. It can be any side, source or destination.

If any of the above condition fails, GenSys does not make chained connections and makes connections as discussed in case 2 above.

The sequence in which instances are connected to each other depends on the CONNECT_ORDER attribute specified for them.

GenSys arranges instances in an increasing CONNECT_ORDER value. If some or all instances do not have the CONNECT_ORDER attribute on them, GenSys puts them at the end of the instances list in order in which you select them.

Note:

You can also set the CONNECT_ORDER attribute on instances also.

Now consider that arrangement of instances is something like below:

(I1 --- >> I2 -- >> I3 -- >> I4 --- ... so on (IN nth instance) .. connected back to I1)

In this case, GenSys connects those pins of I1 with I2 that have the same connect order on them. If pins do not have the CONNECT_ORDER attribute on them, GenSys connects the pins sequentially one by one, taking one pin from one side and other from the destination list. Same goes for all instances.

Example of Chaining

Consider the following commands in a Tcl file:

```
new component compl
add_port -name p1 -direction OUT
add_port -name p2 -direction IN

add_port -name p11 -direction OUT
add_port -name p22 -direction IN
save

new design dsgn2

add_instance -name inst1 -master compl
add_attribute -name CONNECT_ORDER -value 1

add_instance -name inst2 -master compl
add_attribute -name CONNECT_ORDER -value 2

add_instance -name inst3 -master compl
add_attribute -name CONNECT_ORDER -value 3

add_instance -name inst4 -master compl
```



```
add_attribute -name CONNECT_ORDER -value 4

add_instance -name inst5 -master compl
add_attribute -name CONNECT_ORDER -value 5

add_instance -name inst6 -master compl
add_attribute -name CONNECT_ORDER -value 6

add_instance -name inst7 -master compl
add_attribute -name CONNECT_ORDER -value 7

select_terminal -instance inst1 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst1 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst1 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst1 -pin p22
add_attribute -name CONNECT_ORDER -value 2

select_terminal -instance inst2 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst2 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst2 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst2 -pin p22
add_attribute -name CONNECT_ORDER -value 2

select_terminal -instance inst3 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst3 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst3 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst3 -pin p22
add_attribute -name CONNECT_ORDER -value 2

select_terminal -instance inst4 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst4 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst4 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst4 -pin p22
add_attribute -name CONNECT_ORDER -value 2

select_terminal -instance inst5 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst5 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst5 -pin p11
add_attribute -name CONNECT_ORDER -value 2
```

```

select_terminal -instance inst5 -pin p22
add_attribute -name CONNECT_ORDER -value 2

select_terminal -instance inst6 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst6 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst6 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst6 -pin p22
add_attribute -name CONNECT_ORDER -value 2

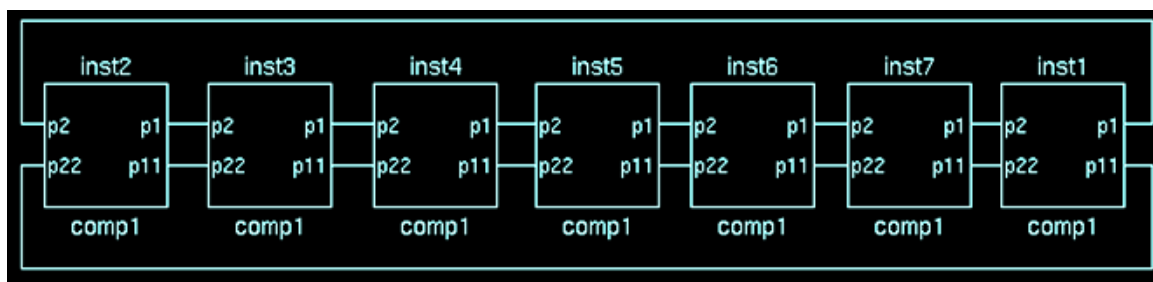
select_terminal -instance inst7 -pin p1
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst7 -pin p2
add_attribute -name CONNECT_ORDER -value 1
select_terminal -instance inst7 -pin p11
add_attribute -name CONNECT_ORDER -value 2
select_terminal -instance inst7 -pin p22
add_attribute -name CONNECT_ORDER -value 2

add_autoconnect_rule -src_inst inst.* -src_pin p1.*
-dst_inst inst.* -dst_pin p2.* -name rule1

run_autoconnect

```

Chaining in this case occurs, as shown in the following schematic:



Example of the -export_hier Argument

In this example, since the -export_hier argument is set to true, the command exports p1 across multiple hierarchies to top design inst1.

```

add_autoconnect_rule -src_inst inst1::ggg::hhh -src_pin p1
-export TRUE -name rr2 -export_hier true

```

Example of the -allow_pin_fanout Argument

In this example, the source instance contains a single port port2 and the destination instance contains two ports, port3 and port3a. When the the -allow_pin_fanout argument is set to false, it restricts the add_autoconnect_rule to make only one-to-one connection between port2 (of source instance) and port3 (of destination instance).

```
add_autoconnect_rule -src_inst inst2 -src_pin port2.*  
-dst_inst inst3 -dst_pin port3.* -name rule4 -allow_pin_fanout false
```

However, setting the argument to true allows the command to make one-to-many connections between port2 (of source instance) and the ports, port3 and port3a (of destination instance).

add_config

Defines configurations for a design

Usage

```
add_config visibility  
-default <default-condition> |  
-table <table-name>  
  (-<col-name> <value>)*  
-visible <Y | N>
```

Description

The add_config command defines configurations for a design. These configurations are used by generators.

Arguments

The add_config command has the following arguments:

visibility

Specifies that visibility configuration has to be set.

`-default <default-condition>`

Specifies the comma-separated list of default visibility condition for the objects in a design.

All objects which have visibility condition same as specified in this switch will be made visible to the generators.

To know more on visibility conditions, see [Iterating Over a List or Table](#).

`-table <table-name>`

Specifies the table in which a particular object has to be made visible or hidden.

`-<col-name> <value>`

Specifies the column name of the table specified by the `-table` option and the value of the column for which visibility needs to be specified.

You can specify multiple column name-value pairs.

`-visible <Y | N>`

Specifies whether the particular object has to be made visible (Y) or hidden (N).

Examples

Some of the examples of using the `add_config` command are as follows:

1. The following example sets all those objects visible, which have their visibility set to either `con1` or `con2`, apart from the objects which have no visibility set:

```
add_config visibility -default "con1, con2"
```

2. The following example makes the register, `R2`, of component, `comp`, visible irrespective of the default visibility specified by the configuration:

```
add_config visibility -table component\[comp\].Memory.RegisterBlock  
-name R2 -visible Y
```

3. The following example selects the bitfield subtable of the row having the key column value as `R1` in the `RegisterBlock` table:

```
add_config visibility -table  
  
component\[comp\].Memory.RegisterBlock\[R1\].bitfield -name R1_B1  
-visible Y
```

add_connection

Adds a connection.

Usage

```
add_connection
  <from-options>
  <to-options>
  [ -align <LSB | MSB> ]
  [ -debug <Y | N>
  [ -noconn ]
  [ -visibility <visibility-condition>]
  [ -description <description>]
  [ -deltadelay <delay-value> [<delay-unit>]]
  [ -bits <normal | reverse > ]
  [ -name <net-name> ]
  [ -exclude {logical-pins} ]
  [ -macro_visibility <macro-condition> ]
  [ -conn_backref <backref-string> ]
  [ -create_ports <TRUE | FALSE> ]
  [ -reuse_ports <TRUE | FALSE> ]<from-options>::=
  -instance <inst-name> -pin <pin-name>
  [ -through <hier-pin-name-list>
  [ -lsb <lsb> ] [ -msb <msb> ] ] | -instance <inst-name>
  -interface <interface-name>
  | -port <port-name> [ -lsb <lsb> ] [ -msb <msb> ] | -interface
  <interface-name>
  [ -pin <eco-port-name>]
  [ -reroute <reroute_expression> ]
  <to-options>::=
  -instance <inst-name> -pin <pin-name>
  [ -through <hier-pin-name-list>
  [ -lsb <lsb> ] [ -msb <msb> ] ]
  | -instance <inst-name>
    -interface <interface-name>)
  | -port <port-name> [ -lsb <lsb> ] [ -msb <msb> ]
  | -interface <interface-name>
    [ -lsb <lsb> ]
    [ -msb <msb> ]
  | -tieoff <tieoff-value>
  [ -pin <hierarchy-port-name>]
```

Description

The `add_connection` command adds connections between specified FROM object (instance pins, instance's interface, design ports, design logical ports, or design's interface) and the specified TO object (instance pins, instance logical pins, instance's interface, design ports, design logical ports, design's interface, or a tiedoff value).

The connection types are as follows:

Connection Types	FROM Object	TO Object
Adhoc-to-Adhoc	Instance pins	Instance pins, design ports
	Design ports	Design ports
	Design ports	Instance pins
Adhoc-to-Tieoff	Instance pins, design ports	tiedoff value
Interface	Instance's interface, design's interface	Instance's interface, design's interface

Adding Connections across the Hierarchies

GenSys uses the `::` delimiter as the design hierarchy separator. A hierarchical name must start with the `::` delimiter to indicate that this is from the top plane.

Consider the following example:

```
add_connection -instance I1 -pin P1 -instance ::dmass::DMA -pin P2
```

Here, the connection is from pin P1 of instance I1 to pin P2 of instance DMA in subsystem dmass in the top plane.

Hierarchical connections are resolved during netlisting by automatically creating the required internal ports and connections.

Arguments

The `add_connection` command has the following arguments:

`-instance <inst-name>`

Specifies the name of the instance for which the connection is being defined. You can also specify the hierarchical instance name in this argument.

Note:

The `-instance` argument is not applicable for connections to design ports and design interface ports.

`-pin <pin-name>`

Specifies the name of the instance pin of instance *<inst-name>*.

`-port <port-name>`

Specifies the name of the design port.

Note:

The `-instance` argument is not applicable for connections to design ports.

`-lsb <lsb>`

(Optional) Specifies the LSB of the instance pin, instance logical pin, design port, or the design logical port.

Use the `-lsb` argument to specify the LSB of vector objects.

See [Specifying Values for Integer Arguments](#) for more details.

Note:

You must specify both the `-msb` and `-lsb` arguments. If either of them is specified, GenSys reports an error.

`-msb <msb>`

(Optional) Specifies the MSB of the instance pin, instance logical pin, design port, or the design logical port.

Use the `-msb` argument to specify the MSB of vector objects.

See [Specifying Values for Integer Arguments](#) for more details.

Note:

You must specify both the `-msb` and `-lsb` arguments. If either of them is specified, GenSys reports an error.

Note:

For vector ports, if you do not specify both the `-lsb` argument and the `-msb` argument, the connection is applicable for the entire vector port.

`-interface <interface-name>`

Specifies the name of the instance's interface or design's interface.

`-macro_visibility <macro-condition>`

(Optional) Specifies a macro condition in which the connection should be visible in the generated Verilog RTL.

Based on the specified macro condition, a connection is placed under appropriate Verilog conditional pre processor directives, such as ``ifdef` in the generated Verilog RTL.

Refer to [Example 4](#).

`conn_backref <backref-string>`

(Optional) When the `-datasource <Y | N>` argument of the `set_gensys_options` command is set to N, no back-reference information is attached to a connection to enhance performance.

However, if you still want to attach some back-reference information, such as a generator name, to a connection, specify that information by using this argument.

`-create_ports <TRUE | FALSE>`

(Optional) Specifies if the connection being created should be a hierarchical connection in RTL. This argument is useful only for a hierarchical connection created in GenSys by the `add_connection` command. It is not useful for non-hierarchical connection.

By default, this argument is TRUE. When this argument is TRUE, the hierarchical connection specified in GenSys creates additional ports in the boundary to make connections between two ports or terminals present in a different hierarchy.

Set this argument to FALSE to print the connection as hierarchical in the RTL.

In a hierarchical connection:

- One end of the connection is in the current design.
For example, either a port is in the design or a pin of an instance is in the design.
- The other end of the connection is in some other hierarchy.
For example, an instance pin is in a sub hierarchy of the current design. It cannot be the pin of an instance in the current design.

If black boxes are detected in the path for hierarchical pin, user-specified hierarchical pin path is assumed to be correct.

This argument makes only the current design dirty and do not make any change to the components in the hierarchical pin path.

RTL (Verilog) generated for the current design in this case will contain the hierarchical connectivity with the hierarchical references to a hierarchical pin (if it exists or assumed to be existing due to black boxes in the hierarchical path at RTL generation).

See [Example 12](#).

`-reroute <reroute_expression>`

(Optional) Evaluates the given expression (*<reroute_expression>*), creates a glue instance in the active design, and makes the required connectivity with the pins of this glue instance.

For example, consider that you specify the value to this argument as "glueComp::glueInst(g1::I1.p1,g2::I2.p2)". In this case:

- glueComp is the existing component.
- GenSys instantiates the instance glueInst of this glue component.
- GenSys connects the g1 pin of glueInst with the p1 pin of the I1 instance and connects the g2 pin of glueInst with the p2 pin of the I2 instance.

Also see [Example 11](#).

`-tieoff <tieoff-value>`

Specifies the tieoff value when creating a tieoff connection.

For vector ports, the connection mapping is LSB-based mapping on the binary equivalent of the specified tieoff value.

`-align <LSB | MSB>`

(Optional) Specifies the order of scalar-level connectivity for the (vector) port.

The default value of the `-align` argument is MSB and the port is connected MSB onwards.

`-noconn`

(Optional) Specifies that the terminal corresponding to the instance or interface should be forcefully kept open.

If `-noconn` is specified for a single terminal of instance, then the `-noconn` argument is given highest priority. However, for an interface instance, if some adhoc connection is specified on any individual pin of the interface, then the adhoc connection is honored. However, if another interface connection is specified on that interface, it is ignored and `-noconn` (open specification) is given highest priority.

`-visibility <visibility-condition>`

(Optional) Specifies the comma-separated list of conditions under which the object would be visible to the generators.

For the RTL flow, use the `-macro_visibility` argument.

`-debug <Y | N>`

Specifies if the extra debug information needs to be dumped.

`-description <description>`

(Optional) Specifies description for the new connections. This description for connections is used during netlisting.

This argument also accepts unicode characters.

`-deltadelay <delay-value> [<delay-unit>]`

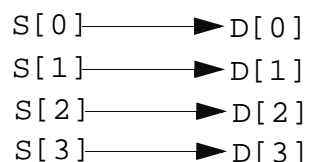
(Optional) Specifies delay value for the new connections. This delay value for connections is used in netlisting. You can also specify the unit for the delay value as hr, min, sec, ms, us, ns, ps, or fs. By default, the delay unit is considered as ns (nano seconds).

It is optional to provide a space between the delay value and delta unit value. Therefore, the value 2ns is same as 2 ns.

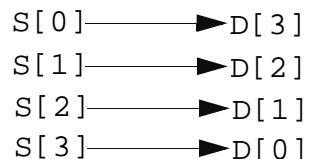
`-bits <normal | reverse>`

(Optional) Specifies the way in which source and destination LSB/MSB bits should be connected.

If the value of this argument is set to normal (default behavior), the first bit of the source is connected with the first bit of the destination. Similarly, the second bit of the source is connected with the second bit of the destination, and so on. For example, consider the source port, S[0:3], which needs to be connected with the destination port, D[0:3]. If the value of this argument is set to normal, the source and destination ports are connected in the following manner:



If you set the value of this argument to reverse, the source and destination ports will be connected in the following manner:



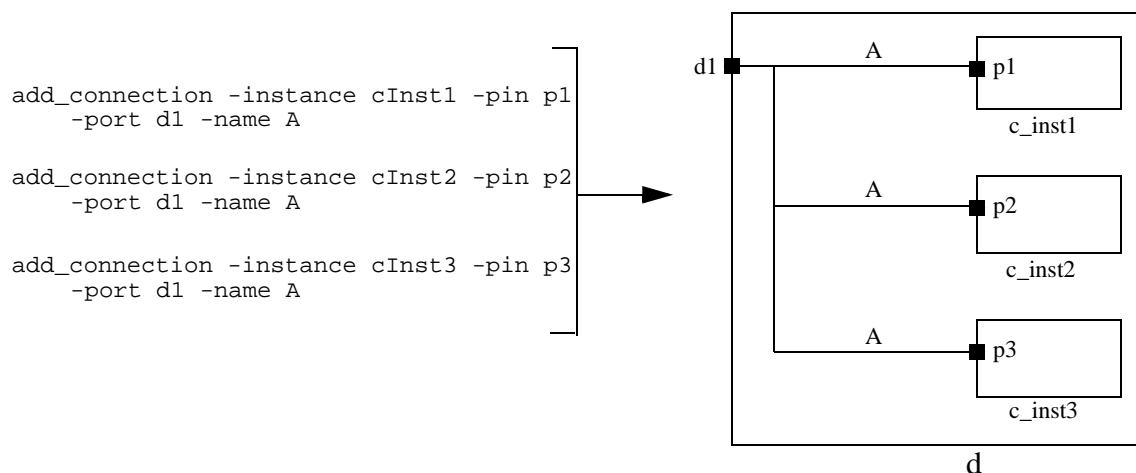
In the above case, the first bit of the source is connected with the last bit of the destination, and so on.

`-name <net-name>`

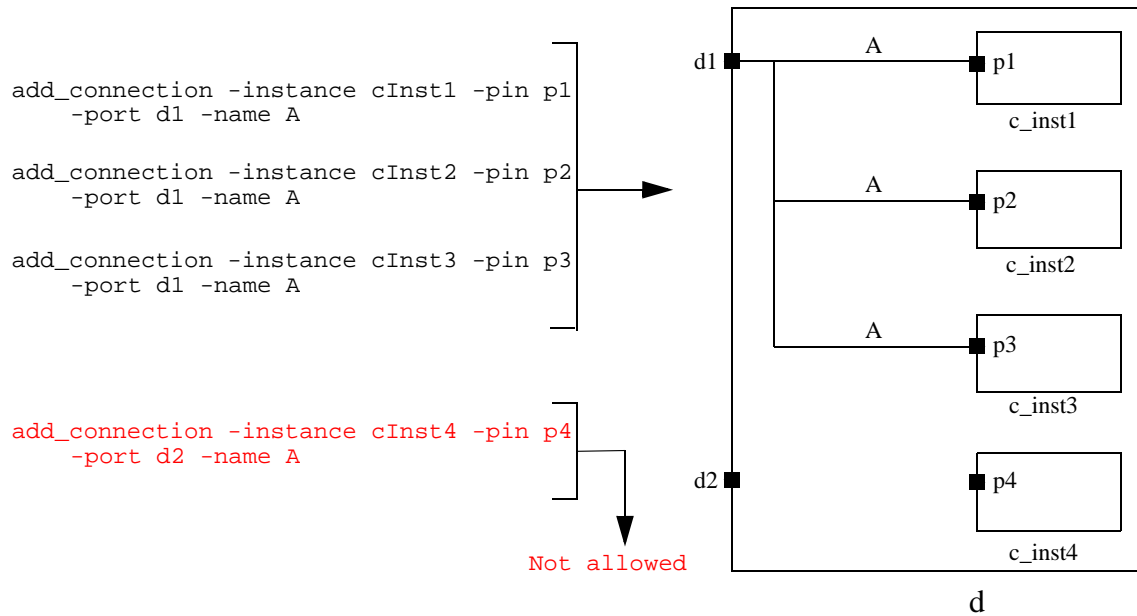
(Optional) Specifies the name of a net connecting two ports/pins. This net name is saved in the RTL generated by the `GenerateVerilog` or `GenerateVHDL` command.

The advantage of using this argument is that the generated RTL becomes more readable as you can easily identify nets connecting different ports/pins.

You can specify the same net name in the `-name` argument of multiple `add_connection` commands only if the driver of all these connections is same. For example, in the following scenario, you can assign the same name to the nets connecting the `d1` port to the `p1`, `p2`, and `p3` pins:



However, you cannot assign the name `A` to the net connecting the `d2` port and the `p4` pin, as shown in the following figure:

**Note:**

Ensure that the driver side is completely connected while using this argument. That is, all bits of the driver side are completely connected to each destination.

For more details, see [Example 10](#).

Also see [Priority of Net Names](#).

`-exclude {logical-pins}`

(Optional) Specifies interface logical pins that should be excluded while making interface connections. You can later specify connectivity for such excluded pins.

For example, consider the following command:

```

add_connection -instance comp10 -interface intf0
               -instance comp20 -interface intf0 -exclude {L0}
  
```

In the above example:

- An interface connection between the `intf0` interface of the `comp10` instance and the `intf0` interface of the `comp20` instance is being made.
- The `L0` logical pin of interfaces is excluded while making a connection. You can later specify connectivity for this pin, as shown in the following example:

```

add_connection -instance comp10 -pin P0 -tieoff 0
  
```

Port P0 is mapped to logical port L0.

Please note the following points for this argument:

- You can specify a comma-separated list of logical pins of an interface of an instance or a design. The list can contain name of logical pins or splice of logical pins.

The following command excludes the L0 and L1 logical pins from the interface connection being created:

```
add_connection -instance comp10 -interface intf0 -instance comp20
-interface intf0 -exclude {L0,L1}
```

You can also specify bit slices of logical pins to be excluded from a connection. For example, consider that each interface of the comp10 and comp20 components have following logical pins:

L0[0:4], L1[0:7], and L2[2:4]

Now consider that you specify the following command in which specific bit slices of logical pins are being excluded from the connection:

```
add_connection -instance comp10 -interface intf0
-exclude {L0} -instance comp20 -interface intf0
-exclude {L0[0],L1[2:5],L0[1],L1[1],L2,L1[7]}
```

In the above case, connections will be created for the following bits of logical pins of interfaces:

L0[2:4], L1[0], and L1[6]

- This argument is common to both the interfaces being connected. That is, you do not need to specify this argument for each interface separately.
- You should use this argument only while connecting two interfaces in a design. These interfaces could either be on a design or a component instance. If you use this argument while making a connections other than an interface connection, GenSys reports an error message.
- In MRS, excluded pins are reflected within the `<excludeConn>` tag for both the interfaces, as shown in the following example:

```
<ExcludeConn>L0</ExcludeConn>
```

- In IP-XACT, connections are reflected as adhoc connection as in usual interface connections.
- In GUI, excluded logical pins are stored in the exclude column of the Connections.Interface table.

```
-reuse_ports <TRUE | FALSE >
```

(Optional) Reuses existing ports while elaborating a specified hierarchical connection.

Default value is FALSE and this feature is disabled.

If you set the value to TRUE and specify a hierarchical connection, GenSys reuses existing ports while elaborating the hierarchical connection. See [Example 13](#).

`-through <hier-pin-name-list>`

(Optional) Specifies a space separated list of hierarchical pin names that should be used when creating a hierarchical connection.

The connection created passes through the specified via points. If there is no pin with the specified name in the hierarchical instance, a pin with that name is created and the connection created traverses through the created pin. See [Example 14](#).

Examples

Some of the examples of using the `add_connection` command are as follows:

Example 1

Adhoc-to-Adhoc connection between two instance pins:

```
add_connection
  -instance inst1 -pin pin1 -msb 7 -lsb 0
  -instance inst2 -pin pin2 -msb 15 -lsb 8
```

The above specification creates an Adhoc connection between pin `pin1[7:0]` of instance `inst1` and pin `pin2[15:8]` of instance `inst2`.

Example 2

Adhoc-to-Adhoc connection between an instance pin and a design port:

```
add_connection
  -instance COMPONENT2 -pin P1 -lsb 0 -msb 31
  -port P1 -lsb 0 -msb 31
```

The above specification creates an Adhoc connection between pin `P1[0:31]` of instance `COMPONENT2` and design port `p1[0:31]`.

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case2` directory.

Note:

For an interface instance, if some adhoc connection is specified on any individual pin of interface, the adhoc connection specification is honored.

Example 3

Adhoc-to-Tieoff connection for an instance pin:

```
add_connection
  -instance inst1 -pin pin1 -tieoff 0x10
```

The above specification creates a tieoff connection for pin pin1 of instance inst1 with tieoff value 0x10.

Example 4

Interface connection between two instance interfaces:

```
add_connection
  -instance inst1 -interface i1
  -instance inst2 -interface i2
```

The above specification creates an interface connection between interface i1 of instance inst1 and interface i2 of instance inst2.

Example 5

Interface connection between an instance interface and a design interface:

```
add_connection
  -instance inst1 -interface i1
  -interface di2
```

The above specification creates an interface connection between interface i1 of instance inst1 and the design interface di2.

Example 6

Deferred connection from an instance pin to some logical point in space:

```
add_connection
  -instance U1 -pin modclk
```

Example 7

Open connection on a pin of an instance:

```
add_connection -instance inst1 -pin mocclk -noconn
```

Here, an open pin modclk is created for instance inst1.

Example 8

Open connection on an interface of an instance:

```
add_connection -instance inst1 -pin mocclk -noconn
```

Here, an open pin modclk is created for instance inst1.

```
add_connection -instance inst1  
               -interface intf1 -noconn
```

Here, an open interface intf1 is created for instance inst1. Later, if you give the following command to create another connection using this interface intf1:

```
add_connection -instance inst1  
               -interface intf1 -interface des_intf
```

In this case, the command to create a new connection is ignored and the open connection is given priority.

Example 9

Adhoc connection between two design ports

```
new design d  
add_port -name p1 -lsb 0 -msb 3 -direction IN  
add_port -name p2 -lsb 0 -msb 3 -direction OUT  
add_connection -port p1 -lsb 0 -msb 3 -port p2 -lsb 0  
               -msb 3
```

Here, the design ports, p1 and p2 will be connected.

Example 10

Using the -name <net-name> argument to assign a name to the net connecting ports/pins:

```
new component c  
add_port -name c1 -direction IN -lsb 0 -msb 1
```



```

new design d
add_port -name d1 -direction IN -lsb 0 -msb 1
add_instance -name cInst -master c

add_connection -instance cInst -pin c1 -port d1
               -name firstConn
run GenerateVerilog

```

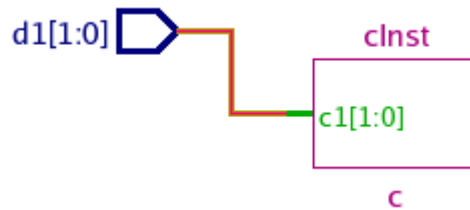
After running the above commands, the following RTL is generated:

```

module d (d1);
input    [1:0]    d1;
wire     [1:0]    firstConn;
    c cInst (
        .c1(firstConn) //I:2
    );
    assign firstConn = d1;
endmodule

```

Following is the schematic of the above RTL:



In the above RTL, the name of the net connecting the `d1` port and the `c1` pin is `firstConn`.

Example 11

Consider the following example:

```

new component c1
add_port -name i1 -direction IN
new component c2
add_port -name o1 -direction OUT
new component glueComp
add_port -name g1 -direction IN
add_port -name g2 -direction OUT
new design d
add_instance -name inst1 -master c1
add_instance -name inst2 -master c2
add_connection -instance inst1 -pin i1 -instance inst2
               -pin o1
add_connection -reroute
               "glueComp::glueInst1(g1::inst2.o1,g2::inst1.i1)"

```

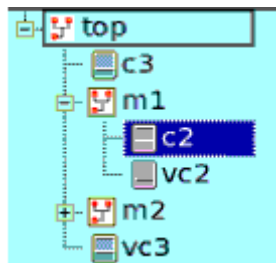
```
# The above command deletes the connectivity between
# inst1.i1 and inst2.o1 and connects inst1.i1 with g2
# and inst2.o1 with g1.
run GenerateVerilog
```

In this example, the generated Verilog is as follows:

```
module d ();
wire      glueInst1_g2;
wire      inst2_o1;
    c1    inst1 (
        .i1(glueInst1_g2)
    );
    c2    inst2 (
        .o1(inst2_o1)
    );
    glueComp    glueInst1 (
        .g1(inst2_o1),
        .g2(glueInst1_g2)
    );
endmodule
```

Example 12

Consider the design hierarchy shown in the following figure:



In the above scenario, consider that you specify the following command when top is the current design:

```
add_connection -instance vc3 -pin out1 -msb 5 -lsb 4
               -instance m1::vc2 -pin in1 -msb 5 -lsb 4 -create_ports FALSE
```

In this case, the RTL generated the top module will have instantiation of vc3 with hierarchical connectivity, as shown below:

```
vcomp    vc3 ( //Instance
            .in1(), //In[5:3]
            .out1({m1.vc2.in1[5:4], //Out[5:4] bits
```

```

        UNINTENTIONAL_OPEN_vc3_out1_0} //Out[3] bit
        ), //Out[5:3]
        .inp(), //In[8:4]
        .outp() //Out[8:4]
    );

```

Now consider you specify the following command:

```

add_connection -port ab -msb 4 -lsb 3 -instance m1::vc2
               -pin in1 -msb 4 -lsb 3 -create_ports FALSE

```

In this case, the RTL generated for top will have the following if both ends of the connection exists or is assumed to exist due to black boxes in the hierarchial path at the time of RTL generation:

```

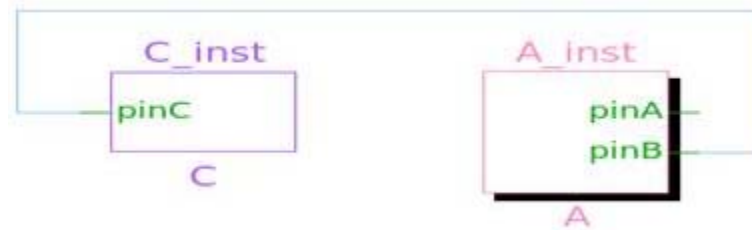
assign m1.vc2.in1[4:3] = ab[4:3];

```

Example 13

This example illustrates the use of the `-reuse_ports` argument.

By default, the `-reuse_ports` option is set to `FALSE` and an extra port is created.



The following Tcl command sets the `-reuse_ports` option to `TRUE`:

```

add_connection
  -instance A_inst::B_inst -pin pinB
  -instance C_inst
  -pin pinC
  -reuse_ports TRUE

```

The resulting connection is shown in the following figure. Notice that an extra port is not created.

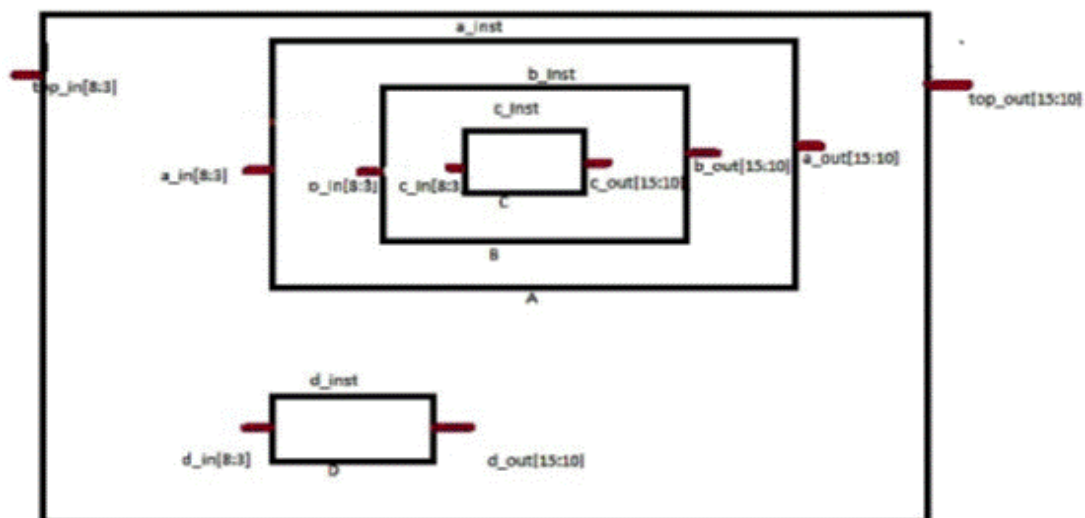


Example 14

This example shows the impact of the -through argument.

Input

Consider the following design:



The RTL file for this design is as follows:

```
module A(a_in,a_out);
input  [8:3] a_in;
output [15:10] a_out;
B b_inst();
endmodule
```

```
module B(b_in,b_out);
input  [8:3] b_in;
output [15:10] b_out;
C c_inst();
```

```
endmodule

module C(c_in,c_out);
input [8:3] c_in;
output [15:10] c_out;
endmodule

module D(d_in,d_out);
input [8:3] d_in;
output [15:10] d_out;
endmodule

module top(top_in,top_out);
input [8:3] top_in;
output [15:10] top_out;
A a_inst();
D d_inst();
Endmodule
```

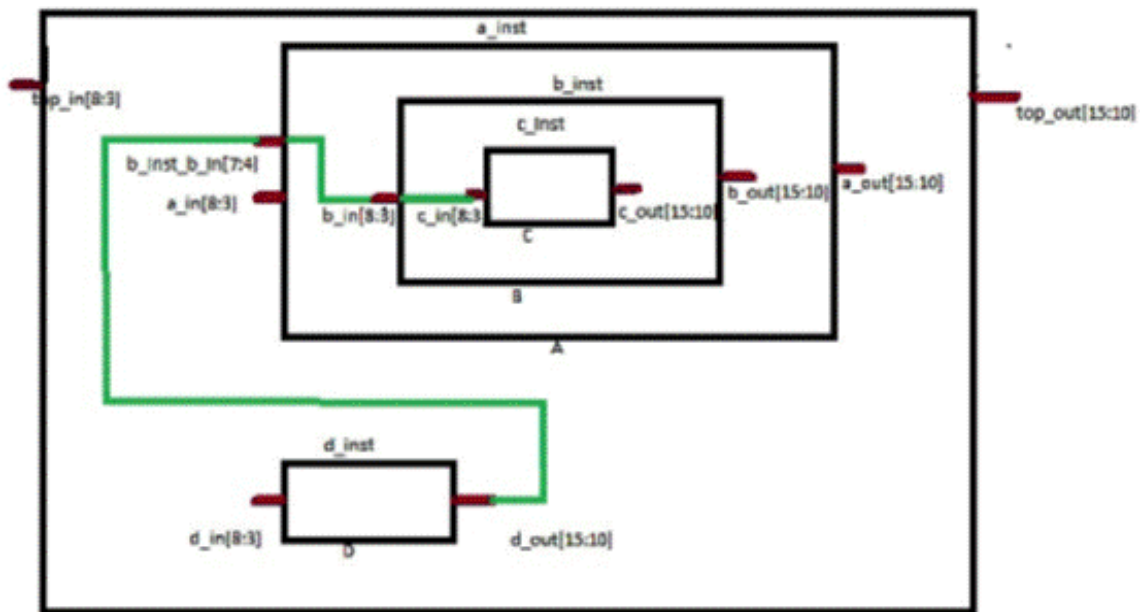
Tcl Command with -through Argument

The following Tcl command is executed:

```
add_connection -instance a_inst::b_inst::c_inst -pin c_in -msb 8 -lsb 5
               -instance d_inst -pin d_out -msb 13 -lsb 10 -through
               {a_inst::b_inst::b_in[7:4]}
```

Output

After running this Tcl command, the following connections are created:



The output RTL for module B is as follows:

```

module B (b_in ,
          b_out);

input    [8:3]    b_in;
output   [15:10]  b_out;
wire     [1:0]    UNINTENTIONAL_OPEN_c_inst_c_in_0;

    C    c_inst ( //Instance
        .c_in({b_in[7:4], //In[8:5] bits
               UNINTENTIONAL_OPEN_c_inst_c_in_0} //In[4:3] bits
        ), //In[8:3]
        .c_out() //Out[15:10]
    );

Endmodule

```

add_datasource

Adds datasource information to the currently active object

Usage

```
add_datasource {  
  [ -personname <datasource-person-name> ]  
  [ -changetype <type> ]  
  [ -changereason <reason> ]  
  [ -changemethod <method> ]  
  [ -changedate <date> ]  
  [ -datafrozen <TRUE | FALSE> ]  
  [ -creationdate <date> ]  
}
```

Description

The add_datasource command adds the datasource information to the currently active object.

Arguments

The add_datasource command has the following arguments:

`-personname <datasource-person-name>`

(Optional) Specifies the datasource name.

`-changetype <type>`

(Optional) Specifies the type of the datasouce

`-changereason <reason>`

(Optional) Specifies the reason for making changes in the datasource

`-changemethod <method>`

(Optional) Specifies the method that make the change

`-changedate <date>`

(Optional) Specifies the date on which the change occurs.

`-datafrozen <TRUE | FALSE>`

(Optional) Specifies whether the changes made should be frozen.

`-creationdate <date>`

(Optional) Specifies the date on which the datasource gets created. If you do not specify the creation date then the current system date is considered.

add_file

Adds the specified file to the specified file set

Usage

```
add_file
  -file_id <file-ID>
  -name <name>
  [ -dependency <dependency1> <dependency2> ... ]
  -file_type <file-type>
  [ -is_include_file <TRUE | FALSE> ]
  [ -is_package_file <TRUE | FALSE> ]
  [ -logical_name <logical-name> ]
  [{-define_attribute <name>
    -value <value>}* ]
  -file_set <FileSet-name>
  [ -description <description> ]
  [ -macro_visibility <macro-condition> ]
  [ -dissolve_include <TRUE | FALSE> ]
  [ -include_position <BEFORE | AFTER | DEFAULT> ]
```

Description

The `add_file` command adds the specified file to the specified file set.

Arguments

The `add_file` command has the following arguments:

`-file_id <file-ID>`

Specifies the unique ID of the file.

`-name <name>`

Specifies the name of the file.

`-dependency <dependency1> <dependency2> ...`

Specifies absolute or relative directory paths that store include files on which this file is dependant.

`-file_type <file-type>`

Specifies the type of the file.

`-is_include_file <TRUE | FALSE>`

Specifies whether the file is an include file.

`-is_package_file <TRUE | FALSE>`

Specifies whether the file is a package file.

`-logical_name <logical-name>`

Specifies the logical name of the file.

`-define_attribute <name>`

Specifies the name of the DEFINE attribute.

`-value <value>`

Specifies the value of the DEFINE attribute.

`-file_set <FileSet-name>`

Specifies the name of the file set.

`-description <description>`

Specifies the description of the file.

`-macro_visibility <macro-condition>`

(Optional) Specifies a macro condition in which an include file should be visible in the generated Verilog RTL. This is applicable only on include files specified through the ``include` directive.

Based on the specified macro condition, the include file is placed under appropriate Verilog conditional pre processor directives, such as ``ifdef` in the generated RTL.

Refer to [Example 4](#).

`-dissolve_include <TRUE | FALSE>`

(Optional) Set this argument to TRUE to dissolve the content of the include file to the RTL being generated.

By default, this argument is set to FALSE. In this case, the content of the include file is not dissolved to the RTL being generated. Instead, the generated RTL contains the include statement that includes a file but not its content.

`-include_position <BEFORE | AFTER | DEFAULT>`

(Optional) This argument accepts any of the following values:

- **BEFORE**
Specify this value to place the include statement for a file or its content before the module keyword of the relevant module.
- **AFTER**
Specify this value to place the include statement for a file or its content after the endmodule keyword of the relevant module.
- **DEFAULT**
This is the default value that causes RTL generation in the default manner.

add_file_set

Adds the specified file set to the most recently active design.

Usage

```
add_file_set
  -name <FileSet-name>
  [ -group <group-name> ]
  [ -dependency <dependency1> <dependency2> ... ]
  [ -description <description> ]
```

Description

The `add_file_set` command adds the specified file set to the most recently active design.

A file set refers to a group of files such as VHDL/Verilog/System Verilog/System C/C source files, etc. These file sets provide multiple views for a design/component.

The file set information is used and specified by various generators. For example, during RTL import, GenSys captures the information of the RTL files such as language and file path, and specify this information to the file sets and the related files for the design/component. Similarly, during RTL export, GenSys specifies this information in case if it missing from the design/component.

Arguments

The `add_file_set` command has the following arguments:

`-name <FileSet-name>`

Specifies the name of the file set.

`-group <group-name>`

Specifies the name of the group to which the file set belongs.

`-dependency <dependency1> <dependency2> ...`

Specifies absolute or relative directory paths that store include files on which the rest of the files are dependant.

`-description <description>`

Specifies the description of the file set.

add_hier_connection

Creates new hierarchical connections between the source and the sink

Usage

```
add_hier_connection    -source <source-hier-path>
                        -dest <dest-hier-path>
                        -through <hier-inst-names>
```

Description

The `add_hier_connection` command creates new hierarchical connections between the source and the sink while optionally forcing the connectivity through a set of via instances.

Arguments

The `add_hier_connection` command has the following arguments:

`-source <source-hier-path>`

Specifies the hierarchical path for the source with reference to the top design.

`-dest <sink-hier-path>`

Specifies the hierarchical path for the sink with reference to the top design.

`-through <hier-list-names>`

Specifies a space-separated list of hierarchical instance names with reference to the top design.

Examples

See [Examples of Rerouting GenSys Connections through Via Points](#).

add_info

Adds miscellaneous information for the current object (design/interface/component)

Usage

```
add_info
[ -short_name <short-name> ]
[ -description <description> ]
[ -componentType <comp-type> ]
[ -rtl_library <library-name> ]
[ -rtl_package <package-name> ]
[ -systemGroupNames <group-names> ]
[ -busDefName <bus-def-name> ]
[ -busDefVersion <bus-def-version> ]
[ -busDefLibrary <bus-def-library> ]
[ -busDefVendor <bus-def-vendor> ]
[ -extendsBusDefName <extends-bus-def-name> ]
[ -extendsBusDefVersion <extends-bus-def-version> ]
[ -extendsBusDefLibrary <extends-bus-def-library> ]
[ -extendsBusDefVendor <extends-bus-def-vendor> ]
[ -desViewName <design-view-name> ]
[ -desViewVersion <design-view-version> ]
[ -desViewVendor <design-view-vendor> ]
[ -desViewLib <design-view-library> ]
```

Description

The `add_info` command adds miscellaneous information for a current object (design, interface, or component).

Arguments

The `add_info` command has the following arguments:

`-short_name <short-name>`

(Optional) Specifies the short name of the current object.

`-description <description>`

(Optional) Specifies the description of the current object.

This argument also accepts unicode characters.

`-componentType <comp-type>`

The `-componentType` argument is applicable for components only.

(Optional) Specifies the component type generally as one of the following:

BUSLOGIC	CBSR_CELL	CPU
GLUE	MBSR_CELL	MEMORY
MUX_CELL	PAD_CELL	PERIPHERAL
SBSR_CELL	SUBSYSTEM	OTHERS

If you do not specify the `-componentType` argument, the component type is marked as " " (empty) for components and `SUBSYSTEM` for designs.

You can also specify a user-defined value to the `-componentType` argument, if your requirement does not match with any of the above listed component types.

`-rtl_library <library-name>`

(Optional) Specifies the library information of the component. This argument value is used when the `-component_decl` argument of the [set_rtl_option](#) Tcl command is set to `FALSE`. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to `FALSE`, the component declaration is not dumped in VHDL, but use library and package statements are dumped. In such case, the component declaration is picked from the library specified in this argument.

`-rtl_package <package-name>`

(Optional) Specifies the package information of the component. This argument value is used when the `-component_decl` argument of the [set_rtl_option](#) Tcl command is set to `FALSE`. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to `FALSE`, the component declaration is not dumped in VHDL, but use library and

package statements are dumped. In such case, the component declaration is picked from the package specified in this argument.

`-systemGroupNames <group-names>`

(Optional) Specifies a comma-separated list of system group names allowed in a bus definition.

`-busDefName <bus-def-name>`

(Optional) Specifies the name of a bus definition.

`-busDefVersion <bus-def-version>`

(Optional) Specifies the version of the bus definition.

`-busDefLibrary <bus-def-library>`

(Optional) Specifies the library name of the bus definition.

`-busDefVendor <bus-def-vendor>`

(Optional) Specifies the vendor name of the bus definition.

`-extendsBusDefName <extends-bus-def-name>`

(Optional) Specifies the name of a bus definition extension.

`-extendsBusDefLibrary <extends-bus-def-library>`

(Optional) Specifies the library of the bus definition extension.

`-extendsBusDefVersion <extends-bus-def-version>`

(Optional) Specifies the version number of the bus definition extension.

`-extendsBusDefVendor <extends-bus-def-vendor>`

(Optional) Specifies the vendor of the bus definition extension.

`-desViewName <design-view-name>`

(Optional) Specifies the name of a design view.

`-desViewVersion <design-view-version>`

(Optional) Specifies the version number of the design view.

`-desViewVendor <design-view-vendor>`

(Optional) Specifies the vendor of the design view.

`-desViewLib <design-view-library>`

(Optional) Specifies the library of the design view.

Example

The following example adds the specified miscellaneous information to the current component:

```
add_info
  -componentType CPU
  -short_name C1
  -thirdPartyGenerator "mygenfunc"
  -generatedFileName "master"
  -description "my first component"
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/examples/Genesis/case1 directory.

add_instance

Adds a component instance or a subsystem to the active design

Usage

```
add_instance
  -name <inst-name>
  -master <comp-name> | <des-name> | <predefined-glue-master>
  [ -vendor <vendor-name> ]
  [ -version <version-num> ]
  [ -library <library-name> ]
  [ -power_domain <pd-name> ]
  [ -voltage <voltage-value> ]
  [ -visibility <visibility-condition> ]
  [ -configurability
    <space-separated-list-of-master-  IP-macros-being-defined> ]
  [ -conditionality <valid-Tcl-expression> ]
  [ -pragma <pragma-name> ]
  [ -input <num-of-input-pins> ]
  [ -width <width-of-input-pin> ]
  [ -macro_visibility <macro-condition> ]
  [ -macro <macro-name> ]
  [ -macro_file <file-name> ]
```

Description

The `add_instance` command adds an instance of the specified component or subsystem to the active design.

Arguments

The `add_instance` command has the following arguments:

`-name <inst-name>`

Specifies the name of the component or subsystem instance being created.

`-master <comp-name | des-name> | <predefined-glue-master>`

Specifies the name of the master component or design (in case of subsystems) of the new instance made.

Alternatively, you can specify any of the following predefined glue master to this argument:

GENSYS_AND	GENSYS_OR	GENSYS_NOR
GENSYS_NAND	GENSYS_XOR	GENSYS_XNOR
COM_RTL_NOT	COM_RTL_BUF	COM_RTL_FDP C
COM_RTL_MUX	COM_RTL_ONEHOT_MU X	COM_RTL_NEQ
COM_RTL_LD		

`-vendor <vendor-name>`

(Optional) Specifies the vendor name for the master component or subsystem. If you do not specify the `-vendor` argument, GenSys selects the specified master of the first found vendor.

`-version <version-num>`

(Optional) Specifies the version number of the master component or subsystem. If you do not specify the `-version` argument, GenSys selects the specified master of the first found version.

`-library <library-name>`

(Optional) Specifies the name of library of the master component or subsystem. If you do not specify the `-library` argument, GenSys selects the specified master of the first found library.

`-power_domain <pd-name>`

(Optional) Specifies the power domain of the component or subsystem instance.

`-voltage <voltage-value>`

(Optional) Specifies the voltage value for the power domain of the component or subsystem instance.

`-visibility <visibility-condition>`

(Optional) Specifies the comma-separated list of conditions under which the object would be visible to the generators.

For RTL flow, use the `-macro_visibility` argument.

`-configurability <space-separated-list-of-master-IP-macros-being-defined>`

(Optional) Specifies the macros to be defined for the master IP during instantiation. All instances of the master IP need to have the same set of macros defined otherwise an ERROR message is thrown and the originally defined set of macros are retained for the IP. Refer to [Example 3](#).

This switch value is visible in the configurability column under the info tab of the GenSys GUI for instance.

`-conditionality <valid-Tcl-expression>`

(Optional) Enables conditional instantiation in RTL (Verilog/VHDL).

If any instance has a valid conditional Tcl expression that can be converted to an equivalent Verilog/VHDL expression by the tool, that instance is dumped inside the `if generate` block in Verilog/VHDL.

Note:

Conditionality expression should either be a constant expression, or it should be dependent on top-level design parameters.

`-pragma <pragma-name>`

When you apply pragmas on an instance, then during RTL export, GenSys dumps that instance under the pragma specified by this argument. For example, consider the following command:

```
add_instance -name inst -master comp -pragma SYNTHESIS
```

Based on the above command, the following RTL is generated:

```
//synopsys synthesis_offcomp inst();//synopsys synthesis_on
```

```
-input <num-of-input-pins>
```

(Optional) Specifies the number of input pins for the glue instance to be added in the design. For example, you can add an 8-input AND gate.

For an N-input glue instance, GenSys names the input pins in the in[0 - N-1] format. The output pin is named as out0. For example, consider the following example in which you add a 4-input AND gate:

```
add_instance -name I1 -master "GENERIC_AND" -input 3 -width 4
```

In the above case, the 4-input AND gate has input pins as in0, in1, in2, and in3. The output pin is named as out0.

See [Example 4](#) for adding an N-input glue instance in a design.

Note:

When you use this argument, specify any of the following values to the -master argument:

GENSYS_A ND	GENSYS_OR	GENSYS_NO R	GENSYS_NAN D
GENSYS_X OR	GENSYS_XN OR		

```
-width <width-of-input-pin>
```

(Optional) Specifies the width of each input pin for the glue instance to be added in the design.

See [Example 4](#) for adding an N-input glue instance in a design.

Note:

When you use this argument, specify any of the following values to the -master argument:

GENSYS_A ND	GENSYS_OR	GENSYS_NO R	GENSYS_NA ND
GENSYS_X OR	GENSYS_XN OR		

```
-macro_visibility <macro-condition>
```

(Optional) Specifies a macro condition in which an instance should be visible in the generated Verilog RTL.

Based on the specified macro condition, an instance is placed under appropriate Verilog conditional pre processor directives, such as ``ifdef` in the generated Verilog RTL.

Refer to [Example 4](#).

```
-macro <macro-name>
```

(Optional) Specifies the macro name that should be compiled instead of the design name of an instance during RTL generation.

See [Example 5](#).

```
-macro_file <file-name>
```

Specifies the macro file containing the definition of the macro specified by the `-macro <macro-name>` argument.

See [Example 5](#).

Examples

Example 1

The following example adds instance COMPONENT2 of master component comp2 with power domain and voltage set as 5 and 2, respectively:

```
add_instance -name COMPONENT2 -master comp2 -power_domain 5 -voltage 2
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case2` directory.

Example 2

You can add multiple instances of the specified component or subsystem to the active design. This can be done by using the `foreach` command, as shown in the following example:

```
foreach i {1 2 3}{  
  add_instance -name comp_${i} -master comp1  
}
```

In the above example, the component comp1 is instantiated three times in the design.

Example 3

In this example, the input testcase is:

```
new component comp
add_port -name p1 -lsb 0 -msb 2 -macro_visibility {X}
add_port -name p2 -lsb 0 -msb 2 -macro_visibility {!X}
add_port -name p4 -lsb 0 -msb 2
add_port -name p5 -lsb 0 -msb 2 -macro_visibility {!Y}
add_port -name p6 -lsb 0 -msb 2 -macro_visibility {Y}
new design des
add_instance -name comp0 -master comp -macro_visibility {X}
generated Verilog RTL from GenSys will be
des.v
module des ();
.....`ifdef X
comp    comp0 (
        .p5(UNINTENTIONAL_OPEN_comp0_p5_0),    //IO:3
        .p1(UNINTENTIONAL_OPEN_comp0_p1_0),    //IO:3
        .p4(UNINTENTIONAL_OPEN_comp0_p4_0)     //IO:3
        );`endif
endmodule
```

The component comp.v is:

```
module comp (`ifndef Y
            p5,
            `else
            p6,
            `endif

            `ifdef X
            p1,
            `else
            p2,
            `endif

            p4);
`ifndef Y
inout    [2:0]    p5;
`else
inout    [2:0]    p6;
`endif

`ifdef X
inout    [2:0]    p1;
`else
```

```

inout    [2:0]    p2;
`endif

inout    [2:0]    p4;
endmodule

```

Example 4

Consider the following example of adding an N-input glue instance in a design:

```

new design top
add_port -name P1 -direction IN -lsb 0 -msb 4
add_port -name P2 -direction IN -lsb 0 -msb 4
add_port -name P3 -direction IN -lsb 0 -msb 4
add_port -name P4 -direction IN -lsb 0 -msb 4
add_port -name P5 -direction OUT -lsb 0 -msb 4
add_instance -name I1 -master GENSYS_AND -input 4
-width 5add_connection -instance I1 -pin in0 -port P1
add_connection -instance I1 -pin in1 -port P2
add_connection -instance I1 -pin in2 -port P3
add_connection -instance I1 -pin in3 -port P4
add_connection -instance I1 -pin out0 -port P5

```

In this case, the generated RTL would look like as follows:

In Verilog:

```
assign P5 = P1 & P2 & P3 & P4;
```

In VHDL:

```
P5 <= P1 and P2 and P3 and P4;
```

Example 5

Consider the following command:

```
add_instance -name i0 -master uart -macro MYUART -macro_file inc.v
# Where inc.v contains `define MYUART uart
```

In the above example, MYUART is considered as the master name instead of uart during RTL generation. GenSys retrieves the definition of MYUART from the *inc.v* macro file.

Now consider the following command in which you do not specify the macro file:

```
add_instance -name i0 -master uart -macro MYUART
```

In the above case, GenSys reports an error message indicating the missing macro file and compiles the following information before the module definition:

```
`define MYUART uart
```

add_interface

Adds an interface instance to the active design or component

Usage

```
add_interface
  -name <interface-inst-name>
  -interface_ref <interface-lib-name>
    [ -vendor <vendor-name> ]
    [ -version <version-num> ]
    [ -library <library-name> ]
  [ -address_bus_width <width> ]
  [ -address_range <range> ]
  [ -access_size <size> ]
  [ -logical <Y | N>
    | -prefix <label> -postfix <label>
    | -map {<logical-name>:<actual-name>} ]
  [ -exclude {<logical-name>} ]
  [ -type <MASTER | SLAVE | MONITOR | SYSTEM> ]
  [ -groupName <system-group-name> ]
  [ -mirror <YES | NO | MONITOR> ]
  [ -short_name <short-name> ]
  [ -description <description> ]
  [ -control_clock <clk-name> ]
  [ -control_reset <rst-name> ]
  [ -power_domain <pd-name> ]
  [ -voltage <voltage-value> ]
  [ -open <true | false> ]
  [ -visibility <visibility-condition> ]
  [   -attribute_names {space-separated-list}
    -attribute_values {space-separated-values}
  ] [ -connectionRequired <true | false> ]
  [ -endianness <big | little> ]
  [ -bitSteering <on | off> ]
```

Description

The add_interface command adds the specified interface to the active design or component.

Arguments

The `add_interface` command has the following arguments:

`-name <interface-inst-name>`

Specifies the name of the interface instance to be added to the currently active design/component.

`-interface_ref <interface-lib-name>`

Specifies the name of the master interface library of the new interface instance.

Note:

An interface library is the library of the top-level interface.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name for the master Interface library.

`-version <version-num>`

(Optional) Specifies the version number of the master Interface library.

`-library <library-name>`

(Optional) Specifies the library name for the master Interface library.

`-address_bus_width <width>`

(Optional) Specifies the address bus width of the interface.

`-address_range <range>`

(Optional) Specifies the address range of the interface.

`-access_size <size>`

(Optional) Specifies the access size of the interface.

`-logical <Y | N>`

(Optional) Specifies whether actual ports should be created for all the logical ports of an interface.

By default, the value of the `-logical` argument is set to N.

You can set the value of the `argument` to Y to create the actual ports for all the logical ports with the same name as the logical port name.

However, the names of the existing actual ports, specified in the interface library, will not get changed.

`-prefix <label>`

(Optional) Specifies the prefix to be used for creating the actual names of the ports to be added to the parent design/component for the interface ports. For example, the following specification add ports named MYINT_P1 and MYINT_P2 to the parent design/component for interface ports, P1 and P2:

`-prefix MYINT_`

Note:

If the `-logical` argument is set to Y then the `-prefix` argument is ignored.

`-postfix <label>`

(Optional) Specifies the postfix to be used for creating the names of the ports to be added to the parent design/component for the interface ports. For example, the following specification adds ports named P1_MYINT and P2_MYINT to the parent design/component for interface ports P1 and P2:

`-postfix _MYINT`

If you specify both `-prefix` and `-postfix` argument, both the labels are used. The following example add ports named A_P1_B and A_P2_B to the parent design/component for interface ports, P1 and P2:

`-prefix A_ -postfix _B`

Note:

If the `-logical` argument is set to Y then the `-prefix` argument is ignored.

`-map {<logical-name>:<actual-name>}`

(Optional) Specifies a mapping between the interface logical ports and the actual ports of the parent design/component.

The port mapping is specified as a comma-separated list of interface port name-to-expected design/component port name. You can specify the comma-separated list enclosed within double quotes (") or curly braces ({}).

Following is an example of using the `-map` argument:

`-map "P1:PORT21,P2:PORT22"`

Here, interface port, P1, is added to the parent design/component as port PORT21 and interface port, P2, is added to the parent design/component as port, PORT22.

You can also specify MSB and LSB of the ports being mapped. Following is an example:

`-map "intf_P1:des_P1{4:0}, intf_P2:des_P2{2:0}"`

Note:

The `-map` argument overrides the `-prefix`, `-postfix`, and `-logical` arguments.

Note:

If you do not specify any of the -prefix argument, the -postfix argument, or the -map argument, the interface ports are not added to the parent design/component. Then, you can use the [set_interface_port](#) command to add the ports.

`-exclude {<logical-name>}`

(Optional) Specifies the comma-separated name list of interface ports that are not added to the parent design/component.

You can specify the comma-separated list enclosed within double quotes (") or curly braces ({}).

Note:

The interface port specified with the -exclude argument is not added even if its mapping is specified using the -map argument. In this case, you can use the [set_interface_port](#) command to add the ports.

Note:

The -exclude argument is given priority over the -logical argument.

`-type <MASTER | SLAVE | MONITOR | SYSTEM>`

(Optional) Specifies the type of the interface instantiation as MASTER or SLAVE or MONITOR or SYSTEM.

The value of the -type argument specifies the property of interface describing the role of the interface in the bus protocol.

By default, the -type argument is set to MASTER. If you specify -type MONITOR, all the interface ports are added as input ports (of type IN), including the inout ports; and all the interface ports will be read-only. The MONITOR option also implies that the interface is added in the testbench partition and not in the design partition.

`-groupName <system-group-name>`

(Optional) Specifies the name of the system interface being mirrored if the interface is instantiated as type SYSTEM.

The specified name must match with the group name present on one or more ports in the corresponding bus definition.

`-mirror <YES | NO | MONITOR>`

(Optional) Specifies that the interface ports should be added with the same direction or the opposite direction as in the interface library.

Note:

Inout ports are always added as inout ports.

If you specify -mirror YES, interface input ports are added as output ports in the parent design/component or vice-versa. If you specify -mirror NO, interface input (output) ports are

added as input (output) ports in the parent design/component. You can also specify direct instead of NO with the -mirror argument. The NO and direct values perform the same function.

Note:

The -mirror YES | NO argument is useful only when the interface type is set to MASTER or SLAVE using the -type argument. If you set the -mirror argument to YES, and the -type argument is set to MONITOR, then the -mirror setting is ignored.

If you specify -mirror MONITOR, all the interface ports are added as input ports (of type IN), including the inout ports.

The default value for the -mirror option is NO.

Note:

The MONITOR value of the -mirror argument is supported only for backward compatibility.

`-short_name <short-name>`

(Optional) Specifies the short name of the interface instance.

`-description <description>`

(Optional) Specifies the description of the interface instance.

This argument also accepts unicode characters.

`-control_clock <clk-name>`

(Optional) Specifies the name of the control clock for the interface instance.

The specified control clock must be already defined for the parent design/component.

`-control_reset <rst-name>`

(Optional) Specifies the name of the control reset for the interface instance.

The specified control reset must be already defined for the parent design/component.

`-power_domain <pd-name>`

(Optional) Specifies the voltage of the interface instance.

`-voltage <voltage-value>`

(Optional) Specifies the voltage value for the power domain of the interface instance.

`-open <true | false>`

(Optional) Specifies whether the interface of the component can be left unconnected in the design where it has been instantiated.

In the RTL health check report, if the interface is left unconnected, ports of the interface are shown as unconnected. However, if the interface is marked open, the ports of the interface are shown as Intentional Open.

In the generated RTL, if the interface is left unconnected, ports of the interface are shown connected to the net named UNINTENTIONAL_OPEN_*.

However, if the interface is marked open, the ports of the interface are shown connected to the net named OPEN_*.

`-visibility <visibility-condition>`

(Optional) Specifies the comma-separated list of conditions under which the object would be visible to the generators.

`-attribute_names {space-separated-list} -attribute_values {space-separated-values}`

(Optional) Specifies attribute values that should override the values of the specified attributes present in an interface library while instantiating an interface.

The `-attribute_names` argument accepts a space-separated list of attribute names and the `-attribute_values` argument accepts a space-separated list of attribute values.

`-connectionRequired <true | false>`

(Optional) Specifies how a bus interface is connected when a component is added to a design already containing a bus owner.

`-endianness <big | little>`

(Optional) Specifies the endianness, big or little, of the bus interface.

`-bitSteering <on | off>`

(Optional) Specifies whether bit steering should be used to map an interface on a bus of a different data width.

Examples

The following example adds an interface, `Intf1`, with master interface library `intf_COMP`:

```
add_interface -name Intf1 -interface_ref intf_COMP -prefix IN_ -mirror
YES -voltage 5
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case1` directory.

add_interface_mapping

Maps logical to actual interface port before inferring interfaces on a module

Usage

```
add_interface_mapping
{
  -component <comp-name>
  [ -version <comp-version> ]
  [ -library <comp-library> ]
  [ -vendor <comp-vendor> ] }
{
  -interface <intf-name>
  [ -version <intf-version> ]
  [ -library <intf-library> ]
  [ -vendor <intf-vendor> ] }
-interface_instance <inst-name>
[ -type <MASTER | SLAVE> ]
{-logical <logical-port> -actual <actual-port>}*
```

Description

The `add_interface_mapping` command maps logical to actual interface port before inferring interfaces on a module.

After specifying interface mapping on a module by using this command, if you specify the [infer_interfaces](#) command on that module, GenSys instantiates the interface with properties given through the `add_interface_mapping` Tcl command and does the logical to actual port mapping.

In addition, GenSys tries to infer the logical ports that were left out in this mapping based on the rules specified by the user or default rule.

Arguments

The `add_interface_mapping` command has the following arguments:

`-component <comp-name>`

Specifies the name of the component on which you want to perform interface mapping.

`-version <comp-version>`

(Optional) Specifies the version of the specified component.

`-library <comp-library>`

(Optional) Specifies the library of the specified component.

`-vendor <comp-vendor>`

(Optional) Specifies the vendor of the specified component.

`-interface <intf-name>`

Specifies the name of the interface.

`-version <intf-version>`

(Optional) Specifies the version of the interface.

`-library <intf-library>`

(Optional) Specifies the library of the interface.

`-vendor <intf-vendor>`

(Optional) Specifies the library of the interface.

`-interface_instance <inst-name>`

Specifies interface instance name.

`-type <MASTER | SLAVE>`

(Optional) Specifies the interface type as MASTER or SLAVE.

`-logical <logical-port>`

Specifies the logical port of the interface.

`-actual <actual-port>`

Specifies the actual port of the interface.

Note:

You must specify atleast one pair of -logical and -actual arguments.

Examples

Example 1

Consider the following command:

```
add_interface_mapping -component comp -interface intf1
-interface_instance myIntf1_inst -type MASTER -logical L1
```

```
-actual {A_L1[2:0]} -logical L2 -actual A_L2 -logical L3
-actual {A_L3[1]} -logical L6 -actual A_L6
```

The above command specifies logical to actual mapping for the comp module. The library interface is intf1 and is of type MASTER. This interface is instantiated on the module with the name, myIntf1_inst. Here, slices of actual ports have been mapped. For example, logical port L1 is mapped to the slice [2:0] of actual port A_L1.

Example 2

Consider an interface, intf1, which has eight logical ports, L1, L2, L3, L4, L5, L6, L7, and L8. Similarly, the component comp has eight actual ports, A_L1, A_L2, A_L3, A_L4, A_L5_port, A_L6_port, A_L7_port, and A_L8_port.

Now consider that you specify the following Tcl command:

```
add_interface_mapping -component comp -interface intf1
-interface_instance myIntf1_inst -type MASTER -logical L1
-actual {A_L1[2:0]} -logical L2 -actual A_L2 -logical L3
-actual {A_L3[1]} -logical L4 -actual A_L4 make_rule -name R1
-prefix A_ -postfix _port apply_rule -component comp
-interface intf1 -rule R1 infer_interfaces
```

The above command specifies mappings for L1, L2, L3, and L4 and given a rule which matches L5, L6, L7, and L8. Now, by using mapping specified by in this command, an interface instance with name myIntf1_inst is instantiated on comp and L1, L2, L3, and L4 are mapped by it. Since a rule is also specified for remaining ports which can be matched through rule, L5, L6, L7, and L8 also get mapped in same myIntf1_inst interface instance.

add_model_view

Adds a model view to the most recently active design

Usage

```
add_modelview
-name <name>
[ -language <language> ]
[ -hierarchy_name <name> ]
[ -hierarchy_vendor <vendor> ]
[ -hierarchy_version <version> ]
[ -hierarchy_library <library> ]
[ -description <description> ]
```

```
[ -filesets <comma-separated list> ]
```

Description

The `add_model_view` command adds a model view to the most recently active design.

A view refers to a group of file sets. Each design/component in GenSys can have multiple views. Support for multiple views is important because various generators, such as RTL export, register RTL, SystemC export, etc., might end up creating various views for the same component.

Arguments

The `add_model_view` command has the following arguments:

`-name <name>`

Specifies the name of the model view.

`-language <language>`

(Optional) Specifies the language of the model view.

`-hierarchy_name <name>`

(Optional) Specifies the name of the IP-XACT design in case if the component is a subsystem/design.

`-hierarchy_vendor <vendor>`

(Optional) Specifies the vendor of the IP-XACT design in case if the component is a subsystem/design.

`-hierarchy_version <version>`

(Optional) Specifies the version of the IP-XACT design in case if the component is a subsystem/design.

`-hierarchy_library <library>`

(Optional) Specifies the library of the IP-XACT design in case if the component is a subsystem/design.

`-description <description>`

(Optional) Specifies the description of the view.

`-filesets <comma-separated list>`

(Optional) Specifies a comma-separated list of file sets associated with this view.

add_netname

Assigns a net name to existing connections

Usage

```
add_netname -netname <net-name>
<from-options>::=
  -instance <inst-name> -pin <pin-name>
  | -port <port-name>
  -lsb <lsb>
  -msb <msb>

<to-options>::=
  -instance <inst-name> -pin <pin-name>
  | -port <port-name>
  -lsb <lsb>
  -msb <msb>
```

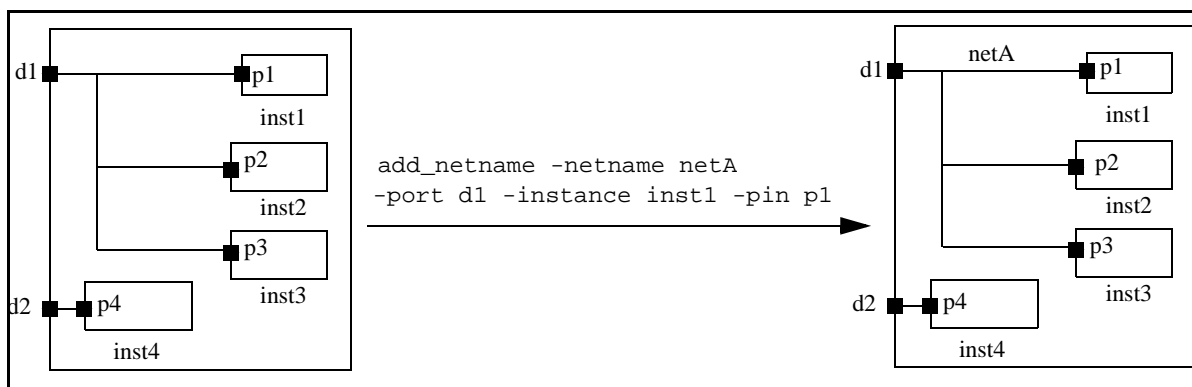
Description

The add_netname command assigns a net name to existing connections.

While using this command, you can:

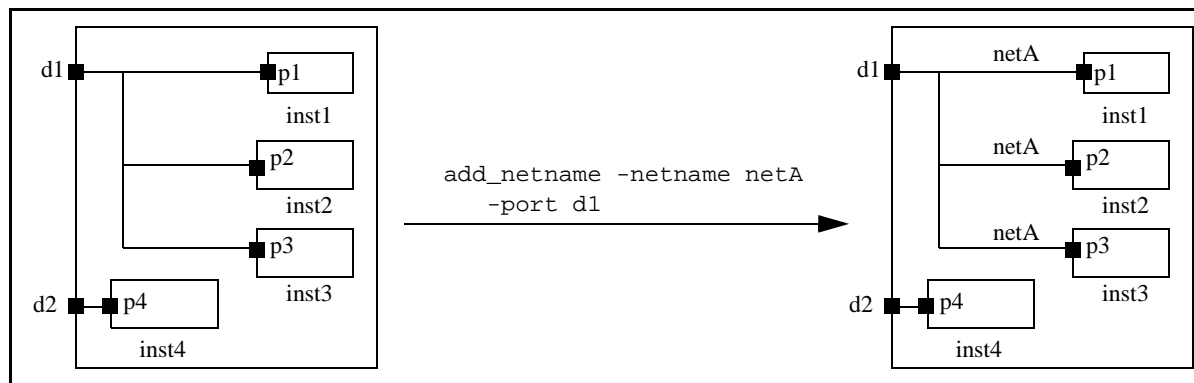
- Specify both ends of a connection, that is, you can specify both *<from-options>* and *<to-options>*.

In this case, GenSys assigns the specified net name to the connection between the specified ends. This is explained in the following figure:



- Specify the driver of a connection, that is, you can specify only *<from-options>*.

In this case, GenSys assigns the same net name to all connections through the specified driver. This is explained in the following figure:



For details, see [Examples](#).

Also see [Priority of Net Names](#).

Arguments

The `add_netname` command has the following arguments:

`-netname <net-name>`

Specifies a net name that will be used to identify a connection between ports/pins.

`-instance <inst-name>`

Specifies the name of an instance.

`-pin <pin-name>`

Specifies the pin of the specified instance.

`-port <port-name>`

Specifies the name of a port.

`-lsb <lsb>`

Specifies the LSB of the specified port/pin.

`-msb <msb>`

Specifies the MSB of the specified port/pin.

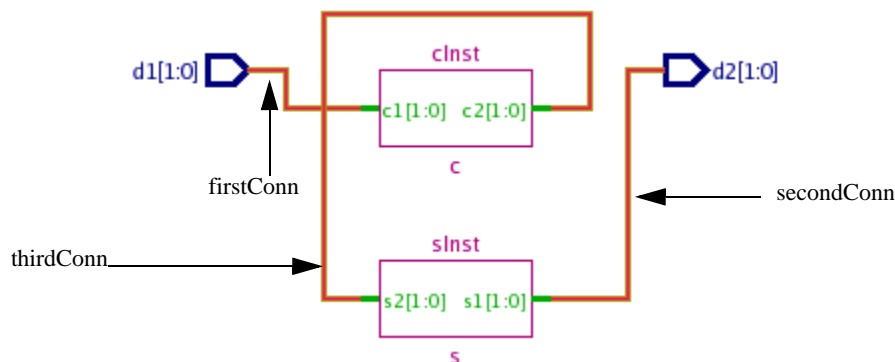
Examples

Consider the following commands:

```
new component c
add_port -name c1 -direction IN -lsb 0 -msb 1
add_port -name c2 -direction OUT -lsb 0 -msb 1

new component s
add_port -name s1 -direction OUT -lsb 0 -msb 1
add_port -name s2 -direction IN -lsb 0 -msb 1
new design d
add_port -name d1 -direction IN -lsb 0 -msb 1
add_port -name d2 -direction OUT -lsb 0 -msb 1
add_instance -name cInst -master c
add_instance -name sInst -master s
add_connection -instance cInst -pin c1 -port d1
add_connection -instance sInst -pin s1 -port d2
add_connection -instance cInst -pin c2 -instance sInst
    -pin s2
add_netname -netname firstConn -port d1
add_netname -netname SecondConn -instance sInst -pin s1
    -port d2
add_netname -netname thirdConn -instance cInst -pin c2
run GenerateVerilog
```

After running the above commands, the following schematic appears:



Following is the generated RTL for the above design:

```
module d (d1,
          d2);
input    [1:0] d1;
output   [1:0] d2;
wire     [1:0] thirdConn;
wire     [1:0] firstConn;
```

```

wire    [1:0]    secondConn;
    c    cInst (
        .c1(firstConn),    //I:2
        .c2(thirdConn)    //O:2
    );
    s    sInst (
        .s1(secondConn),    //O:2
        .s2(thirdConn)    //I:2
    );
    assign firstConn = d1;
    assign d2 = secondConn;
endmodule

```

add_parameter

Adds a parameters to the active design or component

Usage

```

add_parameter
    -name <param-name>
    -type <type-name>
    [ -value <param-value>* ]
    [ -description <description>* ]
    [ -validrange <range>* ]
    [ -rtl <Y | N> ]
    [ -level <level> ]
    [ -isactive <perl-func-name> ]
    [ -const <Y | N> ]
    [ -vhdl_library <library-name> ]
    [ -vhdl_package <pkg-name> ]
    [ -macro_visibility <macro-condition> ]
    [ -pragma_name <pragma-name> ]
    [ -rtl_visibility {TRUE|FALSE|<logical-expression>} ]

```

Description

The add_parameter command adds the specified parameter to the active design/component.

Parameters also support structured types in the form of arrays and records.

Arguments

The add_parameter command has the following arguments:

`-name <param-name>`

Specifies the name of the parameter being added.

`-type <type-name>`

Specifies the type of the parameter being added.

The parameter type can be simple data types like INT, STRING, and BOOL; or can be complex types that are created using the [add_type](#) command. You should not directly add complex types in `add_parameter` command. You must first define a complex type using the [add_type](#) command.

Note:

HEX is not a valid type for parameter. However, you can specify HEX values for INT parameters.

`-value <param-value>`

(Optional) Specifies the value of the parameter being added.

Use the `-value` argument to specify the values of the parameters of simple types.

Note:

For complex parameters, use the [set_parameter](#) command to set values.

`-description <description>`

(Optional) Specifies the description of the parameter being created.

This argument also accepts unicode characters.

`-validrange <range>`

(Optional) Specifies the valid range for INT types.

See [Specifying Valid Range](#) for more details.

Note:

Valid range may be specified only for parameters of INT type.

`-rtl <Y | N>`

(Optional) Specifies whether the parameter being added should be used in RTL being generated.

Note:

Parameters of type INT, ENUM, and BOOL or parameters that are derived from INT, ENUM, and BOOL can only be specified as RTL parameter.

`-level <level>`

(Optional) Specifies the frame level of the parameter being added.

Parameters are displayed in cascading frames of the Parameter Configuration Wizard. This setting indicates the frame number in which the particular parameter is to be displayed.

The level can be any non-zero positive integer number.

Parameters specified with level 1 are not dependent on any parameters. Parameters specified with level 2 are dependent on some of the parameters specified with level 1. Similarly, parameters specified with level 3 may be dependent on some of the parameters specified with level 1 and/or level 2.

Note:

You cannot specify dependency between the parameters specified with the same level.

For more information on specifying levels for parameters, refer to the *GenSys User Guide* (in the *Specifying Levels and Perl Functions* section in *Creating Components in GenSys* chapter).

`-isactive <perl-func-name>`

(Optional) Specifies if the parameter needs to be made active or inactive depending on the value returned by the specified Perl function.

The specified Perl function must return either 0 or 1. If the value returned by the Perl function is 1 (that is, true), the parameter is made active. If the value returned by the Perl function is 0 (that is, false), the parameter is made inactive.

Note:

Inactive parameters are not visible in the Parameter Configuration Wizard.

Note:

The parameter type `Treetable` is passed as an argument to the Perl function inside the parameter.

The Perl function takes four arguments - current table pointer, current row number, current column number, and type tree.

For more information on specifying Perl functions for parameters, refer to the *GenSys User Guide* (in the *Specifying Levels and Perl Functions* section in the *Creating Components in GenSys* chapter) and the *GenSys Customization Guide*.

`-const <Y | N>`

(Optional) Specifies if the type of parameter is of type `const`. If set to Y, type of the parameter is set to `const`. If you try to change the value of a `const` type parameter, the `set_parameter` Tcl command flags an error. By default, this argument is set as N.

Note:

`const` type parameters are not displayed in the Reconfigure Instance wizard to avoid any modifications in their values.

`-vhdl_library <library-name>`

(Optional) Specifies the library name for the const type parameter.

`-vhdl_package <package-name>`

(Optional) Specifies the package name for the const type parameter.

`-macro_visibility <macro-condition>`

(Optional) Specifies a macro condition in which a parameter should be visible in the generated Verilog RTL.

Based on the specified macro condition, the parameter is placed inside an appropriate Verilog conditional pre processor directive, such as ``ifdef` in the generated Verilog RTL.

Refer to [Example 4](#).

`-pragma_name <pragma-name>`

(Optional) Specifies pragma conditions on parameters.

When you apply pragmas on a parameter, then during RTL export, GenSys dumps that parameter under the pragma specified by this argument.

For example, consider that you set this argument to `SYNTHESIS`, as shown in the following command:

```
add_parameter -name P1 -type INT -value 2 -pragma_name SYNTHESIS
```

In this case, the exported Verilog RTL is as follows:

```
//synopsys synthesis_off
parameter    P1    =    2;
//synopsys synthesis_on
```

Similarly, if you set this argument to `TRANSLATE`, GenSys dumps that parameter under the `synopsys translate_off/on` pragmas.

`-rtl_visibility {TRUE|FALSE|<logical-expression>}`

(Optional) Controls the visibility of the parameter in the generated RTL.

You can set this argument to any of the following values:

Value	Description
TRUE	Makes the parameter visible in the generated RTL
FALSE	Hides the parameter in the generated RTL

Value	Description
logical expression	<p>Hides or unhides the parameter based on the evaluated value of the logical expression.</p> <p>This logical expression uses the visibility variable set by the add_hier_connection Tcl command. For example, consider that the M1 visibility variable created by the add_hier_connection Tcl command is set to FALSE. Now, to use this variable on the parameter being added (say p1) by the <code>add_parameter</code> Tcl command, specify the logical expression as <code>{=[d M1]}</code> to the <code>-rtl_visibility</code> argument.</p> <p>Now, when you generate the RTL, the p1 parameter will be hidden in the generated RTL.</p> <p>For more details, see Example 2.</p>

Examples

Example 1

The following example adds a parameter named p4 of type INT with value 0x55 and a valid range between 0x01 and 0xff:

```
add_parameter -name p4 -type INT -validrange 0x01:0xff -value 0x55
```

Example 2

Consider the following command:

```
add_parameter -name D2 -type INT -validrange 0:32 -value 35 -rtl Y
```

When you run the above command, GenSys reports the following error:

```
Parameter Value 35 is outside the given range '0:32' for the parameter D2
```

To fix this error, specify the command as follows:

```
add_parameter -name D2 -type INT -validrange 0:32 -value 5 -rtl Y
```

Example 3

When the Verilog is netlisted, the parameter is dumped along with the macro declaration. Assume you have specified:

```
add_parameter -name P1 ... -macro_visibility {X}
```

In the GenSys GUI, this is visible in property editor under the *macro_visibility* column.

The Verilog RTL for this parameter is:

```
...`ifdef X  parameter P1...`endifOr`ifndef Xparameter P1...`endif
```

This is based on the *macro_visibility* argument that is set as X or !X.

add_port

Adds a port to the active design or component

Usage

```
add_port
  -name <port-name>
  [ -direction <IN | OUT | INOUT | PHANTOM> ]
  [ -port_type <port-type> ]
  [ -lsb <lsb> ]
  [ -msb <msb> ]
  [ -align <LSB | MSB> ]
  [ -short_name <short-name> ]
  [ -clock_rate <clk-rate> ]
  [ -power_domain <pd-name> ]
  [ -voltage <voltage-value> ]
  [ -size <size> ]
  [ -default_value <value> ]
  [ -control_clock <clk-name> ]
  [ -control_reset <rst-name> ]
  [ -description <description> ]
  [ -open <TRUE | FALSE> ]
  [ -visibility <visibility-condition> ]
  [ -vhdl_port_type STD_LOGIC | STD_ULOGIC | BIT |
    SIGNED | UNSIGNED | <complex-type> ]
  [ -lsb_bit <left | right> ]
  [ -net_name <net-name> ]
  [ -net_prefix <net-prefix> ]
  [ -hierarchy <hier-module> ]
  [ -elaborate <TRUE | FALSE> ]
  [ -allLogicalDirectionsAllowed <true | false> ]
  [ -macro_visibility <macro-condition> ]
```



```
[ -pragma <TRANSLATE | SYNTHESIS> ]
```

Description

The `add_port` command adds a port to the active design or component.

If the new port added is already present in the design or component, the specified argument values, if different are appended to the existing port.

Arguments

The `add_port` command has the following arguments:

`-name <port-name>`

Specifies the name (string) of the port.

`-direction`

(Optional) Specifies the direction of the port as IN (for input ports), OUT (for output ports), INOUT (for inout ports), or PHANTOM (for phantom ports). If you do not specify the `-direction` argument, the port is assumed to be an inout port.

If you specify the direction as PHANTOM, GenSys elaborates the connections to/from the phantom ports and makes a direct connection between the source and sink. Phantom ports are skipped in the generated RTL. See [Example 3](#)

However, please note the following points:

- GenSys supports the IP-XACT dump/restore of phantom ports.
- GenSys supports phantom ports only on components/designs.
- GenSys supports only adhoc connectivity to phantom ports.

`-port_type <port-type>`

(Optional) Specifies the port type as one of the following:

ADDR	ANALOG	CLK	CONFIG
CONTROL	CORE	CORE_INPUT	CORE_INPUT_ FN
CORE_INPUT_T EST	CORE_OEN_FN	CORE_OEN_TE ST	CORE_OUTPU T

CORE_OUTPUT _FN	CORE_OUTPUT_TE ST	DATA	DFT
EVT	FUNCTION_IN	IO	IO_OEN
IO_PAD	IO_PULLEN	IO_SELECT	OSCCTL
PAD_INPUT	PAD_OEN	PAD_OUTPUT	PVT
PWRDN	RST	SCANIN	SCANOUT
SLEWRATE	TEST_IN	TIEOFF	PRIMARY_INP UT
PRIMARY_OUTP UT			

The PRIMARY_INPUT port type is considered to be the valid port type for the input pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells

The PRIMARY_OUTPUT port type is considered to be the valid port type for the output pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells.

If you do not specify the -port_type argument, the port is marked as UNDEFPORT.

-lsb <lsb>

(Optional) Specifies the LSB of the port.

Use the -lsb argument to specify the LSB of vector ports.

See [Specifying Values for Integer Arguments](#) for more details.

Note:

You must specify both the -msb and -lsb arguments. If either of them is specified, GenSys reports an error.

-msb <msb>

(Optional) Specifies the MSB of the port.

Use the -msb argument to specify the MSB of vector ports.

See [Specifying Values for Integer Arguments](#) for more details.

Note:

You must specify both the -msb and -lsb arguments. If either of them is specified, GenSys reports an error.

`-align <LSB | MSB>`

(Optional) Specifies the order of scalar-level connectivity for the (vector) port.

The default value of the `-align` argument is MSB and the port is connected MSB onwards.

`-short_name <short-name>`

(Optional) Specifies the short name of the port.

`-clock_rate <clk-rate>`

(Optional) Specifies the clock rate of the port.

`-power_domain <pd-name>`

(Optional) Specifies the power domain of the port.

`-voltage <voltage-value>`

(Optional) Specifies the voltage value for the power domain of the port.

`-size <size>`

(Optional) Specifies the size of the port.

Use the `-size` argument to specify the size of vector ports when you do not want to specify the port size in terms of LSB/MSB using the `-lsb/-msb` arguments. Then, the port LSB is assumed to be 0 and port MSB is assumed to be (size-1).

If you have specified the LSB/MSB values of the port using the `-lsb/-msb` arguments, the size value is automatically calculated and displayed. You cannot then modify the size value using the `-size` argument.

`-default_value <value>`

(Optional) Specifies the default value of the port.

If there is no connections specified for the port, the port is assumed to be tied to the specified default value.

For a component, the default port value is calculated on the basis of the following precedence:

- If actual port has a default value, it will get the highest precedence.
- If the actual port does not have a default value and no default value is specified while mapping to the interface port through the [set_interface_port](#) Tcl command, the default value of the logical port is set as the default value of the actual port.
- If the actual port does not have default value but a default value is specified while mapping to the interface port through the `set_interface_port` Tcl command, that default value is set for the actual port.

`-control_clock <clk-name>`

(Optional) Specifies the name of the control clock for the port.

The specified control clock must be already defined for the design or component.

`-control_reset <rst-name>`

(Optional) Specifies the name of the control reset for the port.

The specified control reset must be already defined for the design or component.

`-description <description>`

(Optional) Specifies the description of the port.

This argument also accepts unicode characters.

`-open <TRUE | FALSE>`

(Optional) Specifies whether the port can be left unconnected in the RTL being generated.

The default value of the `-open` argument is `FALSE`. This specifies that the port must be connected in the design or for each instantiation of the component in the design.

Set the value of the `-open` argument as `TRUE` to specify that the port can be left unconnected

Unconnected ports that are not marked as `OPEN` using the `-open` argument are reported by the RTL-level Connectivity Check.

`-visibility <visibility-condition>`

(Optional) Specifies the comma separated string of conditions under which the object would be visible to the generators.

For the RTL flow, use the `-macro_visibility` argument.

`-vhdl_port_type`

(Optional) Specifies the VHDL type of a port. This information is used by VHDL RTL generator while dumping ports.

The default value of VHDL port type is `STD_LOGIC` and the port is dumped as `STD_LOGIC_VECTOR` or `STD_LOGIC`. You can change the VHDL port type to `STD_ULOGIC`, `BIT`, `SIGNED`, `UNSIGNED`, or any complex type.

`-lsb_bit <left | right>`

(Optional) Specifies the bit endianness of the port. By default, the value of this argument is set to `right`, which means that while specifying a port (`P[<left>:<right>]`), the right-most

bit is considered as LSB and the left-most bit is considered as MSB. This type of endianness is also known as little bit-endian.

If you set the value of this argument as left, GenSys considers the left-most bit of the port as LSB and the right-most bit of the port as MSB. This type of endianness is also known as big bit-endian.

Note:

The bit endianness specified by this Tcl command overrides the default bit endianness specified by the [set_lsb_bit](#) Tcl command for the specified port.

`-net_name <net-name>`

(Optional) Specifies the net name for the tieoff connection (default/temporary/forced). This net name appears in the generated RTL (Verilog/VHDL).

If multiple connections (on different or same pin) have same net name, GenSys creates a vector net. The size of this vector net is same as the number of connections having the same net name. If unique net name is specified for each tieoff connection, GenSys creates a scalar net.

`-net_prefix <net-prefix>`

(Optional) Specifies the net prefix for the tieoff connection (default/temporary/forced). The net prefix specified by this argument gets prepended to the net name, and this net name then appears in the generated RTL (Verilog/VHDL).

If multiple connections (on same pin) have same net prefix, GenSys creates a vector net. The size of this vector net is same as the number of connections having the same net prefix. If unique net prefix is specified for each tieoff connection, GenSys creates a scalar net.

`-hierarchy <hier-module>`

(Optional) Specifies the partition hierarchy where extra ports (ECO ports) can be added during RTL generation. Here, *<hier-module>* is created dynamically.

GenSys does not process these extra ports (ECO ports) till RTL netlisting. It processes these ports on the RTL object model during RTL generation.

Note:

For details on ECO ports, refer to the section, [Creating ECO Ports](#).

`-elaborate <TRUE | FALSE>`

(Optional) Specifies whether the port being added should be elaborated.

If this argument is set to TRUE, GenSys performs the following actions:

- Creates scalar-level ports for vector ports

- Creates terminal for the port in the corresponding component instances of the component in which the port is added.

By default, this argument is set to FALSE.

```
-allLogicalDirectionsAllowed <true | false>
```

(Optional) Specifies whether a port can be mapped to a port in the abstraction definition (logical port) with a different direction.

```
-macro_visibility <macro-condition>
```

(Optional) Specifies a macro condition in which a port should be visible in the generated Verilog RTL.

Based on the specified macro condition, the port is placed inside an appropriate Verilog conditional pre processor directive, such as `ifdef in the generated Verilog RTL.

Refer to [Example 4](#).

```
-pragma <TRANSLATE | SYNTHESIS>
```

(Optional) Specifies pragma conditions on ports.

When you apply pragmas on a port, then during RTL export, GenSys dumps that port under the pragma specified by this argument.

For example, consider that you set this argument to TRANSLATE, as shown in the following command:

```
add_port -name P1 -direction IN -pragma TRANSLATE
```

In this case, the exported Verilog RTL is as follows:

```
//synopsys translate_off
Input P1;
//synopsys translate_on
```

Similarly, if you set this argument to SYNTHESIS, GenSys dumps that port under the synopsys synthesis_off/on pragmas.

Examples

Example 1

The following example adds the port named P01 with LSB 0 and MSB 31 and direction IN:

```
add_port -name P01 -lsb 0 -msb 31 -direction IN
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

Example 2

The following example adds the ports: D1, D2, and D3, with VHDL port types as BIT, STD_LOGIC_VECTOR, STD_ULOGIC, respectively:

```
new design top
add_port -name D1 -lsb 0 -msb 2 -direction OUT
      -vhdl_port_type BIT
add_port -name D2 -lsb 0 -msb 2 -direction OUT
add_port -name D3 -lsb 0 -msb 2 -direction OUT
      -vhdl_port_type STD_ULOGIC
save top
```

For the above given design, VHDL RTL will be generated as given below:

```
ENTITY top is
  PORT (      D1  : OUT  bit_vector (2 downto 0);
          D2  : OUT  STD_LOGIC_VECTOR (2 downto 0);      D3  : OUT
          std_ulogic_vector (2 downto 0) );END top;
```

Example 3

The following example creates a phantom port, and makes connection using that port:

```
new component comp
add_port -name p1 -lsb 0 -msb 3 -direction INOUT
save
new design des
add_instance -name i1 -master comp
add_instance -name i2 -master comp
add_port -name dp1 -lsb 0 -msb 3 -direction PHANTOM

#phantom port connections
add_connection -instance i1 -pin p1 -lsb 0 -msb 3 -port
dp1 -lsb 0 -msb 3
#instance-pin to phantom port connection
add_connection -instance i2 -pin p1 -lsb 0 -msb 3 -port
dp1 -lsb 0 -msb 3
#instance-pin to phantom port connection
save
```

In the generated RTL of the above example, GenSys makes a direct connection between i1.p1 and i2.p1, and the phantom port, dp1, does not appear in the generated RTL.

Example 4

Consider the following commands:

```
new component master1
add_port -name a -direction IN -align MSB -lsb_bit RIGHT-macro_visibility
{X:A}
add_port -name b -direction OUT -align MSB -lsb_bit
RIGHT-macro_visibility {X:A}
```

The above commands create the ports a and b and add a configurability for them by using the -macro_visibility argument. This configurability indicates that when both the macros X and A are TRUE, the ports a and b should be visible.

Based on this configurability, ports are placed under appropriate `ifdef statements in the generated Verilog RTL, as shown below:

```
module master1 (
    `ifdef X
        `ifdef A
            a,
            b,
        `endif
    `endif
);
`ifdef X
    `ifdef A
        input    a;
        output   b;
    `endif
`endif
endmodule
```

add_to_plane

Adds an object to a plane.

Usage

```
add_to_plane
    -plane_name <plane-name>
    [ -component <comp-name> ]
    [ -component <comp-name> -port <port-name> ]
    [ -instance <inst-name> ]
```



```
[ -instance <inst-name> -pin <pin-name> ]  
[ -interface <intf-lib-name> ]  
[ -instance <inst-name> -interface <intf-inst-name> ]  
[ -move ]
```

Description

The `add_to_plane` command adds an object to a plane. If the plane does not exist, the command creates the plane. The planes appear in the Planes pane of the Modular Schematic window. You can show or hide these planes to show/hide the objects belonging to them.

Arguments

The `add_to_plane` command has the following arguments:

`-plane_name <plane-name>`

(Optional) Name of the plane to which an object is to be added.

`-component <comp-name>`

(Optional) Name of the component to be added to the plane.

`-component <comp-name> -port <port-name>`

(Optional) Name of the port (of the specified component) to be added to the plane.

`-instance <inst-name>`

(Optional) Name of the instance to be added to the plane.

`-instance <inst-name> -pin <pin-name>`

(Optional) Name of the pin (of the specified instance) to be added to the plane.

`-interface <intf-lib-name>`

(Optional) Name of the interface library to be added to the plane.

`-instance <inst-name> -interface <intf-inst-name>`

(Optional) Name of the interface library (of the specified instance) to be added to the plane.

`-move`

(Optional) If an object is already added to a plane, the `add_to_plane` command updates the plane name and object type in the existing plane. If the `-move` argument is not used,

the object is added in the plane. As an effect of this, the object will now exist in more than one plane.

Examples

1. The following command adds the instance, AXI_1_2_3, and all interfaces and ports on that instance to the plane, BUS:

```
add_to_plane -plane_name BUS -instance AXI_1_2_3
```

2. The following command places the interface, intf1, of an instance to a plane, CLOCK:

```
add_to_plane -plane_name CLOCK -instance inst1 -interface intf1
```

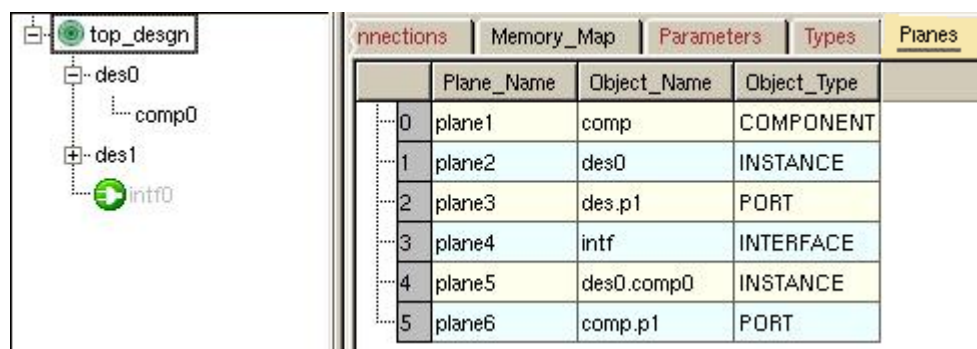
In this case, all the instances of the interface, intf1, are also placed in the plane, CLOCK.

Additional Details

- For each plane added by using the add_to_plane command, a new row is inserted in the Planes schema. For example, consider the following Tcl commands:

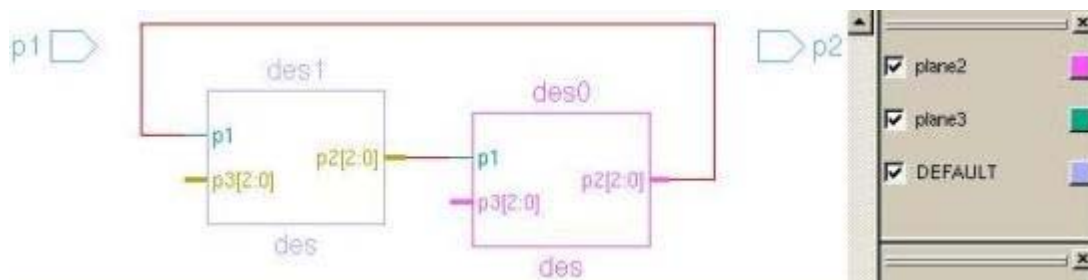
```
add_to_plane -plane_name plane1 -component comp
add_to_plane -plane_name plane2 -instance des0
add_to_plane -plane_name plane3 -instance des -pin des.p1
add_to_plane -plane_name plane4 -interface intf
add_to_plane -plane_name plane5 -instance des0.comp0
add_to_plane -plane_name plane6 -instance comp -pin comp.p1
```

The above Tcl commands populate the Planes table in GUI, as shown below:



	Plane_Name	Object_Name	Object_Type
0	plane1	comp	COMPONENT
1	plane2	des0	INSTANCE
2	plane3	des.p1	PORT
3	plane4	intf	INTERFACE
4	plane5	des0.comp0	INSTANCE
5	plane6	comp.p1	PORT

The Modular Schematic for the above design is shown below:



To know more details on how the objects belonging to a plane will be shown in the Modular Schematic window, refer to the *Modular Schematic Window* section in the *GenSys User Guide*.

- An instance of a component/interface inherits the plane property of its parent component/interface. For ports, GenSys uses the plane property of the instance (component/interface) on which that port is present. However, if you explicitly assign the component/interface instance or port to another plane using the `add_to_plane` Tcl command, the original plane property of that component/interface instance or port gets overridden by the property of the newly assigned plane.

Consider the following example in which a component, C1, is placed in plane, PlaneA:

```
add_to_plane -plane_name PlaneA -component C1
```

When you create an instance (say C1_inst) of C1, that instance automatically inherits the plane property of the plane, PlaneA. Now, consider that you assign C1_inst to another plane, PlaneB, by using the `add_to_plane` Tcl command, as shown below:

```
add_to_plane -plane_name PlaneB -instance C1_inst
```

After executing the above command, GenSys overrides the plane property of C1_inst with the property of PlaneB.

Consider another example in which an interface, Intf1, is placed in plane, PlaneA, as shown below:

```
add_to_plane -plane_name PlaneA -interface Intf1
```

When you create an instance (say Intf1_inst) of Intf1 in component, C1, this instance automatically inherits the plane property of the plane, PlaneA, to which Intf1 belongs. Now, consider that you assign Intf1_inst to another plane, PlaneB, by using the `add_to_plane` Tcl command, as shown below:

```
add_to_plane -plane_name PlaneB -instance Intf1_inst
```

After executing the above command, GenSys will override the plane property of Intf1_inst with the property of PlaneB.

Note:

Details of planes are saved in the Planes table, which is hidden by default. When the design is saved, these details are also saved.

add_type

Creates a user-defined parameter type

Usage

The add_type Tcl command can be specified in the following ways:

```
add_type
  -name <type-name>
  -type <type>
```

Description

The add_type command creates complex or non-basic parameter types.

A complex type can be an array or a record of type INT, BOOL, STRING, ENUM. Types may also reference other previously defined types.

A complex parameter can be added by first creating a complex type using add_type command and then adding the parameter of that complex type using the [add_parameter](#) command.

After adding a parameter, its value can be set using the [set_parameter](#) command.

The add_type command is also used to create complex port types. A complex port type can be created in the following way:

- Create a complex type by using the add_type command. The new type becomes visible in the Types tab of a design/component table in GUI.
- Create a port using the [add_port](#) command. Specify the model information for this port using the [add_model_view](#) command. Using this command, you can specify complex type for the active port. The model details of a port are visible in the model subtable of the Ports table.

If you are importing the component details from IP-XACT XML files, you need to first add types using the add_type command. This helps GenSys in recognizing the new complex types. Next, you can import the IP-XACT XML files. In these XML files, left (MSB) and right (MSB) values specified with the signal (port) are associated with the type and not with the port. LSB and MSB of the port are calculated after flattening.

Please note the following:

- Complex types can be used only for component instances and not for design instances.
- Support for complex types is available for VHDL RTL generation only.
- Complex types with only one scalar member of standard type are not supported. For example,

```
add_type -name T1 -type BIT
add_port -name p1 -direction IN -vhdl_port_type T1
//this does not work
```

Arguments

The `add_type` command has the following arguments:

`-name <type-name>`

Specifies the name of the parameter type being created.

`-type <type>`

Specifies the type of the new parameter. It can take any of the following values:

INT | BOOL | STRING | STD_LOGIC | STD_ULOGIC | BIT | SIGNED | UNSIGNED |
user-defined type created previously using `add_type`

Examples

The following example creates parameter type T2 of type STRING and then creates a parameter based on that type:

```
add_type -name T2 -type STRING
add_parameter -name Sparam -type T2
```

The following example creates a complex type T1 of type STD_LOGIC before importing the component details from IP-XACT XML files in which left (LSB) is 3, right (MSB) is 0, and type name is T1:

```
add_type -name T1 -type STD_LOGIC -size 4
```

In this case, the LSB of port is 0 and MSB will be calculated as

$4 \times 4 - 1 = 15$

add_wrapper

Adds wrapper ports for instances

Usage

```
add_wrapper{-instance <comma-separated-list-of-instance-names> |  
-component <master-name> }[ -cmd <tcl-command> ][ -wrapper_module_name  
<module-name> ][ -wrapper_instance_name <instance-name> ]
```

Note:

This command is added for specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

Description

The add_wrapper command adds a wrapper port for instances. GenSys allows creation of wrapper for a particular instance or a list of instances or for all the instances of a component.

During RTL generation, a wrapper module will be created and dumped for the specified instance(s)/component. While dumping wrappers, wrapper ports are not saved. However, connections for those ports are saved with RTL.

You can stop dumping of wrapper during RTL generation by setting the -wrapper_creation argument of the [set_rtl_option](#) Tcl command to FALSE.

Wrappers do not have any parameters. For parameterized ports, wrapper instances can have hard-coded values of the parameters. Therefore, there will be different wrappers for each instance. However, if two or more instances have same parameter values, a single wrapper module can be created for all of them. In addition, if different wrapper module names are specified for different instances of a component, different wrapper modules get generated.

Arguments

The add_wrapper command has the following arguments:

`-instance <comma-separated-list-of-instance-names>`

Specifies a comma-separated list of instances for which a wrapper will be created. For more than one instance, that have same wrapper requirements, you can specify a list of instance names. However, if different instances have different wrapper requirements, you need to use the add_wrapper command separately for each instance separately.

`-component <component-name>`

Specifies the name of component instance if a wrapper is to be created for all the instances of a component. This argument is used when all the instances have same wrapper requirements.

Note:

It is mandatory to specify any one of the `-instance` or `-component` arguments.

`-cmd <tcl-command>`

(Optional) Specifies a Tcl command that affects the component. Currently, support has been given only for the `add_port` Tcl command.

`-wrapper_module_name <wrapper-module-name>`

(Optional) Specifies a name for the wrapper module.

Note:

It is recommended to use the `-wrapper_module_name` argument only when the wrapper contains only one instance. GenSys flags an error if this argument is used with a list of instances in the wrapper.

`-wrapper_instance_name <wrapper-instance-name>`

(Optional) Specifies a name to be used for the instance inside the wrapper.

Note:

It is recommended to use the `-wrapper_instance_name` argument only when the wrapper contains only one instance.

Example

The following example shows how ports are created on wrapper modules:

```
new component comp
add_port -name R1 -direction IN -lsb 0 -msb 2
add_port -name R2 -direction IN -lsb 0 -msb 2
add_port -name R3 -direction IN -lsb 0 -msb 2
save comp
new design top
add_port -name P1 -direction IN -lsb 0 -msb 8
add_port -name P2 -direction IN -lsb 0 -msb 3
add_port -name P3 -direction IN -lsb 0 -msb 3
add_instance -name I1 -master comp
add_instance -name I2 -master comp

add_connection -instance I1 -pin R1 -port P1 -lsb 6 -msb 8 -deltadelay 2
add_connection -instance I1 -pin R2 -port P1 -lsb 3 -msb 5
add_connection -instance I2 -pin R3 -port P1 -lsb 0 -msb 2 -deltadelay 5
add_wrapper -instance I1,I2 -cmd { add_port -name R4 -lsb 0 -msb 4
-direction IN}
```

```
add_connection -instance I1 -pin R4 -port P2
add_connection -instance I2 -pin R4 -port P3
```

apply_rule

Applies interface matching rules to infer interface(s) on the designs/components being imported

Usage

```
apply_rule
{-component <design/component-name>}*
{-interface <interface-lib-name>
 [ -vendor <vendor> ]
 [ -library <library> ]
 [ -version <version> } ]*
{-rule <rule-name>}*
```

Description

The `apply_rule` command applies already created interface matching rules to infer interfaces on the specified design/component being imported through RTL import.

Arguments

The `apply_rule` command has the following arguments:

`-component <design/component-name>`

Specifies the name of the design/component being imported on which interfaces need to be inferred.

You can specify this argument multiple times to specify more than one design/component.

`-interface <interface-lib-name>`

Specifies the name of the interface whose logical port names need to be matched.

You can specify this argument multiple times to specify more than one interface.

`-vendor <vendor>`

(Optional) Specifies the vendor of the interface library.

`version <version>`

(Optional) Specifies the version of the interface library.

`-library <library>`

(Optional) Specifies the library of the interface library.

`-rule <rule-name>`

Specifies the rule created by using the [make_rule](#) Tcl command.

You can specify this argument multiple times to specify more than one rule.

Note:

Only those rules that are specified by the `-rule` argument are applied to the specified interfaces.

Example

The following command applies interface matching rule, R1, on interfaces, intf1 and intf2, which need to be inferred on component, comp:

```
apply_rule -component comp
-interface intf2 -interface intf1 -rule R1
```

check_rtl

Compares the specified RTL with a component, design, and subsystem in the library

Usage

```
check_rtl <RTL>
[ -vendor <vendor-name> ]
[ -library <library-name> ]
[ -version <version-num> ]
```

Description

The `check_rtl` command compares the specified RTL with a component, design, or subsystem in the library having the same VNLV and generates a report containing differences between ports and parameters.

Arguments

The `check_rtl` command has the following arguments:

`<RTL>`

Specifies the RTL to be imported.

`-vendor <vendor-name>`

(Optional) Specifies the vendor of the RTL being imported.

`-library <library-name>`

(Optional) Specifies the library of the RTL being imported.

`-version <version-num>`

(Optional) Specifies the version of the RTL being imported.

clearlog

Clears the Tcl log window.

Usage

`clearlog`

Description

The `clearlog` command clears the log information in the Tcl log window in GenSys.

clone_ip

Clones an instance

Usage

```
Usage 1:
clone_ip
-name <instance-name>
-to {<hierarchies-containing-clones>}
-remove_floating_ports <TRUE | FALSE>
Usage 2:
clone_ip
-file <file-with-clone_ip-commands> -ui )
```

Description

The clone_ip command creates a clone of the specified instance in the specified design hierarchy.

Arguments

The clone_ip command has the following arguments:

-name <instance-name>

Specifies the name of the instance to be cloned

-to {<hierarchies-containing-clones>}

Specifies a space-separated list of hierarchies in which the instance clone should be placed.

While specifying hierarchies, use the hierarchy separator specified by the [set_hierarchy_separator](#) command.

If you do not specify this argument, you must specify design masters by using the [set_physical_units](#) command. On specifying this command, the -to argument considers the instance hierarchies of the specified design masters.

-remove_floating_ports <TRUE | FALSE>

Removes hanging or redundant ports that are generated because of cloning.

Set this argument to FALSE to retain such ports.

`-file <file-with-clone_ip-commands>`

Specifies a file containing the clone_ip commands.

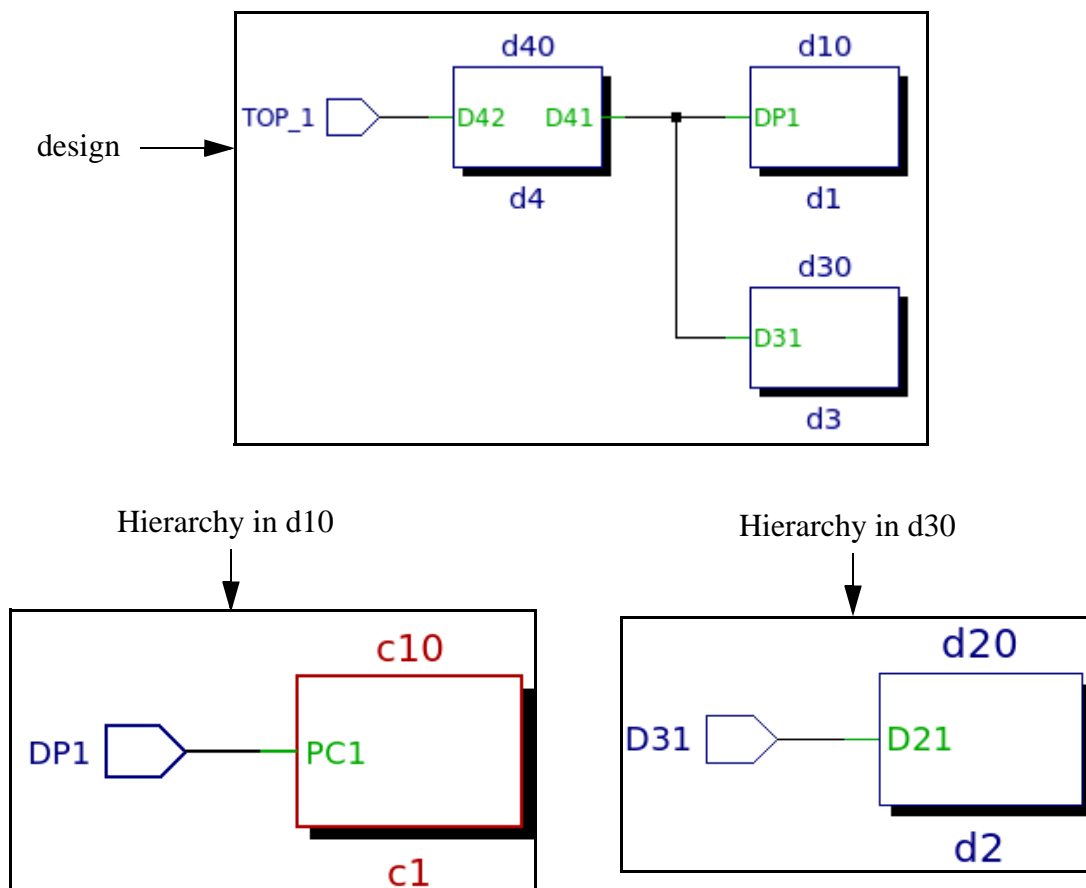
`-ui`

Opens the *Hierarchy Manipulation* widget showing the hierarchy changes resulted by running the file specified by `-file <file-with-clone_ip-commands>`.

Review the changes in the widget and then apply them.

Examples

Consider the following design:



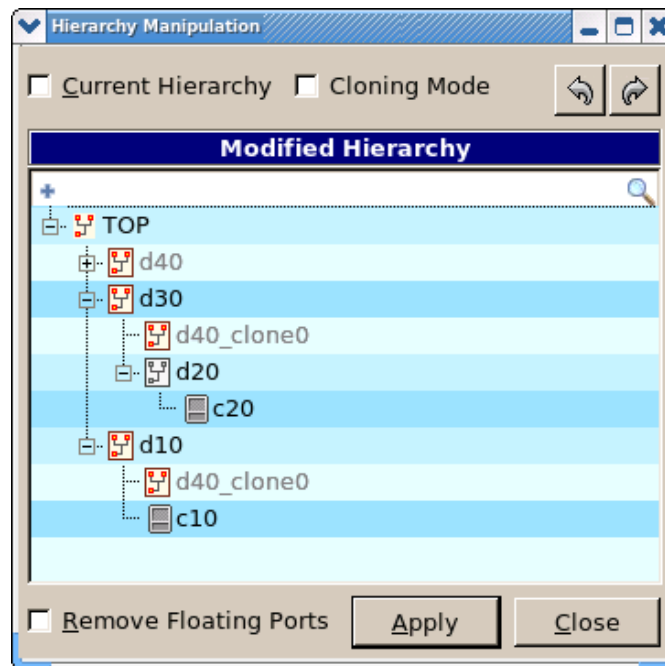
Now consider that you specify the following Tcl commands:

```
set_library_path db
load "db/TOP.xml"
clone_ip -file cmd.tcl -ui
```

Where cmd.tcl contains the following command:

```
clone_ip -name TOP::d40 -to {TOP::d10 TOP::d30}
```

The following Hierarchy Manipulation widget appears:



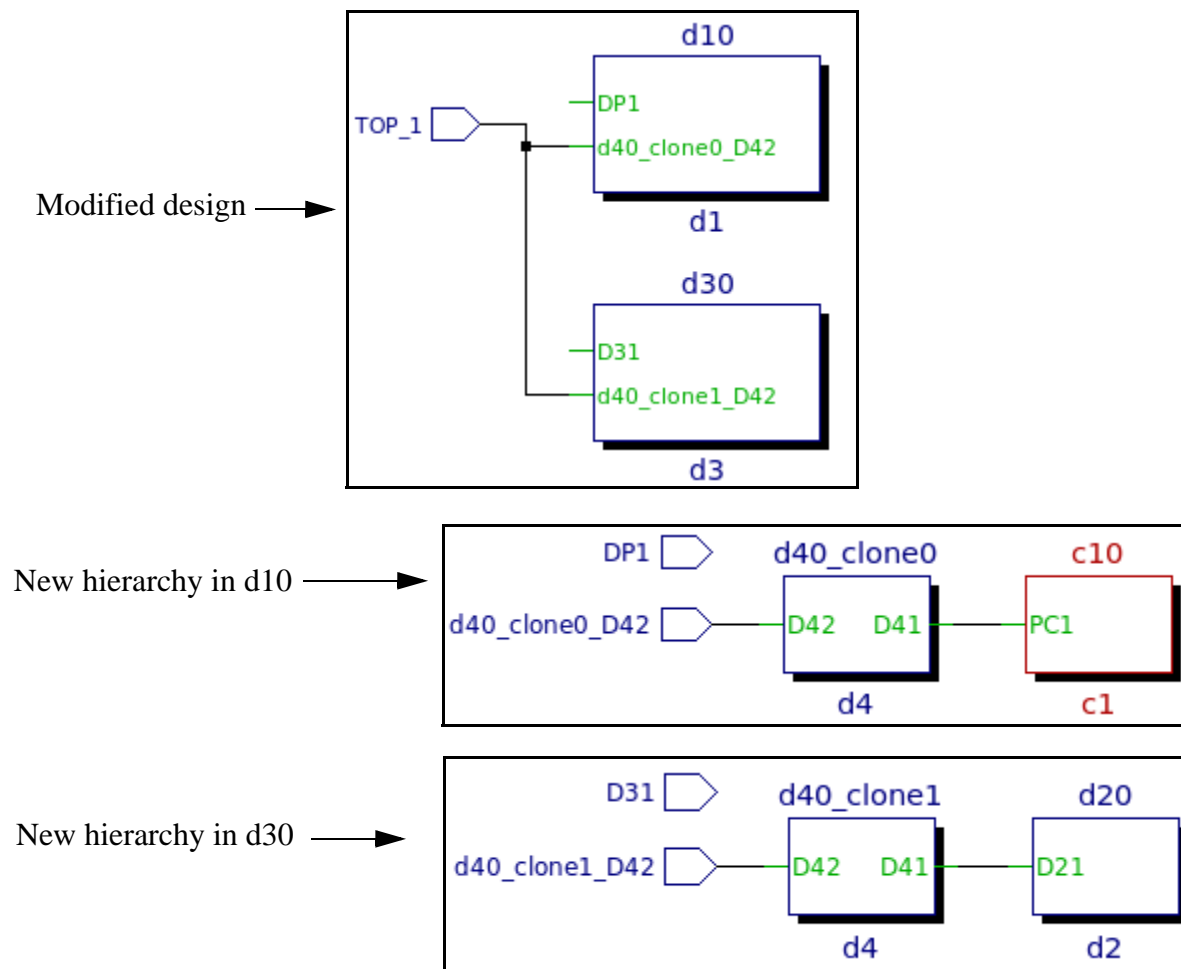
Review the hierarchy changes in the above widget and then click the Apply button.

Note:

In the above dialog, you cannot undo the changes done by the clone_ip command. However, you can undo and redo the changes made over these changes.

On clicking the Apply button, the clones are created. Following is the modified schematic of the design:

The design changes to the following:



close_object

Closes a design, interface, component, or design configuration

Usage

```
close_object
[ -save <object-name> ]
[ <design-name>
  | <interface-name>
```

```
| <component-name> ]  
| <designConfiguration-name> ]
```

Description

The `close_object` command closes the specified object (design, interface, component, design configuration).

If you do not specify the name of the object to be closed, GenSys closes the current design, interface, component, or design configuration.

Arguments

The `close_object` command has the following arguments:

`-save <object-name>`

(Optional) Saves the object (design, interface, component, or design configuration) being closed. If you do not specify the name of the object with this argument, GenSys saves the current object.

Examples

The following example closes the current design, interface, component, or design configuration:

```
close_object
```

closeall

Closes all current designs, components, and interface objects.

Usage

```
closeall
```

Description

The `closeall` command closes all current designs, interfaces, and components.

The `closeall` command removes all current objects from the memory. When you use this command, all the internal pointers earlier returned by Perl APIs/OO methods or Tcl APIs become inaccessible. Hence, you should not use such (stored) pointers after calling this command.

You should use the `closeall` command when you have finished working in a design, interface, or component.

Examples

The following example closes all current designs, interfaces, and components:

```
closeall
```

create_report

Creates the specified report

Usage

Usage 1 (For generating the Design Statistics report)

```
create_report -category statistics
  -report designstats
  [ -file <report-file-name> ]
  [ -hierarchy <TRUE | FALSE> ]
```

Usage 2 (For generating the Interface Summary report)

```
create_report -category statistics
  -report interfaces
  [ -file <report-file-name> ]
  [ -hierarchy <TRUE | FALSE> ]
  [ -detail <TRUE | FALSE> ]
```

Usage 3 (For generating the Connectivity report)

```
create_report -category statistics
  -report connectivity
  [ -file <report-file-name> ]
  [ -design <design-name> ]
  [ -detail <TRUE | FALSE> ]
  [ -instance <inst-name> ]
  [ -scalar <TRUE | FALSE> ]
```

Usage 4 (For generating the Suggested Interfaces report)

```
create_report -category statistics
```



```
-report suggested_interfaces
[ -file <report-file-name> ]
[ -hierarchy <TRUE | FALSE> ]
```

Usage 5 (For generating the Design Differences report)

```
create_report -category differences
-gold {-name <gold_component/design_name>
-vendor <vendor_name>
-version <version_number>
-library <library_name> }
-report <ports | parameters | attributes | filesets |
interfaces | instances | connectivity | all>
[ -file <report-file-name> ]
[ -hierarchy {TRUE|FALSE} ]
[-current {-name <current_component/design_name>
-vendor <vendor_name>
-version <version_number>
-library <library_name> } ]
```

Usage 6 (For generating the RTL Health Check report)

```
create_report -category debug
[ -file <report-file-name> ]
[ -hierarchy <TRUE | FALSE> ]
[ -design <design-name> ]
[ -flatten {TRUE | FALSE | <Integer-value>} ]
```

Usage 7 (For generating consolidated report for all macros of imported RTL)

```
create_report -category statistics
-report macros
[ -file <report-file-name> ]
```

Description

The `create_report` command creates the following reports: Design Statistics, Interface Summary, Connectivity, Suggested Interfaces, Design Differences, Macros, or the RTL Health Check report.

The report is generated in the directory specified by the `set_report_dir` Tcl command.

Arguments

For [Usage 5 \(For generating the Design Differences report\)](#), the `create_report` command arguments are documented in the [Design Differences Report Arguments](#) section.

For the following usages, the `create_report` command has these arguments.

- [Usage 1 \(For generating the Design Statistics report\)](#)

- Usage 2 (For generating the Interface Summary report)
- Usage 3 (For generating the Connectivity report)
- Usage 4 (For generating the Suggested Interfaces report)
- Usage 6 (For generating the RTL Health Check report)
- Usage 7 (For generating consolidated report for all macros of imported RTL)

`-report <designstats|interfaces|connectivity|suggested_interfaces|macros>`

Specifies the type of the report to be generated, as mentioned in the following table:

Option	Report Generated
designstats	Design statistics report
interfaces	Interface summary report
connectivity	Connectivity report
suggested_interfaces	<p>Suggested interfaces report</p> <p>This report displays details of possible interfaces that can be inferred on a design based on the existing design connections.</p> <p>You can generate this report prior to inferring interfaces on a design by using the <code>infer_interfaces</code> Tcl command.</p>
macros	<p>Macros report</p> <p>This is a consolidated report for all macros of an imported RTL design.</p>

Note:

To view the suggested interfaces report, perform the following actions:

Specify the `infer_interfaces -connections true -suggested` Tcl command.

Specify the `create_report -category statistics -report suggested_interfaces` Tcl command.

On performing the above actions, GenSys creates the `suggested_interfaces.txt` report under the Report directory. In addition, interfaces suggested are visible in the Interface Mapping Widget dialog. You can use `-file` argument of the `create_report` command to specify a different name of the report.

`-file <report-file-name>`

(Optional) Specifies the name of the report to be generated.

If you do not specify this argument, a default name is assigned to the generated report. The following table shows the default name generated for different type of reports:

Report Type	Default Name Generated
Design summary report	<active-object-name>_summary.txt
Interface summary report	<active-object-name>_interface_summary.txt
Connectivity report	For an active design: <design-object>_connectivity_report.txt For a sub-design with an instance name:<design-object>_<instname>_connectivity_report.txt
Suggested Interfaces report	suggested_interfaces.txt
RTL Health Check report	rtl_health_check.csvIf there are multiple files being generated, as is the case of the Hierarchical RTL Health Check report, the design name is appended to the specified file name. The default file name used is:rtl_health_check_<design name>.csv.
Macros report	macros.rpt

`-hierarchy <TRUE | FALSE>`

(Optional) Specifies if the report should also include details of subsystems instantiated within the hierarchy of the currently selected design/component.

In case of generating the suggested interfaces report, if you set this argument to TRUE, the report contains information of suggested interfaces for the top-level design as well as for subsystems in the design hierarchy. However, if you set this argument to FALSE, the suggested interfaces report contains information of suggested interfaces only for the top-level design.

By default, this argument is set to FALSE.

`-detail <TRUE | FALSE>`

(Optional) For interface summary report, this argument specifies if additional details, such as logical to actual port-mapping should be generated in the report.

For connectivity report, this argument specifies if additional connection details of an instance (specified by the `-instance <inst-name>` argument) within the currently active design or a subsystem (specified by the `-design <design-name>` argument) should be generated in the report.

Note:

While generating the connectivity report, if the `-detail` argument is set to TRUE, you must specify an instance by using the `-instance <inst-name>` argument.

By default, this argument is set to FALSE.

`-design <design-name>`

(Optional) Specifies the name of a design for which the connectivity report should be generated.

The design can be a top-level design or a subsystem-level design.

For RTL Health Check Report

If a component instance name is given, an ERROR is reported because the RTL Health Check report cannot be generated for components. If the `-hierarchy <TRUE | FALSE>` argument is set to TRUE, all internal sub-system's RTLHC reports are generated starting from the given sub-system. If the `-hierarchy <TRUE | FALSE>` is not specified, the top-level design is the default.

`-instance <inst-name>`

(Optional) Specifies the name of an instance (instantiated in currently active design or the subsystem specified by the `-design <design-name>` argument) for which additional connection details should be generated in the connectivity report.

Note:

While generating the connectivity report, specify this argument only if the `-detail <TRUE | FALSE>` argument is set to TRUE.

`-scalar <TRUE | FALSE>`

(Optional) Specifies if scalar ports information should be generated in the connectivity report.

`-flatten { TRUE | FALSE | <Integer-value> }`

(Optional) Specifies whether to generate a flattened RTL Health Check report.

Default value is FALSE. If set to TRUE, the top-level design is flattened up to the level specified and the report is generated.

Depending on the design and the levels, generating a flattened RTL Health Check report can be time-consuming.

Note:

The `-hierarchy <TRUE | FALSE>` and `-flatten { TRUE | FALSE | <Integer-value> }` arguments cannot be specified together.

Design Differences Report Arguments

The `create_report` command has the following arguments for generating the Design Differences report:

```
-gold {-name <gold_component/design_name> [-vendor <vendor_name>]
[-version <version_number>] [-library <library_name>]}
```

Specifies the reference design/component against which the current design/component is compared to generate the difference report.

The `-name` argument is mandatory, while the `-vendor`, `-version`, and `-library` arguments are optional.

```
-report <ports | parameters | attributes | filesets | interfaces |
instances | connectivity | all>
```

(Optional) Specifies the item, such as ports or parameters for which the report should be generated.

The default value of the `-report` argument is `all`. Therefore, if you do not specify this argument, the design difference report is generated for all the item types.

```
-file <report-file-name>
```

(Optional) Specifies the name of the report to be generated.

If you do not specify this argument, the name assigned to the generated report is: `<Current component/design_name>_difference.txt`

```
-hierarchy <TRUE | FALSE>
```

(Optional) Specifies if the report should also include details of subsystems instantiated within the hierarchy of the currently selected design/component.

By default, this argument is set to `FALSE`.

```
-current {-name <current_component/design_name> -vendor <vendor_name>
-version <version_number> -library <library_name>}
```

(Optional) Specifies the current design/component against which the reference design/component is compared to generate the difference report. By default, the currently selected component/design in the design browser is taken as current.

For specifying the `-current` argument, the `-name` argument is mandatory, while the `-vendor`, `-version` and `-library` arguments are optional.

Examples

Example 1 - Generating a Design Summary Report

The following command generates a design summary report for the currently active design:

```
create_report -category statistics -report designstats  
-file dsgn1_design_summary.txt -hierarchy TRUE
```

In the above example, the generated report also includes design summary of subsystems instantiated in the currently active design.

Example 2 - Generating an Interface Summary Report

The following command generates an interface summary report for the currently active design:

```
create_report -category statistics -report interfaces -file  
mpy_top_interface_summary.txt -detail TRUE -hierarchy FALSE
```

In the above example, the generated report would contain additional details, such as logical to actual port-mapping details in the currently active design. However, the report will not contain interface summary of subsystems within the hierarchy of the currently active design.

Example 3 - Generating a Connectivity Report

The following command generates a connectivity report for the subsystem check instantiated in the currently active design:

```
create_report -category statistics -report connectivity-design check  
-instance comp10 -detail TRUE -filecheck0_comp10_connectivity_report.txt
```

In the above example, the generated report also contains additional connection details of the comp10 instance instantiated within the check subsystem instance.

Example 4 - Generating a Design Differences Report

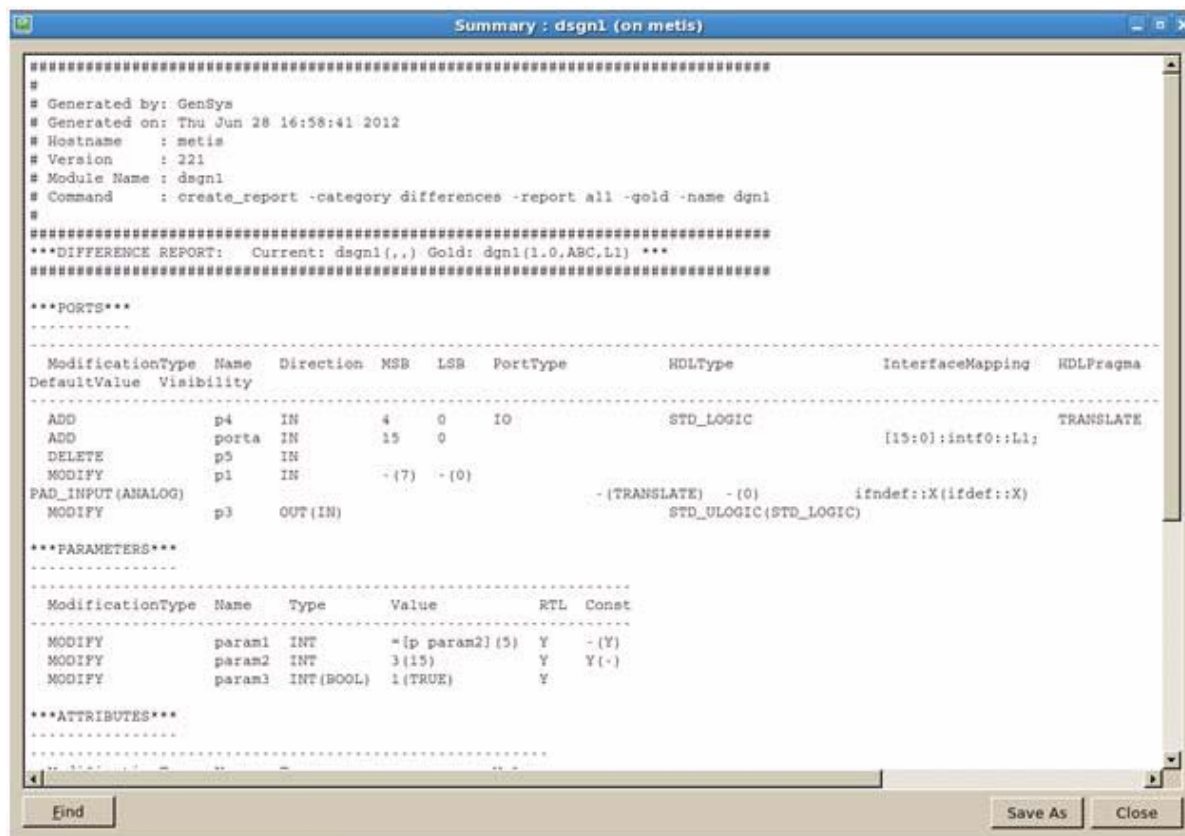
The following command generates a design difference report for the currently active component with respect to reference component comp1:

```
create_report -category differences -report all -gold -name comp1
```

To specify a current component/design instead of taking the default, that is, currently active (selected) one, you can also give the command as:

```
create_report -category differences -report all
-gold -name comp1 -current -name comp2.
```

This command generates the design difference report as shown in the following figure:



Since `-report all` is specified in the Tcl command, this report shows differences based on all the object types applicable for components, that is ports, parameters, attributes, interfaces and filesets.

For designs, differences in ports, parameters, attributes, interfaces, instances and connectivity would be reported.

Example 5 - Consolidated Report for All Macros of Imported RTL

The `create_report` command generates a consolidated report for all the macros of the imported RTL design. The report includes the macro names, macro values, macro definition file names, and line numbers in tabular format.

Before using the `create_report` command, you must set the `set_rtlimport_option -enable_macro_genreport` to true:

```
set_rtlimport_option -enable_macro_genreport true
```

To generate the report after importing the RTL, use the `create_report` command as follows:

```
create_report -category statistics -report macros [ -file
<report-file-name>]
```

This command generates the report file, *report-file-name*. If you do not specify the file name, the default file name is `macros.rpt`.

The following example shows an RTL module with macros, SIZE1 AND SIZE2, and their report.

Top.sv

```
`define SIZE1 2
`define SIZE2 (`SIZE1+6)
module top(a, b);
input [`SIZE1:0] a;
input [`SIZE2:0] b;
dummy dummy();
endmodule
```

macros.rpt

```
-----
MacroName  FileName  LineNumber  Value
-----
SIZE1      top.sv    1           2
SIZE2      top.sv    2           8
-----
```

create_report_ERC

Creates the ERC report

Usage

```
create_report_ERC
-help
-currentoptions
-resetoptions
-noexec
-triggerOn <list>
-triggerOff <list>
[-file <Report filename>]
[ -disconnects <TRUE | FALSE> ]
  [ -input <TRUE | FALSE> ]
[ -output <TRUE | FALSE> ]
[ -opens <TRUE | FALSE> ]
[ -partials <TRUE | FALSE> ]
[ -endians <TRUE | FALSE> ]
[ -splices <TRUE | FALSE> ]
[ -tieoffs <TRUE | FALSE> ]
[ -ports <TRUE | FALSE> ]
[ -verbose <TRUE | FALSE> ]
[ -multidriviers <TRUE|FALSE> ]
```

Description

The create_report_ERC command generates the ERC report. The report recurses down the hierarchy, reporting at each design instance as well as at the top.

For a description of the arguments in blue, refer to [create_report_Packaging](#).

Arguments

The create_report_ERC command has the following arguments:

-disconnects <TRUE | FALSE>

(Optional) Reports disconnects or floating terminals. The default value is TRUE.

-input <TRUE | FALSE>

(Optional) Reports floating inputs. The default value is TRUE.

This option is most useful for simulation, where floating inputs may cause wasted runs. Therefore, if a port is going to be used as an input, it is reported if it is floating. For example, an output port on a design that is disconnected internally. However to avoid confusion, the port direction is per the COM.

-output <TRUE | FALSE>

(Optional) Reports floating outputs. The default value is TRUE.

Similar to the input argument, this option checks for pins that are used as the outputs floating. Therefore, the design input ports left unconnected inside the design are reported only by this argument. However to avoid confusion, the port direction is per COM.

`-opens <TRUE | FALSE>`

(Optional) Reports forced opens or connections that are intentionally left open. The default value is TRUE.

`-partials <TRUE | FALSE>`

(Optional) Reports partial connected. The default value is TRUE.

The number of pins connected is reported. If the port's pins are split, for example, Port A 16 bits, byte 15:8 drives port B, byte 7:0 drives port C, Port A is reported as partially connected. However, after both connections are identified a further message indicates that it is a split connection.

`-endians <TRUE | FALSE>`

(Optional) Reports endian bit reversal. The default value is TRUE.

Fully connected vector connections are analyzed for complete pin re-arrangement by the endian connection.

`-splices <TRUE | FALSE>`

(Optional) Reports fully spliced. The default value is TRUE.

Fully connected vector connections are analyzed for complete pin re-arrangement by splicing. Splices is for 1:1 port only, for bit scrambling other than reverse or normal bit arrangements.

`-tieoffs <TRUE | FALSE>`

(Optional) Reports tie off terminals. The default value is TRUE.

`-ports <TRUE | FALSE>`

(Optional) Reports all ports. The default value is FALSE.

`-verbose <TRUE | FALSE>`

(Optional) Generates a report that contains all connections. The default value is FALSE.

`-multidriviers <TRUE | FALSE>`

(Optional) Reports multidriver in the ERC report. By specifying this argument, only multiple drivers on the design level are reported. It will not recursively iterate the hierarchy. The default value is TRUE.

create_report_Packaging

Creates the Packaging report

Usage

```
create_report_Packaging
-help
-currentoptions
-resetoptions
-noexec
-triggerOn <list>
-triggerOff <list>
[-file <Report filename>]
```

Description

The create_report_Packaging command generates the Packaging report.

Arguments

The create_report_Packaging command has the following arguments:

-help

Lists autogenerated usage.

-currentoptions

Lists current options.

-resetoptions

Specifies the reset value to InitValue.

-noexec

Suppresses execution. For example, for setting options for subsequent triggers.

-triggerOn <list>

For example, 'save' - will execute on each save (Tip: use save -top)

-triggerOff <list>

For example, 'load gload' - will stop execution of this report when open through the Tcl or GUI browser.

```
[-file <Report filename>]
```

(Optional) Specifies the name of the report file.

default_datasource

Enables/disables the addition of datasources to all the COM objects created after the execution of this command

Usage

```
default_datasource
| -reset
[-personname <default_datasource_name>]
[-creationdate <date>]
[-changetype <type>]
[-changereason <reason>]
[-changemethod <method> ]
[-changedate <date>]
[-datafrozen <TRUE|FALSE>]
[-table <table-hier-name>]
```

Description

The default_datasource command enables or disables the addition of datasources to all the COM objects created after the execution of this command.

Note:

The default_datasource Tcl command is valid for the complete GenSys session even after closing all the current designs, interfaces, and components using the [closeall](#) Tcl command.

Arguments

The default_datasource command has the following arguments:

-reset

Resets the global or table level data source additions. When this argument is used, no other arguments are required.

-personname <default-datasource-name>

(Optional) Specifies the default datasource name.

Note:

For backward compatibility, -name is also supported.

-creationdate <creation-date>

(Optional) Specifies the date of creation of the datasource. If you do not specify the creation date then the current system date is considered.

Note:

For backward compatibility, -date is also supported.

-changetype <type>

(Optional) Specifies the type of datasource.

-changereason <reason>

(Optional) Specifies the reason to make changes in the datasource.

-changemethod <method>

(Optional) Specifies the method that made the changes.

-changedate <date>

(Optional) Specifies the date on which the change occurs.

-datafrozen <TRUE | FALSE>

(Optional) Specifies whether the changes made should be frozen.

-name <datasource-person-name>

(Optional) Specifies the default datasource name.

-date <creation-date>

(Optional) Specifies the date of creation of the datasource. If you do not specify the creation date then the current system date is considered.

-table <table-hier-name>

(Optional) Specifies the hierarchical table name for which the default datasource is being set.

If you specify this option, the datasource addition is restricted to the specified table only and not all the COM objects.

default_match_rule

Applies a default rule on all the interfaces if rule-based infrastructure is NOT being used for matching interfaces for designs/components

Usage

```
default_match_rule
[ -effort <high | low> ]
[ -substr <num>:<num> ]*
[ -case_insensitive_match <TRUE | FALSE> ]
```

Description

The default_match_rule command applies a default rule on all the interfaces if no rule-based infrastructure is used. A rule-based infrastructure is not implemented if you do not use the [make_rule/apply_rule](#) Tcl commands to match interfaces to be inferred on designs/components. For such cases, GenSys applies a default rule on all the interfaces by using the default_match_rule Tcl command.

Arguments

The default_match_rule command has the following arguments:

`-effort <high | low>`

(Optional) Specifies the effort required to match the logical port names with the actual port names on designs/components.

By default, the value of this argument is set to high, which means that a definite matching pattern is followed while matching the logical ports with the actual ports. In this case, the prefix and postfix pattern of all the logical port names should be exactly the same.

For example, consider an interface which has logical ports, L1 and L2, and consider a component which has actual ports, A1_L1_P1 and A1_L2_P1. Now, if you set the value of the -effort argument to high, GenSys checks the prefix and postfix pattern of the logical ports, L1 and L2, appearing in the actual port names, A1_L1_P1 and A1_L2_P1. In this case, the prefix/postfix pattern in both the ports are same, i.e., A1:P1. Hence, the logical ports are mapped with their corresponding actual ports in this case.

Now consider another case in which the actual port names are A1_L1_P1 and A1_L2_P2. In this case, the prefix/postfix pattern for L1 is A1:P1 and for L2 is A1:P2. Since both these patterns are different, the logical ports are not mapped with their corresponding actual ports in this case.

You can set the value of this argument to low if you do not want to follow the exact matching pattern. When you set the value of this argument to low, GenSys only checks whether the logical port name is a substring of the actual port name. If the logical port name is a substring of the actual port name, GenSys maps those logical ports with the actual ports.

```
-substr <num>:<num>
```

(Optional) Specifies the subset of input string, which is used for interface matching.

A substring is created from the logical port name. For example, consider a logical port name, InterfacePort1. If you want to create a substring, IPort1, from this logical port name, you need to specify the following:

```
-substr 1:1-substr 10:14
```

```
-case_insensitive_match <TRUE | FALSE>
```

Specifies whether the matching pattern should be case sensitive. By default, this argument is set to FALSE, and the interface matching pattern is case sensitive.

del_library_obj

Deletes the specified library object.

Usage

```
del_library_obj
  -type <design | component | interface>
  -name <name> | -vendor <vendor-name>
  | -version <version-num>
  | -library <library-name>
  [ -file <file-name> ]
```

Description

The del_library_obj command deletes the specified library object.

Arguments

The del_library_obj command has the following arguments:

`-type <design | component | interface>`

Specifies the type of library object that needs to be deleted. The `-type` argument can take the following values:

`design | component | interface`

`-name <name>`

Specifies name of the library object to be deleted.

`-vendor <vendor-name>`

Specifies name of the vendor of the object to be deleted.

`-version <version-num>`

Specifies version number of the object to be deleted.

`-library <library-name>`

Specifies name of the library of the object to be deleted.

`-file <file-name>`

(Optional) Specifies name of the file that contains the object that is to be deleted. If the file exists in the library, it would be deleted.

delete_autoconnect_rule

Deletes the rule in the autoconnect table of design

Usage

```
delete_autoconnect_rule
[ -name <rules-list> ]
[ -all ]
```

Description

The `delete_autoconnect_rule` command deletes that autoconnect rule from the Autoconnect table whose rule name matches the rule name specified in this command.

Deleting Connections Created by an Autoconnect rule

The `delete_autoconnect_rule` command does not delete connections generated by the rule being deleted. To delete such connections, use the [delete_autoconnections](#) Tcl command before deleting the corresponding rule by using the `delete_autoconnect_rule` command.

For example, consider the following Tcl commands:

```
# rule1 autoconnect rule added in Autoconnect table
add_autoconnect_rule -name rule1 -src_pin p1.* -dst_pin p2.*

#Generating connections specified by the rule1 rule
run_autoconnect
```

Now, to delete connections generated by the `rule1` rule of the above example, run the following command:

```
delete_autoconnections -name rule1
```

However, if you first delete the `rule1` rule by using the `delete_autoconnect_rule` Tcl command and then try to delete connections of this rule by using the `delete_autoconnections` Tcl command, GenSys does not delete connections of the `rule1` rule.

Arguments

The `delete_autoconnect_rule` command has the following argument:

`-name <rule-list>`

(Optional) Specifies a space-separated list of rules that you want to delete from the Autoconnect table.

The following example shows the usage of this argument:

```
delete_autoconnect_rule -name { rule1 rule2 }
```

`-all`

(Optional) Used to delete all Autoconnect rules in the Autoconnect table.

delete_autoconnections

Deletes autoconnect connections generated by the execution of autoconnect rules

Usage

```
delete_autoconnections
  -name <rule-name-list> | -all
  -hierarchy <yes | no>
```

Description

The `delete_autoconnections` command deletes automatic connections generated in any of the following ways:

- By using the [delete_autoconnect_rule](#) Tcl command
- By specifying connection details in the Autoconnect table in the GUI

Arguments

The `delete_autoconnections` command has the following arguments:

`-name {rule-names-list}`

Specifies a space-separated list of rules that generated automatic connections. In this case, automatic connections generated by the specified rules are removed.

You can use regular expressions while specifying rule names.

`-all`

Specifies that automatic connections generated by all rules should be removed.

`-hierarchy <yes | no>`

Specifies if connections within the design hierarchy should also be deleted along with the connections present at the top-level design.

Use this argument only if you are running this command in a new GenSys session and you have generated automatic connections in a previous session.

For example, consider that you create a rule by running the following command:

```
add_autoconnect_rule -name rule1 -src_pin P1
-dst_inst inst1::inst2::inst3 -dst_pin P2
```

After running the above rule, when you elaborate your design, you see multiple connections generated by that rule. These connections include connections at the top-level design and connections within the hierarchy of that design (that is, connections at sub-design level).

Now, if you run the `delete_autoconnections -name rule1` command in the same session, all connections at the top-level design and within the hierarchy of that design are deleted.

However, if you save your design and close GenSys session after running the `add_autoconnect_rule` command and run `delete_autoconnections -name rule1` command in the next session after reloading the same design, GenSys only deletes connections at the top-level. In this case, if you want to delete connections within the hierarchy of that design, use the following command:

```
delete_autoconnections -name rule1 -hierarchy yes
```

delete_by_datasource

Deletes all components which have datasource matching the specified string.

Usage

```
delete_by_datasource <string>
```

Description

The `delete_by_datasource` command deletes all components that have datasource name matching the specified string *<string>*. It also deletes all connections made to those components.

The command is particularly useful to remove components and connections created by a previous run of a generator.

delete_component

Deletes the specified component.

Usage

```
delete_component
  -name <name>
  [ -vendor <vendor-name> ]
  [ -version <version-num> ]
```

```
[ -library <library-name> ]
```

Description

The `delete_component` command deletes the specified component *<name>* with the given *<vendor-name>*, *<version-num>*, and *<library-name>* from the root. This command also deletes all the instances of the specified component along with the connections specified for those instances.

Arguments

The `delete_component` command has the following arguments:

`-name <name>`

Specifies the name of the component *<name>* that needs to be deleted.

Note:

If you specify only the name of the component to be deleted using the `-name` argument, and many components exist with the same name (but with different vendor, version, and library information) then the component to be deleted is based on the order in which the components are populated in the root. In such a case, the first component that exists in the root is deleted.

Note:

Always specify the vendor, library, and version along with the name of the component to avoid any unexpected deletion of components.

`-vendor <vendor-name>`

(Optional) Specifies the name of the vendor of the component.

`-version <version-num>`

(Optional) Specifies the version number of the component that needs to be deleted.

`-library <library-name>`

(Optional) Specifies the library name of the component that needs to be deleted.

Examples

The following example deletes the component named `comp5`:

```
delete_component -name comp5
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case3 directory.

delete_connection

Deletes a connection.

Usage

```
delete_connection
  <from-options>
  <to-options>
  -debug <Y | N>
<from-options>::=
  -instance <inst-name> -pin <pin-name>
  [ -lsb <lsb> -msb <msb> ]
  | -instance <inst-name>
  | -instance <inst-name>
    -interface <interface-name>)
  | -port <port-name> [ -lsb <lsb> -msb <msb> ]
  | -interface <interface-name>
  [ -netname <net-name> ]
  [ -hierarchynetname <Y | N> ]
<to-options>::=
  -instance <inst-name> -pin <pin-name>
  [ -lsb <lsb> -msb <msb> ]
  | -instance <inst-name>
  | -instance <inst-name>
    -interface <interface-name>)
  | -port <port-name> [ -lsb <lsb> -msb <msb> ]
  | -interface <interface-name>
```

Description

The delete_connection command deletes connections between specified FROM object (instance pins, instance logical pins, instance' interface, design ports, design logical ports, or design's interface) and the specified TO object (instance pins, instance logical pins, instance' interface, design ports, design logical ports or design's interface). This command deletes the connection from the design.

Note:

Connections formed as a result of using the export_interface command, should be deleted using the delete_export_interface command.

Note:

The `delete_connection` command supports partial deletion of a connection.

The connection types are as follows:

Connection Types	FROM Object	TO Object
Adhoc-to-Adhoc	Instance pins, design ports	Instance pins, design ports
Interface	Instance's interface, design's interface	Instance's interface, design's interface

You can specify the `delete_connection` command in the following manner:

- Specify only the *<from-options>*
- Specify only the *<to-options>*
- Specify the complete connection (both *<from-options>* and *<to-options>*)

Arguments

The `delete_connection` command has the following arguments:

`-instance <inst-name>`

This option has been deprecated for the deletion of a single instance.

Specifies the name of the instance for which the connection is being deleted.

The `-instance` argument is not applicable for connections to design ports and design logical ports.

The following specification will delete all the connections having instance I1:

```
delete_connection -instance I1
```

It is not essential to specify the pin or the port name of the instance when you are deleting all the connections of a specified instance *<inst-name>*.

When deleting connection between vector port/pins, it is recommended to specify both ends of the connection to uniquely delete the connection.

`-pin <pin-name>`

Specifies the name of the instance pin of instance *<inst-name>*. The following specification will delete all the connections associated with the pin P1 of the instance I1:

```
delete_connection -instance I1 -pin P1
```

Note:

When deleting connection between vector port/pins, it is recommended to specify both ends of the connection to uniquely delete the connection.

```
-port <port-name>
```

This option has been deprecated for the deletion of a single port.

Specifies the name of the design port *<port-name>*.

When deleting connection between vector port/pins, it is recommended to specify both ends of the connection to uniquely delete the connection.

The -instance argument is not applicable for connections to design ports.

The following specification will delete all the connections having port P1:

```
delete_connection -port P1
```

When deleting connection between vector port/pins, it is recommended to specify both ends of the connection to uniquely delete the connection.

```
-interface
```

This option has been deprecated for the deletion of a single interface.

Specifies name of the instance or design interface.

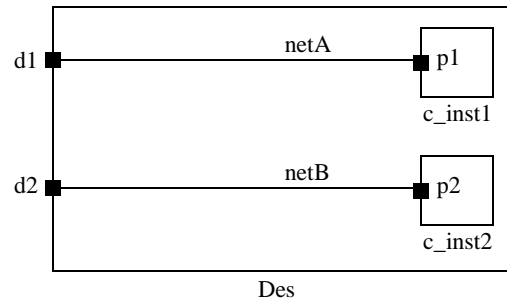
```
-netname <net-name>
```

(Optional) Specifies the name of a net so that GenSys deletes all connections created by that net. This is explained in the following figure:

Scenario 1

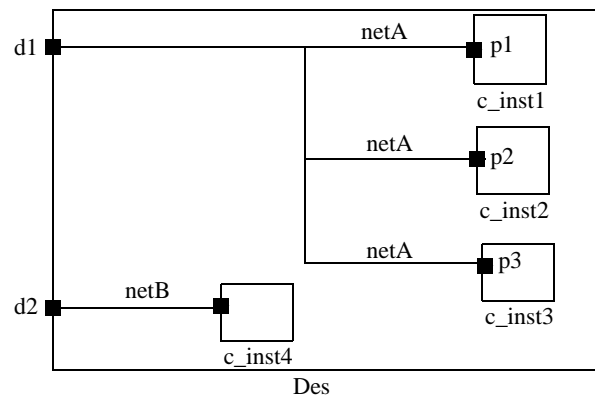
```
delete_connection -netname netA
```

(Deletes the connection between the **d1** port and the **p1** pin)

**Scenario 2**

```
delete_connection -netname netA
```

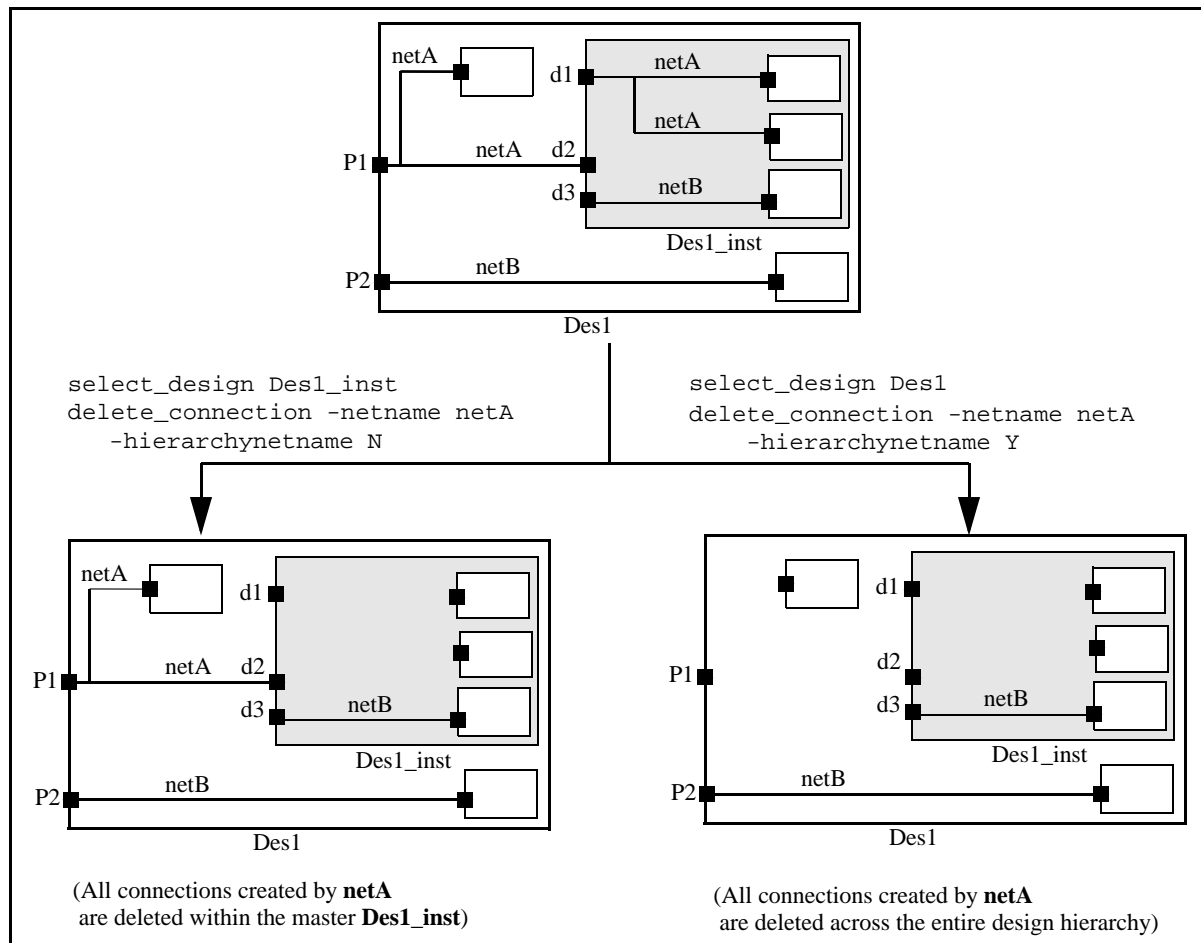
(Deletes the connections between the **d1** port and the **p1**, **p2**, and **p3** pins)



Also see [Example 7](#).

```
-hierarchynetname <Y | N>
```

(Optional) If you set this argument to Y, GenSys deletes all connections containing the net specified by the `-netname <net-name>` argument across the entire design hierarchy. This is explained in the following figure:



-debug <Y | N>

Specifies if the extra debug information needs to be dumped.

Examples

Some of the examples of using the delete_connection command are as follows:

Example 1

Delete Adhoc-to-Adhoc connection between two instance pins:

```
delete_connection
-instance inst1 -pin pin1
```

```
-instance inst2 -pin pin2
```

The above specification deletes an Adhoc connection between pin pin1 of instance inst1 and pin pin2 of instance inst2.

Example 2

Delete Adhoc-to-Adhoc connection between an instance pin and a design port:

```
delete_connection -instance inst1 -pin pin1 -port p1
```

The above specification deletes an Adhoc connection between pin P3 of instance COMPONENT2 and design port P2.

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case2 directory.

Example 3

Delete Logical connection between two instance pins:

```
delete_connection
  -instance inst1 -logical_pin pin1
  -instance inst2 -logical_pin pin2
```

The above specification deletes the logical connection between pin pin1 of instance inst1 and pin pin2 of instance inst2.

Example 4

Delete Interface connection between two instance interfaces:

```
delete_connection
  -instance inst1 -interface i1
  -instance inst2 -interface i2
```

The above specification deletes an interface connection between interface i1 of instance inst1 and interface i2 of instance inst2.

Example 5

Delete Interface connection between an instance interface and a design interface:

```
delete_connection
```

```
-instance inst1 -interface i1
-interface di2
```

The above specification deletes an interface connection between interface i1 of instance inst1 and the design interface di2.

Example 6

Delete partial connection between two pins:

A connection is created between two vector ports R1 on I1 and R2 on I2, using the command given below:

```
add_connection -instance I1 -pin R1 -instance I2 -pin R2
```

You can delete connectivity between only a single bit slice by using the following `delete_connection` command:

```
delete_connection -instance I1 -pin R1 -lsb 3 -msb 4
-delete_connection -instance I2 -pin R2 -lsb 3 -msb 4
```

Example 7

Delete connections through the specified net name:

Consider the following example:

```
new component c
add_port -name c1 -direction IN -lsb 0 -msb 1

new design d
add_port -name d1 -direction IN -lsb 0 -msb 1
add_instance -name cInst -master c
add_connection -instance cInst -pin c1 -port d1
               -name firstConn
delete_connection -netname firstConn
run GenerateVerilog
```

When you run the above commands, GenSys generates the following RTL:

```
module d (d1);
input    [1:0]    d1;
wire     [1:0]    UNINTENTIONAL_OPEN
           c cInst (
                       .c1(UNINTENTIONAL_OPEN)    //I:2
                   );
endmodule
```

In the above RTL, notice that the c1 pin is unconnected (as indicated by UNINTENTIONAL_OPEN). This is because the firstConn net connecting the c1 pin and the

d1 port is deleted by specifying this net in the -netname argument of the delete_connection command.

delete_export_interface

Deletes an export interface.

Usage

```
delete_export_interface
  -interface <interface-inst-name>
  -instance <inst-name>
  -name <exported-interface-name>
```

Description

The delete_export_interface command deletes the specified exported interface (created using the [export_interface](#) command).

Arguments

The delete_export_interface command has the following arguments:

-interface <interface-inst-name>

Specifies the name of the interface instance whose exported interface is to be deleted.

Note:

An Interface library is the library of the top-level interface.

-instance <inst-name>

Specifies the name of the parent instance out of which the specified interface was exported.

-name <actual-name>

Specifies the name by which the interface was exported.

Examples

The following example deletes the exported interface exportIntf1 of interface Intf1:

```
delete_export_interface
  -interface Intf1
  -instance COMPONENT1
  -name exportIntf1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case1 directory.

delete_hier_connection

Deletes a hierarchical connection from the source to the destination

Usage

```
delete_hier_connection
  -source <source-hier-path>
  -dest <dest-hier-path>
  [-remove_floating_ports <TRUE | FALSE>]
```

Description

The delete_hier_connection command deletes a hierarchical connection from the source (driver) to the destination (sink). The source and destination can be either design ports or instance pins.

Arguments

The delete_hier_connection command has the following arguments:

-source <source-hier-path>

Specifies the hierarchical path for the source with reference to the top design (design port or instance pin).

-dest <sink-hier-path>

Specifies the hierarchical path for the sink with reference to the top design (design port or instance pin).

`-remove_floating_ports <TRUE | FALSE>`

(Optional) Specifies whether to remove hanging ports after deleting. By default, this argument is set to FALSE.

Even if you set this option to TRUE, top-level ports will not be removed.

Examples

This example shows the impact of using the `delete_hier_connection` argument. Consider the following connections:

```
new design top
add_port -name p -direction IN
add_port -name q -direction OUT
new design dsg
add_port -name p -direction IN
add_port -name q -direction OUT
new design dd
add_port -name p -direction IN
add_port -name q -direction OUT
add_connection -port p -port q
select_design dsg
add_instance -name dd_inst -master dd
add_connection -port p -instance dd_inst -pin p
add_connection -instance dd_inst -pin q -port q
select_design top
add_instance -name dsg_inst -master dsg
add_connection -port p -instance dsg_inst -pin p
add_connection -instance dsg_inst -pin q -port q
```

Command with `[-remove_floating_ports FALSE]` (default value)

The following command deletes the connectivity, but it will not delete the port `p` from the design `dsg`, even though the port `p` is a hanging port after deletion.

`delete_hier_connection -source p -dest dsg_inst::dd_inst::p`
 After running the `GenerateVerilog` command, the generated RTL for the top and `dsg` modules is:

```
module top (p,
            q);

    input      p;
    output     q;

    dsg    dsg_inst ( //Instance
                    .p(), //In[1]
```

```

        .q(q)    //Out[1]
    );

endmodule

module dsg (p,
            q);

input      p;
output     q;

    dd    dd_inst ( //Instance
                .p(), //In[1]
                .q(q)  //Out[1]
            );

endmodule

```

Command with [-remove_floating_ports TRUE]

The following command deletes the connectivity and removes port p from design dsg because port p is a hanging port after deletion.

```

delete_hier_connection -source p -dest dsg_inst::dd_inst::p
-remove_floating_ports TRUE

```

After running the GenerateVerilog command, the generated RTL for the top and dsg modules is:

```

module top (p,
            q);

input      p;
output     q;

    dsg    dsg_inst ( //Instance
                .q(q)  //Out[1]
            );

endmodule

module dsg (q);

output     q;

    dd    dd_inst ( //Instance
                .p(), //In[1]
                .q(q)  //Out[1]
            );

endmodule

```

```
endmodule
```

delete_instance

Deletes the specified instance

Usage

```
delete_instance    -name <inst-name>
```

Description

The delete_instance command deletes the specified instances.

Arguments

The delete_instance command has the following arguments:

-name <inst-name>

Specifies the name of the instance to be deleted.

Examples

The following example deletes the instance named COMPONENT4:

```
delete_instance -name COMPONENT4
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case3 directory.

delete_interface

Deletes the specified interface instance

Usage

```
delete_interface <interface-inst-name>
```

Description

The `delete_interface` command deletes the specified interface instance *<interface-inst-name>* from the currently active design/component.

Examples

The following example deletes the interface instance named `Intf1` from the currently active design/component:

```
delete_interface Intf1
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case3` directory.

delete_logicalport_constraints

Removes constraints from logical port

Usage

```
delete_logical_port_constraints
  -logical_port_name <logical-port-name>
  [ -onMaster <Y | N> ]
  [ -onSlave <Y | N> ]
  [ -onSystem <Y | N> -group <group-name> ]
```

Description

The `delete_logical_port_constraints` command removes constraints from the specified logical port.

If you do not specify none of the `-onMaster`, `-onSlave`, and `-onSystem` argument, this command deletes all the constraints information.

Arguments

The `delete_logical_port_constraints` command has the following arguments:

`-logical_port_name <logical-port-name>`

Specifies the name of the logical port for which constraints should be deleted.

`-onMaster <Y | N>`

(Optional) Removes the onMaster constraints information from the specified port.

`-onSlave <Y | N>`

(Optional) Removes the onSlave constraints information from the specified port.

`-onSystem <Y | N>`

(Optional) Removes the onSystem constraints information from the specified port.

`-group <group-name>`

(Optional) Specifies the onSystem group name so that constraints of the specified groups are deleted. If you do not specify this argument, GenSys deletes all the onSystem constraints.

delete_parameter

Deletes the parameter from the most recently active component or design

Usage

`delete_parameter -name <parameter-name>`

Description

The `delete_parameter` command deletes the specified parameter from the most recently active component or design.

Arguments

The `delete_parameter` command has the following arguments:

`-name <parameter-name>`

Specifies the name of the parameter to be deleted.

delete_plane

Deletes the specified plane from the specified object.

Usage

```
delete_plane
  -plane_name <plane-name>
  [ -component <comp-name> ]
  [ -component <comp-name> -port <comp-port-name> ]
  [ -component <comp-name> -interface <intf-name> ]
  [ -instance <inst-name> ]
  [ -instance <inst-name> -pin <pin-name> ]
  [ -instance <inst-name>
    -interface_inst <intf-inst-name> ]
  [ -interface_inst <interface-name> ]
  [ -port <design-port-name> ]
```

Description

The delete_plane command deletes the specified plane from the specified object.

Arguments

The delete_plane command has the following arguments:

-plane_name <plane-name>

Specifies the name of the plane to be deleted.

-component <comp-name>

(Optional) Specifies the name of the component from which the plane has to be deleted.

-component <comp-name> -port <comp-port-name>

(Optional) Specifies the name of the port (of the specified component) from which the plane has to be deleted.

-component <comp-name> -interface <intf-name>

(Optional) Specifies the name of the interface (of the specified component) from which the plane has to be deleted.

`-instance <inst-name>`

(Optional) Specifies the name of the instance from which the plane has to be deleted.

`-instance <inst-name> -pin <pin-name>`

(Optional) Specifies the name of the pin (of the specified instance) from which the plane has to be deleted.

`-instance <inst-name> -interface_inst <intf-inst-name>`

(Optional) Specifies the name of the interface (of the specified instance) from which the plane has to be deleted.

`-interface_inst <interface-name>`

(Optional) Specifies the name of the design interface from which the plane has to be deleted.

`-port <design-port-name>`

(Optional) Specifies the name of the design port from which the plane has to be deleted.

delete_port

Deletes the specified port

Usage

`delete_port <port-name>`

Description

The `delete_port` command deletes the specified port *<port-name>* in the current design or component.

Examples

The following example deletes the port `fake` in the current design:

```
delete_port fake
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case2` directory.

delete_row

Deletes the selected row

Usage

```
delete_row  
    NULL | first | last | next | prev  
    | { -<col-name> <value> *} }
```

Description

The delete_row command deletes the specified row.

You can use the delete_row command in the following ways:

```
delete_row
```

Deletes the current row.

```
delete_row first
```

Deletes the first row.

```
delete_row last
```

Deletes the last row.

```
delete_row next
```

Deletes the row AFTER the current row.

```
delete_row prev
```

Deletes the row BEFORE the current row.

```
delete_row -size 8
```

Deletes the first row AFTER the current row where the size column value is 8.

```
delete_row -size 8 -MSB 5
```

Deletes the first row AFTER the current row where the size column value is 8 and the MSB column value is 5.

If a row is deleted from an IP-XACT table, GenSys deletes the corresponding node from the XML tree of the IP-XACT Viewer pane.

Notes

GenSys always searches down from the current row. Thus, you should use the `-<col-name>` argument carefully considering the current row. Consider the following example:

Name	LSB	MSB
P1	1	3
P2	2	5
P3	0	4
P4	1	5

If you are currently at row 1 (Name=P1) (using the [select_row](#) command or using the GUI controls) and run the following command, row 2 (Name=P2) is deleted:

```
delete_row -MSB 5
```

However, if you are currently at row 3 (Name=P3) (using the [select_row](#) command or using the GUI controls) and run the following command, row 4 (Name=P4) is deleted:

```
delete_row -MSB 5
```

Note:

You can specify the `-<col-name>` argument multiple times so that the row search is more focused.

Note:

The `-<col-name>` argument has precedence over the named row specification.

Examples

The following example deletes the row where the MSB column has value 5:

```
delete_row -MSB 5
```

diff

Invokes the Difference Merging Tool window

Usage

```
diff -design | component | interface
    -nameref <reference-object-name>
    [ -vendorref <vendor> ]
    [ -versionref <version> ]
    [ -libraryref <library> ]
    [ -namecurr <current-object-name> ]
    [ -vendorcurr <vendor> ]
    [ -versioncurr <version> ]
    [ -librarycurr <library> ]
```

Description

The diff command invokes the Difference Merging Tool window in which you can compare differences between two objects, reference object and current object.

A reference object is an object with which comparison needs to be made. A current object is an object that is being compared.

You must specify the type of the objects being compared by using -design, -component, or -interface arguments.

For details on the Difference Merging Tool window, refer to *GenSys User Guide*.

Arguments

The diff command has the following arguments:

`-nameref <reference-object-name>`

Specifies the name of the reference object.

`-vendorref <vendor>`

(Optional) Specifies the vendor name of the reference object.

`-versionref <version>`

(Optional) Specifies the version of the reference object.

`-libraryref <library>`

(Optional) Specifies the library name of the reference object.

`-namecurr <current-object-name>`

(Optional) Specifies the name of the current object. If you do not specify this argument, GenSys considers the currently loaded object as the current object.

`-vendorcurr <vendor>`

(Optional) Specifies the vendor name of the current object.

`-versioncurr <version>`

(Optional) Specifies the version of the current object.

`-librarycurr <library>`

(Optional) Specifies the library name of the current object.

dump_perl_stats

Prints the total time of all the Perl functions

Usage

dump_perl_stats

Description

The `dump_perl_stats` Tcl command prints the total time for which some Perl functions are executed in the current session of GenSys. These functions include the functions registered as generators, validation functions, change functions, or color functions in the schema.

For example, a Perl function, `x`, is called five times in the current session. In this case, the `dump_perl_stats` command will print `x` only once and will display the total time (in seconds) taken during all five runs of the Perl function, `x`.

When you exit GenSys, these details are stored in `gensys.perflog` even if you do not call the Tcl command, `dump_perl_stats`.

Note:

This command is useful when GenSys is started with the `--perflog` option.

duplicate_multifanout_port

Duplicates a port or pin present in an instance

Usage

```
duplicate_multifanout_port
  -design <design_name>
  -instance <instance_name>
  -pin <pin on the instance>
  [-naming <counter|destination>]
  [-prefix <prefix_name>]
  [-suffix <suffix_name>]
```

Description

The duplicate_multifanout_port Tcl command is used to duplicate ports or pins of an instance.

Arguments

The description of the arguments is as follows:

- design: Name of the design
- instance: Name of the instance
- pin: Name of the pin
- -naming <counter>: Port naming uses the following naming method:
 <source_port_name>_<counter>
- [-naming <destination>]: Port naming will follow following naming method <destination
 instance name>_<source_port_name>
- [-prefix <prefix_name>]: When this option is used, ports created have their name
 prefixed with its specified value
- [-suffix <suffix_name>]: When this option is used, ports created have their name suffixed
 with its specified value

Example

If the top design contains the mid_inst instance of the mid master, use the following command to duplicate the mypin port present in mid:

```
duplicate_multifanout_port -design top -instance mid_inst -pin mypin
```

By default, counter based naming is used for new ports. If the -naming <destination> option is passed, the generated name is:

```
<destination instance name>_<source_port_name>
```

In the above example, if the port named mypin has four fanouts, the counter based name is created three new ports:

```
mypin_0, mypin_1, mypin_2, mypin_3
```

The destination based name creates three new ports:

```
instance1_mypin , instnace2_mypin, instance3_mypin, instance4_mypin
```

The optional arguments, -prefix and -suffix, can also be used to control the names of the generated port.

elaborate

Resolves the connection between interfaces and vector ports in a design.

Usage

```
elaborate <design-name>
```

Description

The elaborate command resolves the connection between interfaces and vector ports in a design. The elaborate command converts the connection between vector ports to scalar bit-to-bit connection.

For example, you are connecting P[3:0] to A[3:0]. After elaboration, the connection between the ports will be scalar bit connection, such as P[3] will be connected to A[3], P[2] to A[2] and so on.

Note that elaboration should be done before RTL generation because in RTL, single bit-to-bit connectivity is required.

By default, files are evaluated in native GenSys format. If the `-as` format is supplied, GenSys converts and evaluates the files in the specified format.

You can use the `elaborate` command in the following ways:

```
elaborate
```

The above command elaborates the connections of the design currently loaded in the Design Browser. If multiple designs are present in the Design Browser, the `elaborate` command displays a dialog, which prompts you to select the design that requires elaboration.

```
elaborate <design-name>
```

Elaborates the connection of the specified design `<design-name>`.

The connection elaboration flow is order-based. Therefore, connection elaboration is done in the order in which the `add_connection` or `delete_connection` Tcl commands are specified.

Examples

The following example elaborates the connections in `test_design`:

```
elaborate test_design
```

export_all

Saves the entire hierarchy of all the top level loaded objects

Usage

```
export_all
[ -top ]
[ -as ipxact | spirit | mrs | tcl | xml | cxml ]
[ -ipxact_version <version-num> ]
-dir <dir-name>
```

Description

The `export_all` command saves the entire hierarchy of all top level loaded objects in the specified directory. This command calls the [export_data](#) command for each top level object.

By default, the default directory is `db`.

Arguments

The `export_all` command has the following arguments:

`-top`

(Optional) Specifies that only the top-level object should be saved, instead of saving the entire hierarchy.

`-as ipxact | spirit | mrs | tcl | xml | cxml`

(Optional) Specifies a format (ipxact, spirit, mrs, tcl, xml, or cxml) in which the object should be saved.

`-ipxact_version <version-num>`

(Optional) Specifies the IP-XACT version in which the object should be saved.

For example, you can specify the versions, such as 1.4 (to save in IP-XACT version 1.4) or `ieee_2009` (to save in IP-XACT version 1685-2009).

`-dir <dir-name>`

(Mandatory) Specifies the directory in which the object should be saved.

Examples

In the following example, `myDes1` and `myDes2` are saved in IPXACT format, default IPXACT 1.4, into `/tmp/myProject/ipxactDir`. The `set_save_template` settings are honored.

```
set_default_save_format ipxact
set_save_template mrs "./mrsDir/%obj.xml"
set_save_template ipxact "./ipxactDir/%obj.xml"
new design myDes1
new design myDes2
export_all -as ipxact -dir /tmp/myProject
```

Note:

The `/tmp/myProject/ipxactDir` must exist.

export_data

Saves the entire hierarchy of an object and also used to update XML data in a Word or Excel file with the data from GenSys

Usage

Usage 1

```
export_data
[ <object-name> ]
[ -top ]
[ -as mrs | spirit | ipxact | tcl | xml | cxml
[ -file <file-name> ]
[ -ipxact_version <version-num> ]
-dir <dir-name>
```

Usage 2

```
export_data
  <object-name>
  -as <excel>
  -file <original-source-of-document>
  [ -out <output-file> ]
```

Description

The `export_data` command is used for the following:

- Save the entire hierarchy of the object in the format and directory specified. ([Usage 1](#))
If the format is not specified, this commands saves the object in its original format. Even if the object is modified or not, the `export_data` command saves the entire hierarchy.
- Overwrite XML data in an MS Excel file with the data from GenSys. ([Usage 2](#))

The `export_data` command honors the settings of the [set_save_template](#) command.

The `export_data` command does not put the exported object into the library. Therefore, if you export an object and want to use it in a design, you must specify the export directory in the [set_library_path](#) command.

Arguments

The `export_data` command has the following arguments:

`<object-name>`

Specifies the name of the object to be modified.

`-top`

(Optional) Specifies that only the top-level object should be modified. A top-level object refers to the currently active object.

```
-as mrs | ipxact | tcl | xml | cxml | excel
```

Specifies a different format (mrs, ipxact, tcl, xml, or cxml) of the object being saved (For [Usage 1](#)).

Specifies the format (excel) of the file in which XML data from GenSys should be written (For [Usage 2](#)).

Note:

Values of the -as argument can be specified either in all lowercase (tcl, mrs, ipxact, cxml) or in all uppercase (TCL, MRS, IPXACT, or CXML).

```
-ipxact_version <version-num>
```

(Optional) Specifies the IP-XACT version number, such as 1.4 or ieee_2009.

For example, if you specify the version as ieee_2009, GenSys saves the object in the IP-XACT version ieee_2009.

```
-dir <dir-name>
```

(Mandatory) Specifies the directory in which the object should be saved.

```
-file <original-source-of-document>
```

Specifies the name of the file that is the original source of the document.

```
-out <output-file>
```

(Optional) Specified an output file with updated changes from GenSys database.

Examples

Example 1

In the following example, myDes is saved in IPXACT format, default IPXACT 1.4, into /tmp/myProject/ipxactDir.

```
set_default_save_format ipxact
set_save_template mrs "./mrsDir/%obj.xml"
set_save_template ipxact "./ipxactDir/%obj.xml"
new design myDes
export_data -as ipxact -dir /tmp/myProject
```

Note:

The /tmp/myProject/ipxactDir must exist.

Example 2

In the following example, the original format is IPXACT. Hence, to save it in IPXACT, you do not need to specify the format. In this example, myDes is saved in IPXACT format, default IPXACT 1.4, into /tmp/myProject/ipxactDir.

```
set_default_save_format ipxact
set_save_template mrs "./mrsDir/%obj.xml"
set_save_template ipxact "./ipxactDir/%obj.xml"
new design myDes
export_data -dir /tmp/myProject
```

Note:

The /tmp/myProject/ipxactDir must exist.

export_interface

Exports the ports of an interface instance outside the component instance to the enclosing design

Usage

```
export_interface
  -instance <inst-name>
  -interface <interface-inst-name>
  [ -name <export-interface-name> ]
  [ -prefix <string> | -postfix <string> |
    -aprefix <string> | -apostfix <string>
    -map <logical-name>:<actual-name>]
  [ -exclude <logical-name>]
  -unique <name> [ -deltadelay <ms | ps | ns> ]
  [ -naming {Expr(<expression>)} ]
  [ -description <connection-desc> ]
  [ -preserve_parameter <TRUE | FALSE> ]
```

Description

The export_interface command exports the interface on an instance to the enclosing design.

The exported and the original interfaces are connected using Interface-to-Interface connections.

Use the [set_default_port_name](#) command to set a default naming convention for the ports created on the design after running the export_interface command.

In some cases, the logical ports of an interface are mapped with some bits of the actual ports of a component. When you export such an interface, the port created on the design has the same width as that of the logical port of the interface being exported. Here, there can be three cases:

Case 1

Consider an example in which [2:2] bits of port A[0:95] of a component are mapped with [2:2] bits of a logical port, L1, of an interface, intf. In this case, when you export this interface, port A[2:2] gets exported on the design.

Case 2

Consider an example in which [0:2] and [3:6] bits of port A[0:95] of a component are mapped with [0:2] bits of logical port, L1, and [3:6] bits of logical port, L2, respectively, of interfaces, intf1 and intf2. When you export these interfaces, port A[0:6] gets exported on the design.

Case 3

Consider an example in which [0:2] and [4:7] bits of port A[0:95] of a component is mapped with [0:2] bits of logical port, L1, and [4:7] bits of logical port, L2, respectively, of interfaces, intf1 and intf2. When you export these interfaces, port A[0:7] gets exported on the design. In this case, bit 3 of the exported port A[0:7] is default tied off if the 3rd bit of port A[0:95] of the component is default tied off.

Arguments

The `export_interface` command has the following arguments:

`-instance <inst-name>`

`-interface <interface-inst-name>`

Specifies the name of the interface instance.

Note:

An Interface library is the library of the top-level interface.

`-name <export-interface-name>`

Specifies the name by which the interface should be exported. If the `-name` argument is not specified, the name of the exported interface defaults to the *same* name as the interface in the instance, as specified by the `-interface` argument.

If the interface of the same name *<export-interface-name>* already exists in the design, then the name is accepted only if the interface matches the instance interface based on the following criteria:

- The interfaces have the same interfacedef
- All ports in the interfacedef are input
- Physical ports match pair-wise in all respects (actual name, logical name, width)

If the above criteria is not met, an error is displayed.

Note:

The tieoff values in the inputs will not be carried over to the exported interface.

`-prefix <string>`

(Optional) Adds a prefix to the logical port of the instance interface and make this as the physical pin name in the exported interface.

`-postfix <string>`

(Optional) Adds a postfix to the logical port of the instance interface and make this as the physical pin name in the exported interface.

Note:

You should not use the `-aprefix|apostfix` and `-prefix|postfix` arguments in the same command.

`-aprefix <string>`

Adds a prefix to the actual port name of the instance interface and make this as the physical pin name in the exported interface. When this option is set, then the width of the port in the design is same as the width of the actual port in the master interface.

Note:

You should not use the `aprefix|apostfix` and `prefix|postfix` arguments in the same command.

`-apostfix <string>`

Adds a postfix to the actual port name of the instance interface and make this as the physical pin name in the exported interface. When this option is set, then the width of the port in the design is same as the width of the actual port in the master interface.

Note:

You should not use the -aprefix|apostfix and prefix|postfix arguments in the same command.

`-map <logical-name>:<actual-name>`

(Optional) Creates the actual-pin name for the given logical-pin name in the exported interface. The values for this argument should be specified in {} (curly braces).

`-exclude {<logical-name>}`

(Optional) Specifies a list of logical-pin names that will be excluded from actual pin mapping in the exported interface. The values for this argument should be specified in {} (curly braces).

`-unique <name>`

(Optional) Specifies that all physical pins on the exported interface get the same name as the interface or the interface postfixed with a unique name.

Note:

The -unique option can be used for backward compatibility.

`-deltadelay <ms | ps | ns>`

(Optional) Specifies a delta delay value for all pins of an interface connection to be used in netlisting.

By default, the delta delay value is ns.

If you specify a value other than ms, ps, or ns, GenSys reports an error.

`-naming {Expr(<expression>)}`

(Optional) Sets a naming convention in the local scope for ports that are generated on a design by running the export_interface command.

To set a port naming convention, specify an expression that defines a naming convention of pins.

To create an expression, use a combination of the following keywords:

Keywords	Description
si	Represents source instance name
sp	Represents source port name
sintf	Represents source interface name

Keywords	Description
<code>upper(<expression>)</code>	Changes the text generated from <expression> in upper case.
<code>slp</code>	Represents source logical port name
<code>sintfinst</code>	Represents source interface instance name
<code>sd</code>	Represents source port direction
<code>xyz</code>	Represents any string in the expression. For example, in the following expression, <code>atrenta</code> is the string: {Expr(#si_#sp_#sintf_atrenta_#upper(#si_#sp)_#slp_#sintfinst_#sd)}
<code>ebi</code>	Represents an exported bus interface name
<code>lower(<expression>)</code>	Changes the text generated from <expression> in lower case.

The following example shows the usage of the `-naming` argument:

```
export_interface -instance comp0 -interface intf0
-name Atrenta_intf -naming
{Expr(#si_#sp_#sintf_#upper(#si_#sp_#sd)_#slp_#ebi_#lower(#si)_#sd)}
```

```
-description <connection-desc>
```

(Optional) Specifies a description that is added to an interface connection, which is created with an exported interface.

```
-preserve_parameter <FALSE | TRUE>
```

(Optional) If the ports of the interface instance being exported are parameterized and the parameters controlling the ports are mapped to the parent design parameters, you can enable GenSys to:

- Create parameterized exported ports on the parent design boundary such that the ports of the exported interface instance are controlled by the parent design parameters.
- Assign the same global parameter, macro, or constant to the exported ports if the ports of the interface instance being exported are controlled by a global SystemVerilog parameter, macro, or a constant.

Parameters are preserved only if the exported ports do not already exist in the parent design boundary.

By default, parameters are not preserved. To preserve parameters, set this argument to TRUE.

See [Example 2](#).

Examples

Example 1

The following example exports interface Intf1 of instance COMPONENT1 with name exportIntf1:

```
export_interface
  -instance COMPONENT1 -interface Intf1
  -name exportIntf1
```

Note:

To refer to the testcase of this example, go to the \$SPYGLASS_HOME/examples/Genesis/case1 directory.

Example 2

This example shows the impact of using the `-preserve_parameter` argument. The following table shows the original RTL and the RTL generated after running the `export_interface` Tcl command with the `-preserve_parameter` argument.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (); parameter topP = 7; sub #(.subP(topP))s(); endmodule module sub (p1); parameter subP = 4; input [subP:0] p1; leaf l(); endmodule module leaf(); endmodule </pre>	<pre> new interface "intf.xml" set_interface_port -name L1 -direction IN save select_design sub add_interface -name "intf0_inst" -interface_ref "intf" set_interface_port -instance intf0_inst -name L1 -actual_name p1 select_design top export_interface -instance {s} -interface {intf0_inst} -name {intf0_exported} -preserve_parameter TRUE </pre>	<pre> module top (p1); parameter topP = 7; input [topP:0] p1; sub #(.subP(topP)) s (//Start of Interface intf0_inst .p1(p1) //In[7:0] L1 //End of Interface intf0_inst); endmodule </pre> <p>Note: The parameter is preserved in the exported port p1.</p>

export_pin

Exports the pin of an instance to the design.

Usage

```

export_pin
  -instance <inst-name>
  -pin <pin-name>
  [ -export_name <export-pin-name> ]
  [ -lsb <lsb> ]
  [ -msb <msb> ]
  [ -dlsb <design-lsb> ]
  [ -dmsb <design-msb> ]
  [ -deltadelay <delay-value> ]
  [ -naming {Expr(<expression>)} ]
  [ -prefix <prefix name> ]
  [ -postfix <postfix name> ]
  [ -preserve_parameter <TRUE | FALSE> ]

```

```
[ -export_parameter <TRUE | FALSE> ]
```

Description

The `export_pin` command exports a pin from an instance to a design, and creates an adhoc connection between the instance pin and the generated design port.

Use the [set_default_port_name](#) command to set a default naming convention for the generated ports.

While exporting an instance pin of width one with pin indexes, such as 0:0 or 1:1 by using the `export_pin` Tcl command, if you want the pin to appear with indexes in the generated RTL, that is, you do not want the pin to appear as scalarized in the RTL, set the `-vectorize_net` argument of the `set_rtl_option` Tcl command to TRUE. For details, see [Example 2](#).

However, please note the following points:

- If you specify the `-dlsb` and `-dmsb` arguments of the `export_pin` Tcl command, the port created will have indexes specified by these arguments.
- Else, if you specify the `-lsb` and `-msb` arguments of the `export_pin` Tcl command, this means that a bit index of a vector port is getting exported. Therefore, the vector port is scalarized in this case, provided the port width is one.

Arguments

The `export_pin` command has the following arguments:

`-instance <inst-name>`

Specifies the name of the instance whose pin has to be exported to the parent design.

`-pin <pin-name>`

Specifies the name of the pin to be exported.

In case of components having `allow_create` option set to Y, if the pin `<pin-name>` does not exist on the instance and its master component, it will be created with the same name as `<pin-name>`.

`-export_name <export-pin-name>`

(Optional) Specifies the name by which the pin should be exported. If you do not specify the `-export_name` argument, the generated pin name will be same as instance pin name.

`-lsb <lsb>`

(Optional) Specifies the LSB of the pin bit-select/part-select when you want to export only the selected bit(s) of the pin.

See [Specifying Values for Integer Arguments](#) for more details.

Note:

You must specify both the `-msb` and `-lsb` arguments. If either of them is specified, GenSys reports an error.

`-msb <msb>`

(Optional) Specifies the MSB of the pin bit-select/part-select when you want to export only the selected bit(s) of the pin.

See [Specifying Values for Integer Arguments](#) for more details.

If you do not specify the `-msb` argument and you have specified the `-lsb` argument, the MSB of the pin bit-select/part-select is assumed to be the upper value. For example, if `-lsb` is specified as 16, the `-msb` argument is taken as 31.

`-dlsb <design-lsb>`

(Optional) Specifies the LSB of the port created on the design.

`-dmsb <design-msb>`

(Optional) Specifies the MSB of the port created on the design.

`-deltadelay <delay-value>`

(Optional) Specifies delay value for the connection between the instance pin and generated design port. This delay value for connection can be dumped to RTL dump. By default, the delay unit is ns (nano seconds).

`-naming {Expr(<expression>)}`

(Optional) Sets a naming convention in the local scope for ports that are generated on a design by running the `export_pin` command.

To set a port naming convention, specify an expression that defines a naming convention.

Note:

You can also use the `set_default_port_name` TCL command to control port naming.

To create an expression, use a combination of the following keywords:

Keywords	Description
si	Represents source instance name

Keywords	Description
sp	Represents source port name
sintf	Represents source interface name
upper(<expression >)	Changes the text generated from <expression> in upper case.
slp	Represents source logical port name
sintfinst	Represents source interface instance name
sd	Represents source port direction
xyz	Represents any string in the expression. For example, in the following expression, atrenta is the string: {Expr(#si_#sp_#sintf_atrenta_#upper(#si_#sp)_#slp_#sintfinst_#sd)}
lower(<expression >)	Changes the text generated from <expression> in lower case.

The following example shows the usage of the -naming argument:

```
export_pin -instance inst1 -pin p1 -naming
{Expr(#si_#sp_#si_#upper(#si_#sp)_#lower(#si_#sp)_#sd)}
```

Note:

You can also use the set_default_port_name Tcl command to control port naming.

-prefix <prefix name>

(Optional) Specifies the prefix to be added before the name of the input pin after it is exported.

-postfix <postfix name>

(Optional) Specifies the suffix to be appended to the name of the input pin after it is exported.

Note:

When you specify the -export_name , -prefix and -postfix arguments together, the name provided to the exported pin is taken from the export_name and the -prefix and -postfix arguments are ignored.

`-preserve_parameter <FALSE | TRUE>`

(Optional) If the pin being exported is parameterized and the parameters controlling the exported pin are mapped to the parent design parameters, you can enable GenSys to:

- Create a parameterized exported port on the parent design boundary such that the exported port is controlled by the parent design parameters.
- Assign the same global parameter, macro, or constant to the exported port if the port before exporting is controlled by a global SystemVerilog parameter, macro, or a constant.

Parameters are preserved under the following conditions:

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, parameters are not exported. To export parameters, set this argument to TRUE.

`-export_parameter <FALSE | TRUE>`

(Optional) Specifies whether when the pin being exported is parameterized, must the parameters controlling the exported pin that are not mapped to the parent design parameters be exported to the parent design and start controlling from parent design. This argument can be TRUE only when the `-preserve_parameter` option is also TRUE.

Parameters will be exported and preserved under the following conditions,

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, this argument is set to FALSE. When you set this argument to TRUE, GenSys adds parameters, which are present in LSB/MSB expression of the pin being exported on the design boundary, where the pin is being exported. The parameters are exported only when they are not being overridden already from the parent design.

See [Example 3](#).

The `-preserve_parameter` argument works when the exported pin is parameterized and its index parameters are being overridden from the parent design.

Consider following RTL code:

```
module leaf(P1);
  parameter Param1 = 2;
  input [Param1-1:0] P1;
endmodule

module top();
  parameter Param2 = 4;
  leaf #(.Param1(Param2)) I1();
endmodule
```

```
endmodule
```

Next, invoking `export_pin -instance I1 -pin P1 -preserve_parameter TRUE` results in following RTL code of top:

```
module top(P1);  
input [Param2-1:0] P1; // P1 is parameterized  
leaf #(.Param1(Param2)) I1(.P1(P1));  
endmodule
```

Without the `-preserve_parameter` argument, P1 port created is [3:0].

The `-preserve_parameter` works here since Param1 is being overridden with top's parameter Param2.

With the `-export_parameter` argument, even if Param1 is not overridden from top, GenSys creates a parameter "Param1" in top and P1 port created is parameterized with it. Also, it will override I1.Param1 with top.Param1.

Examples

Example 1

The following example exports a pin from instance, I1, to design, des:

```
new component comp  
add_port -name P1 -direction IN -msb 15 -lsb 0  
save  
new design des  
add_instance -name I1 -master comp  
export_pin -instance I1 -pin P1 -msb 15 -lsb 8  
-export_name P_des -dmsb 7 -dlsb 0  
save
```

In the above example, [15:8]bits of the instance pin, P1[15:0], are exported on the design, des. The exported port, P_des, formed on the design is of [7:0] bits.

Example 2

Consider an instance pin of index 0:0 where the width of the pin is one. Consider a Tcl file containing the following commands:

```

set_rtl_option -vectorize_net TRUE
new component M1
add_port -name R1 -lsb 0 -msb 0 -direction IN
add_port -name R2 -lsb 1 -msb 2 -direction IN

new design abc
add_instance -name I1 -master M1
export_pin -instance I1 -pin R1 -export_name P1
export_pin -instance I1 -pin R2 -export_name P2
run GenerateVerilog

```

After running the above Tcl file, the following RTL (Verilog) is generated:

```

module abc (P1,
            P2);
input  [0:0]  P1;
input  [2:1]  P2;
    M1      I1 (
                .R1(P1[0]), //I:1
                .R2(P2)    //I:2
            );
Endmodule

```

In the above RTL, note that index for the exported pin P1 appear as 0:0. This is because the -vectorize_net argument of the [set_rtl_option](#) Tcl command is set to TRUE (Default setting).

Example 3

This example shows the impact of the export_parameter and preserve_parameter arguments.

The following table shows the original RTL and the RTL generated after running the export_pin Tcl command with the -preserve_parameter and the -export_parameter arguments.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top(); submodule sub(); endmodule module submodule(out); parameter P = 1; output [P:0] out; leaf l(); endmodule module leaf(); endmodule </pre>	<pre> export_pi n -instance sub -pin out -export_parameter TRUE -preserve_parameter TRUE </pre>	<pre> module top (out); parameter sub_P = 1; output [sub_P:0] out; submodule #(.P(sub_P)) sub (out); endmodule </pre> <p>Note: The parameter P of submodule is exported as sub_P to the top module and being controlled from top.</p>

export_unconnected

Exports all the unconnected terminals to the design boundary as adhoc connections

Usage

```

export_unconnected
[ -name <design-name> ]
[ -vendor <vendor-name> ]
[ -version <version-num> ]
[ -library <library-name> ]
[ -defaults <Y | N> ]
[ -port_naming_subroutine <Perl-function> ]
[ -preserve_port_names <Y | N> ]
[ -preserve_indexes <Y | N> ]
[ -preserve_parameter <TRUE | FALSE> ]
[ -export_parameter <TRUE | FALSE> ]

```

Description

The `export_unconnected` command exports all the unconnected terminals of the instances within a design to that design boundary as adhoc connections. You can specify the design by providing its VLNV information.

Use the [set_default_port_name](#) command to set a default naming convention for the ports created on the design. For the ports created on the design corresponding to the open ports on the instances within that design, the following naming convention is used:

`<instance-name>_<port-name>_EXPORTED_OPEN`

Ideally, this name should be unique and should not clash with any of the existing ports. However, if any such clash occurs, GenSys uses the existing port name (which is same as that of the generated name) at the time of exporting.

For parameterized ports, exported ports are created on the design with elaborated LSB and MSB values.

Note:

The `export_unconnected` Tcl command does not work on logical hierarchies created by using the `create_report` Tcl command.

Arguments

The `export_unconnected` command has the following arguments:

`-name <design-name>`

(Optional) Specifies the name of the design on which the unconnected ports should to be exported. If you do not specify the design name, GenSys considers the currently active design, and exports the unconnected ports of the instances within that design on that design boundary.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name of the top-level design.

`-version <version-num>`

(Optional) Specifies the version number of the top-level design.

`-library <library-name>`

(Optional) Specifies the library name of the top-level design.

`-defaults <Y | N>`

(Optional) Specifies whether ports with default connections (default tieoff or default open) should be exported to the design boundary. By default, the value of this argument is set to N, and such ports are not exported.

If you set the value of this argument to Y, GenSys exports such unconnected ports with their default values to the design boundary.

`-port_naming_subroutine <Perl-function>`

(Optional) Specifies the name of a Perl function using which you can control port names created for unconnected bits of instance terminals by using this Tcl command.

A pointer to an instance terminal having unconnected bits is passed as an argument to this perl function.

```
-preserve_port_names <Y | N>
```

(Optional) If set this argument to Y, the ports made during export_unconnected have the same name as that of pin. If it is not possible to create a pin with that name, which can be because a pin of that name already existed on the design, we use the same name that this pin would get had this argument been set to N.

By default, the value of this argument is N.

```
-preserve_indexes <Y | N>
```

Set this argument to Y so that the ports created for vector terminals (which are completely unconnected) retain the same index as that of the terminals.

By default, this argument is set to N.

Consider the following example:

```
new design uart
add_port -name aa -msb 11 -lsb 2 -direction IN
save
new design des
add_instance -name i0 -master uart
export_unconnected -export_same_indexes Y
run GenerateVerilog
```

After running the above commands, the i0_aa[11:2] port is created on the des design.

```
-preserve_parameter <TRUE | FALSE>
```

(Optional) If the pin being exported is parameterized and the parameters controlling the exported pin are mapped to the parent design parameters, you can enable GenSys to:

- Create a parameterized exported port on the parent design boundary such that the exported port is controlled by the parent design parameters.
- Assign the same global parameter, macro, or constant to the exported port if the port before exporting is controlled by a global SystemVerilog parameter, macro, or a constant.

Parameters are preserved under the following conditions:

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, parameters are not exported. To export parameters, set this argument to TRUE.

`-export_parameter <FALSE | TRUE>`

(Optional) Specifies whether when the pin being exported is parameterized, must the parameters controlling the exported pin that are not mapped to the parent design parameters be exported to the parent design and start controlling from the parent design. This argument can be TRUE only when the `-preserve_parameter` argument is also TRUE.

Parameters will be exported and preserved under the following conditions,

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, this argument is set to FALSE. When you set this argument to TRUE, GenSys adds parameters, which are present in LSB/MSB expression of the pin being exported on the design boundary, where the pin is being exported. The parameters are exported only when they are not being overridden already from the parent design.

See [Example](#).

Example

This example shows the impact of the `export_parameter` and `preserve_parameter` arguments.

The following table shows the original RTL and the RTL generated after running the `export_pin` Tcl command with the `-preserve_parameter` and the `-export_parameter` arguments.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top(); submodule sub(); endmodule module submodule(out); parameter P = 1; output [P:0] out; leaf l(); endmodule module leaf(); endmodule </pre>	<pre> export_unconnected -preserve_parameter TRUE -export_parameter TRUE </pre>	<pre> module top (out); parameter sub_P = 1; output [sub_P:0] out; submodule #(.P(sub_P)) sub (out); endmodule </pre> <p>Note: The parameter P of submodule is exported as sub_P to the top module and being controlled from top.</p>

get_glue

Retrieves behavioral RTL logic of a glue component in string format

Usage

```
get_glue
  -name <component-name>
  -language <verilog | vhdl>
  -glue <decl_glue | behav_glue>
```

Note:

This command is added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

Description

The get_glue command retrieves behavioral RTL logic of a glue component in string format.

Arguments

The get_glue command has the following arguments:

-name <component-name>

Specifies the glue-block component to be retrieved.

-language <verilog | vhdl>

Specifies the language of the glue-block component.

-glue <decl_glue | behav_glue>

Specifies the string (decl or behave) that GenSys infers while importing RTL logic.

get_library_obj

Gets path of the specified library object.

Usage

```
get_library_obj
  -type <design | component | interface>
  -name <name>
  [ -vendor <vendor-name> ]
  [ -version <version-num> ]
  [ -library <library-name> ]
```

Description

The `get_library_obj` command returns absolute path of the specified library object.

Arguments

The `get_library_obj` command has the following arguments:

`-type <design | component | interface>`

Specifies the type of library object. The `-type` argument can take the following values:

`design | component | interface`

`-name <name>`

Specifies the name of the library object for which you need to get library path.

`-vendor <vendor-name>`

(Optional) Specifies the name of the vendor of library object.

`-version <version-num>`

(Optional) Specifies the version number of library object.

`-library <library-name>`

(Optional) Specifies the name of library

`get_netlist_dir`

Returns the netlist directory

Usage

```
get_netlist_dir
```

Description

The `get_netlist_directory` command returns the netlist directory name.

If the netlist directory is not set by the [set_netlist_dir](#) command, the `get_netlist_dir` command returns the report directory.

get_proc_info

Gets the total CPU time used or total memory consumed since GenSys started

Usage

```
get_proc_info -time | -mem
```

Description

The `get_proc_info` command get the total CPU time used (in seconds) or total memory consumed (in MB) since GenSys started.

get_row

Returns the value in the specified column of the selected row in a table

Usage

```
get_row -<field-name>
```

Description

The `get_row` command returns the value in the specified column of the selected row in a table.

Example

Consider the following example in which you add three ports in a design, d:

```
new design d
add_port -name p1 -lsb 20 -msb 30
add_port -name p2 -lsb 40 -msb 60
add_port -name p4 -lsb 10 -msb 300
```

Now, if you want to retrieve the value stored in the LSB column of that port whose MSB is 300 , you can specify the `get_row` Tcl command, as shown in the following example:

```
select_table design.Portsselect_row -MSB 300get_row -LSB
```

In the above case, the `get_row` Tcl command returns the value, 10.

get_tcltoxpath

Returns an XPATH expression corresponding to a valid Tcl expression

Usage

```
get_tcltoxpath <Tcl-expression>
```

Description

The `get_tcltoxpath` command returns an XPATH expression corresponding to the specified Tcl expression, as shown in the following example:

Input to the <code>get_tcltoxpath</code> command	<code>get_tcltoxpath {[p pa]+3}</code>
Value returned by the <code>get_tcltoxpath</code> command	<code>(id('pa')+3)</code>

The Tcl expression should be a valid expression.

If GenSys is not able to convert a Tcl expression to its equivalent XPATH expression, it returns the same Tcl expression, as shown in the following example:

get_template_extension

Gets space-separated list of template extensions.

Usage

```
get_template_extension
```

Description

The `get_template_extension` command returns a space-separated list of template extensions.

get_template_path

Gets space-separated list of template paths.

Usage

```
get_template_path
```

Description

The `get_template_path` command returns a space-separated list of directory paths in which the template files are searched, if an absolute path is not specified for them.

get_xpathtotcl

Returns a Tcl expression corresponding to a valid XPATH expression

Usage

```
get_xpathtotcl <XPATH-expression>
```

Description

The `get_xpathtotcl` command returns a Tcl expression corresponding to the specified XPATH expression, as shown in the following example:

Input to the <code>get_xpathtotcl</code> command	<code>get_xpathtotcl (id('pa')+3)</code>
Value returned by the <code>get_xpathtotcl</code> command	<code>([p pa]+3)</code>

The XPATH expression should be a valid expression.

If GenSys is not able to convert an XPATH expression to its equivalent Tcl expression, it returns the same XPATH expression.

gload

Loads a design, interface, or component

Usage

```
gload
{design <design-name>
 | interface <interface-name>
 | component <comp-name>}
[ -vendor <vendor-name> ]
[ -version <version-num> ]
[ -library <library-name> ]
```

Description

The `gload` command loads the specified design, interface, or component present in Library Browser. You can also load design, interface, or component by the vendor name, library name, and/or the file version.

Arguments

The `gload` command has the following arguments:

`design <design-name>`

Specifies the name of the design to be loaded.

`interface <interface-name>`

Specifies the name of the interface to be loaded.

`component <comp-name>`

Specifies the name of the component to be loaded.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name for the design, component, or interface to be loaded.

Note:

If you do not specify the `-vendor` argument, GenSys loads the specified object of the first found vendor.

`-version <version-num>`

(Optional) Specifies the version number of the design, component, or interface to be loaded.

Note:

If you do not specify the `-version` argument, GenSys loads the specified object of the first found version.

`-library <library-name>`

(Optional) Specifies the library name for the design, component, or interface to be loaded.

Note:

If you do not specify the `-library` argument, GenSys loads the specified object of the first found library.

Note:

If none of the `-version`, `-vendor`, or `-library` arguments are specified, GenSys loads the specified object of the first found vendor, version, and library.

Notes

You can load multiple designs, interfaces, or components in one command as in the following examples:

```
gload design des1 des2 des3
```

The above command loads designs `des1`, `des2`, and `des3` of first found vendor/version/library.

```
gload component comp1 -vendor VEN_A -version 2.0
comp2 -vendor VEN_A -version 3.1
```

The above command loads component comp1 of vendor VEN_A and version 2.0 and component comp2 of vendor VEN_A and version 3.1.

```
gload interface INF1 INF2 INF3 -vendor ABC -version 1
```

The above command loads interfaces INF1 and INF2 of the first found vendor/version/library and interface INF3 of vendor ABC and version 1.

group_design

Groups instances in a design.

Usage

```
group_design
  -name <group-name>
  -instances {Space-separated-list-of-instances}
  [ -design <design-name> ]
  [ -ports <design-ports> ]
  [ -group_tieoff <TRUE | FALSE> ]
  [ -group_feedthrough <TRUE | FALSE> ]
  [ -master_name <master> ]
  [ -gen_ports <inmodule|outmodule|minport> ]
  [ -port_naming <inmodule_based | perl_based | default > ]
  [ -port_naming_subroutine <perl-func-name> ]
  [ -tieoff_port <Y | N> ]
  [ -interface_port_naming_subroutine <func-name> ]
  [ -export_unconnected <Y | N> ]
  [ -export_unconn_port_naming <Perl-function> ]
  [ -export_defaults <Y | N> ]
  -stop_defaults <Y | N>
  [ -open ]
  [ -inputs ]
  [ -outputs ]
  [ -rtl_library <library-name> ]
  [ -rtl_package <package-name> ]
  [ -remove_holes <Y | N> ] [ -preserve_port_names <Y | N> ]
  [ -preserve_interface_names <Y | N> ]
```

Description

The `group_design` command groups the specified instances into a user-defined hierarchy (subsystem) within a design.

For connections to and from the instances in the group and subsystem, necessary ports and interfaces are created on the new subsystem, appropriately. The naming of such ports is controlled by the `set_default_port_name` command.

Note:

You can add instances into an existing group, but in that case the existing group should be the only instance of its master. However, in this case, properties (like existing port name) of the original subsystem may change.

If a pin of an IP has a parameter associated with it, then during grouping, GenSys evaluates the parameter value of the port created on the group boundary.

If you do not want GenSys to evaluate the parameter value for such ports during grouping, set the value of the `-eval_param_on_group` argument of the `set_gensys_options` Tcl command to N. In this case, GenSys retains parameter expressions in MSB or LSB of the port created on group boundary and also adds the parameter on the group boundary.

By default, the `group_design` command accepts hierarchical instances:

```
group_design -master_name M -name U1 -instances {M1::I1 M1::I2}
```

The tool automatically detects whether the specified instances are hierarchical or not. All the instances that you specify for grouping must belong to the same hierarchy. Otherwise, the tool does not allow grouping and reports an error.

The default hierarchy separator is the double colon (::). To use a different hierarchy separator symbol, specify the symbol with the `set_hierarchy_separator` command. The following example shows how to use the forward slash (/) as the hierarchy separator.

```
set_hierarchy_separator /  
group_design -master_name M -name U1 -instances {M1/I1 M1/I2}
```

When groups are created in one or more instances of a multiply-instantiated module, the `group_design` command unifies the module instances by default.

In the following example, the top module has instances, M1 and M2, of the subsystem, mid. A group is created in only one of the instances' path.

```
group_design -master_name M -name U1 -instances {M1::I1 M1::I2}
```

The tool unifies mid as mid_0 and creates groups in mid_0. The tool uses integers (such as _0, _1, _2,...) to name unified instances.

Arguments

The `group_design` command has the following arguments:

`-name <group-name>`

Specifies the name of the group to be created.

`-instances {Space-separated-list-of-instances}`

Specifies a space-separated list of instances that need to be grouped (pushed into hierarchy).

`-design <design-name>`

(Optional) Specifies the name of the design in which the group of specified instances will be created. If the design name is not specified then the group will get created in the active design.

`-ports <design-ports>`

(Optional) Specifies a comma-separated list of design ports whose connections should be grouped. This option is meaningful only when either `-group_tieoff` or `-group_feedthrough` argument is also set as TRUE.

See [Example 2](#) and [Example 3](#).

`-group_tieoff <TRUE | FALSE>`

(Optional) Specifies if all the tieoff connections (partial or complete) through the specified ports (`-ports <design-ports>`) should be grouped in a hierarchy such that the tieoff value is internal to the hierarchy.

During grouping, GenSys generates ports of each connection on the hierarchy boundary. The port naming is controlled by the `set_default_port_name` command.

See [Example 2](#) and [Example 3](#).

`-group_feedthrough <TRUE | FALSE>`

(Optional) Specifies if all the feedthrough connections between the specified ports (`-ports <design-ports>`) should be grouped in a hierarchy such that they are rerouted through the design.

During grouping, GenSys generates ports of each connection on the hierarchy boundary. The port naming is controlled by the `set_default_port_name` command.

See [Example 2](#) and [Example 3](#).

`-master_name <master>`

(Optional) Specifies the name of the master of the group/subsystem. If master name is not specified, then `<group-name>_Dsgn`, will be considered as the default master name for the group/subsystem, where `<group-name>` is the name of the group.

Note:

If the master name of the group/subsystem conflicts with the name of an already existing design or component, GenSys flags an error and does not proceed for grouping.

`-gen_ports <inmodule | outmodule | minport >`

(Optional) Specifies the type of connections and necessary ports/interfaces that will be created on the newly created subsystem. This option can accept any of the following values:

inmodule	Specifies that in-module-based connections and necessary ports/interfaces will be created.
outmodule	Specifies that out-module-based connections, ports, or interfaces will be created. In this case, the number of ports on the group depends upon the number of distinct external ports connected with any instance terminal inside. Using this option minimizes the number of ports if driver is outside the group and maximizes the number of ports if driver is inside the group.
minport	Specifies that minimum number of adhoc ports are created on the group. In this case, ports get created based on the number of driver ports present (both external and internal to the group boundary) in the connectivity.

By default, this option is set to inmodule.

`-port_naming <default | inmodule_based | perl_based>`

(Optional) Specifies the port-naming convention to be followed. This option can take the following three values:

- **default:** Specifies default port naming convention in which a port is named as `<instance-name>_<pin-name>`. Here, `<instance-name>` and `<pin-name>` are derived from the instances (and their respective pins) inside or outside the group of an in-module or out-module based connection, respectively. If the port name conflicts with an existing port name then the unique port name will be `<instance-name>_<pin-name>_<suffix>`, where `<suffix>` is a numerical value.
- **inmodule_based:** Specifies name ports on the basis of instances inside the group. The port name is in `<instance-name>_<pin-name>`. If the port name conflicts with an existing port name then the unique port name will be `<instance-name>_<pin-name>_<suffix>`, where `<suffix>` is a numerical value.

This type of naming convention is useful if you want to name ports based on instances inside the group, but specify out-module-based connections or create maximum/minimum number of ad-hoc ports on the group.

- **perl_based:** This type of port naming convention enables you to specify your own Perl function using the `-port_naming_subroutine` option. If you want to use the alias names of instances to name the ports of a group, you can provide alias names for instances as attributes inside the Perl function.

Note:

For IN-IN or OUT-OUT type of connections, a unique port gets created on the group.

Note:

While creating minimum/maximum ports on a group, it is important to identify the driver port. However, identification of the driver port is not possible if one side of the connection contains an INOUT port. Particularly for this type of connections, ports will be generated based upon instance-port combination inside the group.

```
-port_naming_subroutine <perl-func-name>
```

(Optional) Specifies a Perl function that provides a logic to name ports of a group. This option is used when the `-port_naming` option is set to perl-based.

This argument controls the naming of pins made due to adhoc/tie-off connections.

```
-tieoff_port <Y | N>
```

(Optional) Specifies if tieoff ports will be created corresponding to the tieoff connections on the newly created subsystem.

By default, this option is set to N.

```
-interface_port_naming_subroutine <func-name>
```

(Optional) Specifies the name of the perl function *<func-name>* that specifies the naming convention of the interface ports on the group instance. The first argument to the function is the interface connection type-item pointer inside the group and second argument is interface connection type-item pointer outside the group.

For more details on interface port naming convention, refer to the *Specifying Interface Port Naming Conventions* topic of *GenSys User Guide*.

Note:

Currently, logical connections (nets) and cross-hierarchical connections are not handled.

```
-export_unconnected <Y | N>
```

(Optional) Specifies whether the unconnected terminals of all the instances within the group design should be exported to the group boundary as adhoc connections.

By default, the value of this argument is set to N, and the unconnected terminals are not exported to the group boundary as adhoc connections.

Set the value of this argument to Y to export all the unconnected terminals to the group boundary as adhoc connections. In this case, the name of the ports created corresponding to the open ports on the instance are inmodule based, and have the following name format:

```
<instance-name>_<port-name>
```

Ideally, this name should be unique and should not clash with any of the existing ports. However, if such clashes happen, GenSys uses those ports at the time of exporting.

For parameterized ports, exported ports are created on the top design with elaborated values of LSB and MSB.

`-export_unconn_port_naming <Perl-function>`

(Optional) (Optional) Specifies the name of a Perl function that enables you to control port names created when the `-export_unconnected` argument of this Tcl command is set to TRUE.

A pointer to an instance terminal having unconnected bits is passed as an argument to this Perl function.

You can use this argument to control port naming of the following type of pins:

- Pins that are intentionally open
- Pins that are left open un-intentionally
- Pins that are default tied-off

`-stop_defaults <Y | N>`

Specifies that the default values (tieoff or open) of unconnected ports of instances inside the group should not be propagated to the group boundary.

By default, the value of this argument is set to N, and the default values are propagated to the group boundary. Set the value of this argument to Y to stop the propagation of the default values to the group boundary.

`-export_defaults <Y | N>`

(Optional) Specifies whether ports with default connections (default tieoff or default open) should be exported to the group boundary. By default, the value of this argument is set to Y, and such ports are exported to the group boundary.

Set the value of this argument to N if you do not want to export such unconnected ports with their default values to the group boundary.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

Do not use this command with the `-stop_defaults` argument.

`-open`

(Optional) Specifies whether the intentional open ports should be exported to the subsystem boundary.

By default, the value of this argument is set to N, and all the intentional open ports are not be exported to the subsystem boundary.

Set the value of this argument to Y to export all the ports, which are marked as intentional open, to the subsystem boundary. In this case, the intentionally open ports will be exported outside the boundary, and the new subsystem port that was created will be marked as intentional open.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-inputs`

(Optional) Specifies whether the unconnected input ports should be exported to the subsystem boundary.

By default, the value of this argument is set to Y, and the unconnected input ports are exported to the subsystem boundary by creating a port and connecting it to the unintentional input pin accordingly.

Set the value of this argument to N to turn off this default behavior.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-outputs`

(Optional) Specifies whether the unconnected open output and inout ports should be exported to the subsystem boundary.

By default, the value of this argument is set to Y, and the unconnected open output and inout ports are exported to the subsystem boundary by creating a port and connecting it to the unintentional output/inout pin accordingly.

Set the value of this argument to N to turn off this default behavior.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-rtl_library <library-name>`

(Optional) Specifies the library information of the component. This argument value is used when the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE, the component declaration is not dumped in VHDL, but use library and package statements are dumped. In such case, the component declaration is picked from the library specified in this argument.

`-rtl_package <package-name>`

(Optional) Specifies the package information of the component. This argument value is used when the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE, the component declaration is not dumped in VHDL, but use library and

package statements are dumped. In such case, the component declaration is picked from the package specified in this argument.

`-remove_holes <Y | N>`

(Optional) Specifies whether holes created during the creation of a group hierarchy should be removed.

A hole gets created due to unconnected bits. For example, consider a vector port of five bits. Now if you connect two lower bits and two upper bits of this port and then run the `group_design` Tcl command, the middle bit remains unconnected resulting in a hole.

Set this argument to Y to eliminate such holes.

By default, the value of this argument is N.

Example

In the following snippet, since the `remove_holes` argument is set to Y, the {inst1_P1[6:2]} port is made on the group boundary with the width 5.

```
new component compl
add_port -name P1 -direction OUT -lsb 0 -msb 20
add_port -name P2 -direction IN -lsb 0 -msb 20
save compl
new design top
add_instance -name inst1 -master compl
add_instance -name inst2 -master compl
add_instance -name inst3 -master compl

add_connection -instance inst1 -pin P1 -lsb 2 -msb 3
               -instance inst2 -pin P2 -lsb 2 -msb 3

add_connection -instance inst1 -pin P1 -lsb 8 -msb 10
               -instance inst3 -pin P2 -lsb 8 -msb 10

group_design -instances { inst1 } -name grp_D -remove_holes Y
-gen_pors inmodule
```

If you set the `remove_holes` to N, the {inst1_P1[10:2]} port is with width 9. In this case, there is a hole from bit 4 to bit 7 inside the port, which means that these bits are not connected to anything.

`-preserve_port_names <Y | N>`

(Optional) When you set this argument to N, the name of ports created on the group boundary is of the format: <instance-name>_<pin-name>.

When the user sets this argument to Y, the name of the ports created on the group boundary is of the format:<pin-name>

However, if the group boundary already has a port of the same name, the instance name is prefixed before the name of the new port being created on the boundary.

By default, the value of this argument is N.

Example

In the following snippet, the p1 port is made on the group boundary.

```
new component compl
add_port -name p1 -direction OUT -lsb 0 -msb 2
save
new design d
add_port -name dport -direction OUT -lsb 0 -msb 2
add_instance -name inst1 -master compl
add_connection -port dport -instance inst1 -pin p1
group_design -instances { inst1 } -name grp_D
-preserve_port_names Y
```

If you set the argument to N, the inst1_p1 port is created.

```
-preserve_interface_names <Y | N>
```

(Optional) When the user sets this argument to N, the name of interfaces created on the group boundary is of the format:<instance-name>_<interface-name>

When the user sets this argument to Y, the name of interfaces created on the group boundary is of the format:<interface-name>

However, if the group boundary already has an interface of the same name, the instance name is prefixed before the name of the new interface being created on the boundary.

By default, the value of this argument is N.

Examples

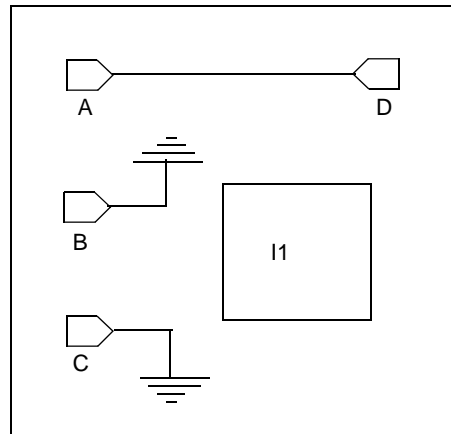
Example 1

The following example groups instances C10, C22, and C41 as a subsystem named Group within the current design. The master design of this subsystem is MasterGroup. Minimum ports will be created based on a perl function named perlfunc. Tieoff ports will be created on the hierarchy corresponding to tieoff connections inside.

```
group_design -name Group - instances {C10 C22 C41}
-master_name MasterGroup -gen_ports minport
-port_naming perl-based
-port_naming_subroutine perlfunc -tieoff_port Y
```

Example 2

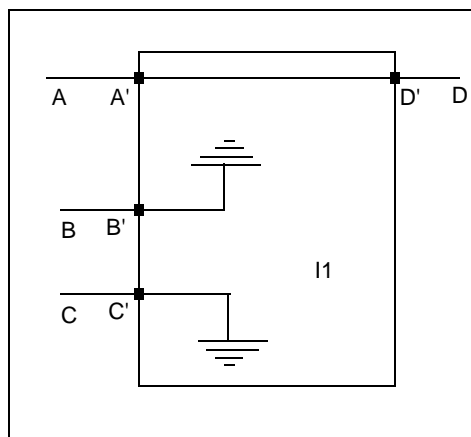
Consider the following design:



Now consider that you specify the following command:

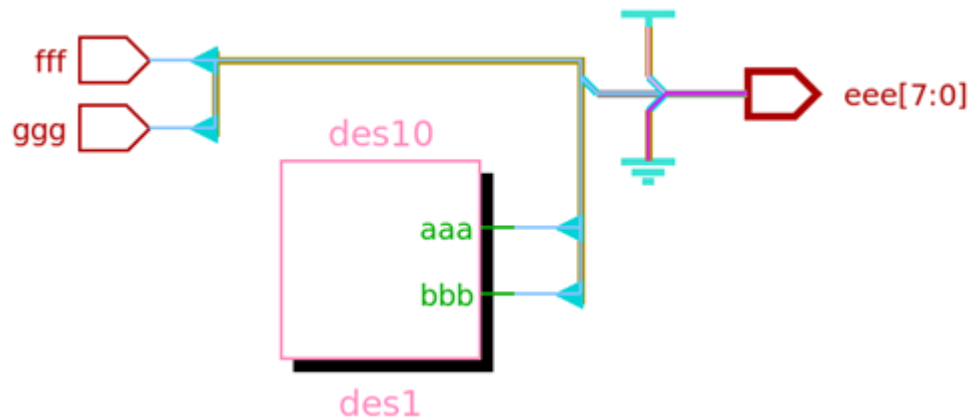
```
group_design -ports {A,B,C,D} -group_tieoff TRUE
-group_feedthrough TRUE
```

After running the above command, the design changes to the following:



Example 3

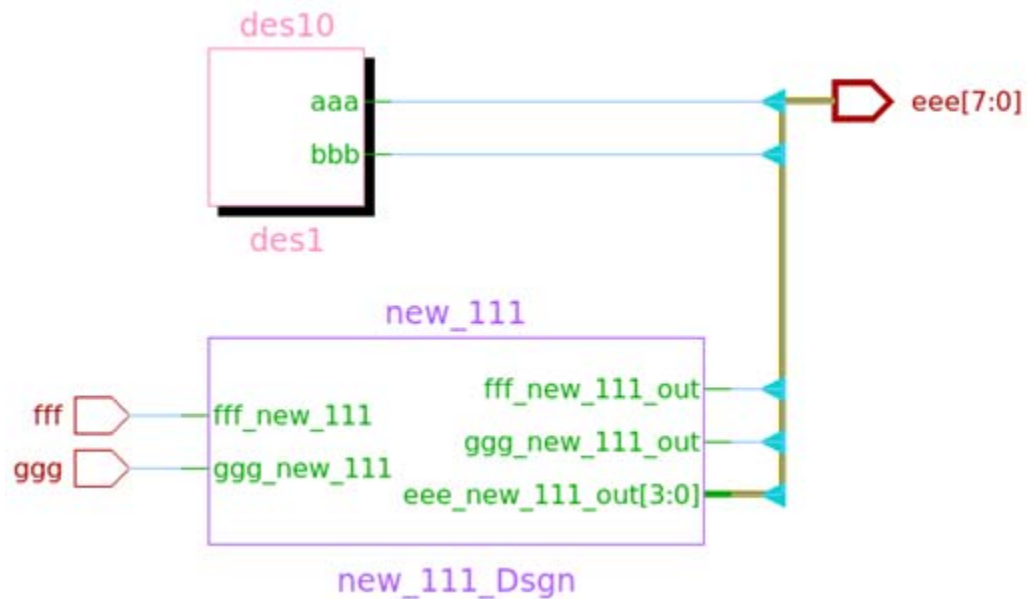
Consider the following figure in which the eee port is partially tiedoff, partially adhoc, and partially feedthrough:



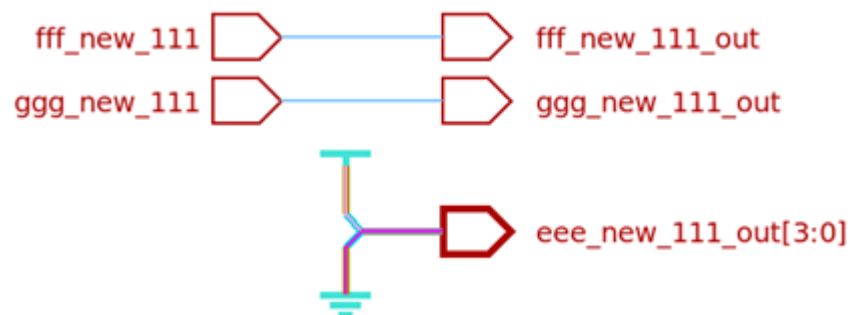
In the above example, consider that you want to push the eee port inside the new hierarchy new_111 by specifying the following command:

```
group_design -name new_111 -ports {eee} -group_tieoff TRUE  
-group_feedthrough TRUE
```

On running the above command, the design changes to the following:



In the above figure, the new_111 hierarchy has the following internal structure:



group_glue

Creates a glue instance that can be moved across hierarchies like any design instance

Usage

```
group_glue
  -instances <glue-instance-name>
  -design <parent-design-name>
```

```
[ -group_instance_name <glue-hierarchy-instance> ]  
[ -group_name <group-name> ]
```

Description

The `group_glue` command creates a glue instance from one or multiple virtual hierarchies representing glue logic such that unlike a virtual hierarchy of a glue logic, the corresponding glue module can be moved across hierarchies like any other design instance.

This process is known as glue solidification because unlike the virtual hierarchy represented by a dotted box in schematic, the corresponding glue module instance as a solid box that acts like any other design instance.

GenSys, while importing RTL, groups all glue logic, i.e. always/assign/process blocks in a pseudo hierarchy (block) known as a Glue block. During RTL generation, this pseudo block is dissolved. This Glue block cannot be moved across hierarchies. However, with the `group_glue` command, you can solidify this block and now can move it just like any user instance.

Note:

This command works only if the `-enable_group_glue` argument of the `set_gensys_options` Tcl command is set to TRUE.

For information on performing glue solidification in GUI, refer to the *Solidifying Glue Instances* topic of *Gensys User Guide*.

Arguments

The `group_glue` command has the following arguments:

`-instances <glue-instance-name>`

Specifies a space-separated list of glue instances that should come under a separate hierarchy so that this hierarchy like any other design instance, which can be moved across hierarchies.

`-design <parent-design-name>`

Specifies the parent name of the glue instances specified by the `-instances` argument of the `group_glue` command.

`-group_instance_name <glue-hierarchy-instance>`

Specifies the name of the instance to be created that contains the glue instances specified by the `-instances` argument of the `group_glue` command. This instance acts like any other design instance that can be moved across hierarchies.

`-group_name <group-name>`

Specifies the master name for the instance specified by the `-group_instance_name` argument of the `group_glue` command.

Consider the following RTL code:

```
`define CPU_NUM 3

module fpu_pipe(
output logic [`CPU_NUM:0] Flush_VarLatUop
);
wire a;
reg b;
endmodule

module fpu();

logic [`CPU_NUM:0] Flush_VarLatUop[`CPU_NUM:0];

generate for(genvar pipe_i = 0; pipe_i <= `CPU_NUM; pipe_i++) begin :
u_pipe_wrap
    fpu_pipe ufpu_pipe(
        .Flush_VarLatUop(Flush_VarLatUop[pipe_i])
    );
end endgenerate
endmodule
```

When importing this RTL in Gensys, the module `fpu` has a Glue block instance with name `u_pipe_wrap_inst`. Now, if you want to solidify it, specify the following:

```
group_glue -instances u_pipe_wrap_inst -design fpu
-group_name fpu_pipe_wrap
```

Therefore, this glue instance is now solidified with its new master block name "`fpu_pipe_wrap`". This instance "`u_pipe_wrap_inst`" can now be moved using the `move_instance` TCL command.

On generating, the RTL for new module "`fpu_pipe_wrap`" is as follows:

```
module fpu_pipe_wrap (Flush_VarLatUop);

output logic [`CPU_NUM:0] Flush_VarLatUop [`CPU_NUM:0] ;
//
generate for(genvar pipe_i = 0; pipe_i <= `CPU_NUM; pipe_i++) begin :
u_pipe_wrap
    fpu_pipe ufpu_pipe(
        .Flush_VarLatUop(Flush_VarLatUop[pipe_i])
    );
end
```

```
end endgenerate
//
endmodule
```

help

Shows the help for the specified Tcl command.

Usage

```
help
[ <command-name> ]
```

Description

The help command displays the help information related to a specific Tcl command.

Refer to [Getting Help on Commands](#) section for more details.

history

Shows the command history

Usage

```
history
```

Description

The history command displays the list of Tcl command, used in the current session of GenSys, in the Log tab of the Message window in GenSys GUI.

import_data

Reads in the structured data in an input Word or Excel Office XML file and populates GenSys database

Usage

```
import_data
  -file <xml-file>
  -type [<excel> ]
```

Description

The import_data command reads in the structured data in an input Word or Excel Office XML file and populates the GenSys database.

Arguments

The import_data command has the following arguments:

-file <xml-file>

Specifies an input Word or Excel Office XML document.

-type <excel>

Specifies the type file (excel) from which data is used to overwrite the source XML file in GenSys.

incr_sch_add

Populates the Incremental Schematic

Usage

```
incr_sch_add
  [ -clear ]
  [ -inst <hier_inst-name> ]
  [ -port <hier-port-name> ]
```

Description

The incr_sch_add command populates the Incremental Schematic with the specified ports and instances specified in the batch mode.

Note:

This command can be used only if the `-batchSch` command-line option is specified while running GenSys in the batch mode.

Arguments

The `incr_sch_add` command has the following arguments:

`-clear`

(Optional) Clears the the Incremental Schematic.

`-inst <hier_inst-name>`

(Optional) Specifies the name of the instance for which the incremental schematic view needs to be displayed.

`-port <hier-port-name>`

(Optional) Specifies the name of the port for which the incremental schematic view needs to be displayed.

incr_sch_print

Prints the Incremental Schematic

Usage

```
incr_sch_print
[ -prn_cmd <command> ]
[ -file <file-name> ]
[ -color_mode <mono | color | colorinv> ]
[ -orient <auto | landscape | portrait> ]
[ -sheet_size <size> ]
[ -scale <full | fullfit | current>]
[ -with_highlight ]
```

Description

The `incr_sch_print` command prints the Incremental Schematic. This command can be specified in the batch mode.

Note:

This command can be used only if the `-batchSch` command-line option is specified while running GenSys in the batch mode.

Arguments

The `incr_sch_print` command has the following arguments:

`-prn_cmd <command>`

(Optional) Specifies the print command of the printer. For example, the print command for Linux platform is `lpr`.

`-file <file-name>`

(Optional) Specifies the name of the file in which the output of the `incr_sch_print` Tcl command is saved.

`-color_mode`

(Optional) Specifies the color mode of the page. This argument accepts one of the following values:

`mono`, `color`, or `colorinv`

`-orient`

(Optional) Specifies the page orientation. This argument accepts one of the following values:

`auto`, `landscape`, or `portrait`

`-sheet_size <size>`

(Optional) Specifies the size of the sheet. You can specify any of the following values:

A	A	A	A	A	B	C	D	E	F	G
4	1	2	3							

`-scale`

(Optional) Specifies the scale of the page. You can specify any of the following values:

Value	Purpose
full	Prints/saves the full schematic at the current scale. GenSys considers the previously set scale as the current scale in this case. For example, the current scale set by using the <code>-factor</code> argument of the preceding incr_sch_set_zoom_factor Tcl command is used.
fullfit	Prints/saves the full schematic ignoring the current scale

Value	Purpose
current	Prints the current window on the currently scaled schematic

`-with_highlight`

(Optional) Enables the printing of highlight information. By default, the highlighted objects are printed in the normal mode.

incr_sch_set_zoom_factor

Sets the zoom factor of the Incremental Schematic

Usage

```
incr_sch_set_zoom_factor  
  -factor <value> | -fullfit
```

Description

The `incr_sch_set_zoom_factor` command sets the zoom factor in the batch mode for the Incremental Schematic.

Note:

While running GenSys in the batch mode, this command can be used only if the `-batchSch` command-line option is specified.

Note:

This command is effective only when schematic has been loaded.

Arguments

The `incr_sch_set_zoom_factor` command has the following arguments:

`-factor <value>`

(Optional) Specifies the zoom factor. This argument accepts an integer value.

The specified zoom factor is absolute (and not relative). If you specify `mod_sch_set_zoom_factor -factor 0.5` multiple times, only the first specified command changes the size. In contrast, the north-east upstroke, tagged as zoom 0.5, in the schematic is relative to the current size.

`-fullfit`

(Optional) Fits the entire schematic view in the visible area of the Incremental Schematic.

Note:

You must specify either `-factor` or `-fullfit` argument with this Tcl command. If you specify both these arguments, GenSys considers the `-fullfit` argument.

infer_interface_connections

Infers interface connections

Usage

```
infer_interface_connections
[ -design <design-name>
  [ -vendor <vendor-name> ]
  [ -version <version-num> ]
  [ -library <library-name> ] ]
```

Description

The `infer_interface_connections` command infers interface connections on the given design having adhoc connections and interfaces instantiated.

If you have not specified any design, GenSys considers the active design.

Arguments

The `infer_interface_connections` command has the following arguments:

`-design <design-name>`

(Optional) Specifies the design name.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name.

`-version <version-num>`

(Optional) Specifies the version number.

`-library <library-name>`

(Optional) Specifies the library name.

infer_interfaces

Infers interfaces on the specified design or component

Usage

```
infer_interfaces
[ -design <design-name> ]
[ -vendor <design-vendor-name> ]
[ -version <design-version-num> ]
[ -library <design-library-name> ]
|
[ -component <component-name>
[ -vendor <comp-vendor-name> ]
[ -version <comp-version-num> ]
[ -library <comp-library-name> ]
-interfaces <interface-libraries>
[ -connections <true | false>
[ -min_ports <num> ]
[ -hierarchy <true | false> ]
[ -filename <file-name> ]
[ -merge <True | False> ]
[ -fanout <num> ]
[ -suggested ]
]
```

Description

The `infer_interfaces` command infers interfaces on the specified design or component.

In case of design, GenSys infers interfaces hierarchically on all the sub-designs and components instantiated on it.

Note:

To infer interface connections, use the [incr_sch_add](#) Tcl command.

Arguments

The `infer_interfaces` command has the following arguments:

`-design <design-name>`

(Optional) Specifies the name of the design on which interfaces should be inferred.

If you do not specify this argument, GenSys infers interfaces on the active design.

`-vendor <design-vendor-name>`

(Optional) Specifies the vendor name of the design.

`-version <design-version-num>`

(Optional) Specifies the version number of the design.

`-library <design-library-name>`

(Optional) Specifies the library name of the design.

`-component <component-name>`

(Optional) Specifies the name of the component on which interfaces should be inferred.

If you do not specify this argument, GenSys infers interfaces on the component design.

`-vendor <comp-vendor-name>`

(Optional) Specifies the vendor name of the component.

`-version <comp-version-num>`

(Optional) Specifies the version number of the component.

`-library <comp-library-name>`

(Optional) Specifies the library name of the component.

`-interfaces <interface-libraries>`

(Optional) Specifies a comma-separated list of interface libraries to be considered for port matching.

`-connections <true | false>`

(Optional) Set this argument to true to infer interfaces on-the-fly based on existing connections present in a design.

You may use this argument when you do not have sufficient design details, such as any existing interface on the design. You may also use this argument when no interface library is present.

`-min_ports <num>`

(Optional) Specifies the minimum number of ports required to create an interface.

When you set the `-connections` argument of this Tcl command to true, GenSys infers only those interfaces that have total number of ports greater or equal to the number specified by this argument.

By default, the value of this argument is 5.

`-hierarchy <true | false>`

(Optional) Specifies if interfaces should be inferred only at the top-level of a design or even for the subsystems in the design hierarchy.

When you set this argument to true (along with the `-connections` argument of this Tcl command to true), GenSys infers interfaces at the top-level as well as at subsystems in the design hierarchy.

By default, this argument is set to false.

`-filename <file-name>`

(Optional) Specifies the name of the file to be generated that stores information that is used for inferring interfaces.

The following sample file shows the format in which such information is dumped:

```
#####
Total number of interfaces suggested for the component C2 is :: 2
Total number of interfaces suggested for the component C3 is :: 2
Total number of interfaces suggested for the component TOP is :: 1
Total number of interfaces suggested for the component c1 is :: 3
```

```
Total number of interfaces suggested for the Design hierarchy TOP
is :: 8
```

ComponentName	Interface	LogicalPort	ActualPort
TOP	C2__TOP0	L_BO1	TO1
		L_BO2	TO2
ComponentName	Interface	LogicalPort	ActualPort
C2	C2__TOP0	L_BO	BO1
		L_BO2	BO2
		L_BO3	BO3
C2	c1__C20	L_AO1	BI1
		L_AO2	BI2
		L_AO3	BI3
		L_AO4	BI4

ComponentName	Interface	LogicalPort	ActualPort
C3	C3__c10	L_DO1	DO1
		L_DO2	DO2
C3	c1__C30	L_AO1	DI1
		L_AO2	DI2
		L_AO3	DI3
ComponentName	Interface	LogicalPort	ActualPort
c1	C3__c10	L_DO1	AI1
		L_DO2	AI2
c1	c1__C20	L_AO1	AO1
		L_AO2	AO2
		L_AO3	AO3
		L_AO4	AO4
c1	c1__C30	L_AO1	AO1
		L_AO2	AO2
		L_AO3	AO3
#####			

-merge <True | False>

(Optional) If you set this argument to true, Gensys tries to merge interfaces on a component and design that are from the same interface library.

If this merge operation occurs successfully, extra interfaces are removed from the component/design.

-fanout <num>

(Optional) Specifies the maximum fan-out of a port/pin such that the port/pin that has a high fan-out is not mapped to an interface during interface inference.

Usually, clock and reset nets are not the part of interfaces. Therefore, any port/pin having a high fan-out is not mapped to an interface.

The default value of this argument is 2.

Note:

Use this argument only if the -connections argument of this Tcl command is set to true.

-suggested

(Optional) When you specify this argument, GenSys suggests how many interfaces can be inferred based on the adhoc connectivity.

You can also view the report showing suggested interfaces by clicking the *Reports -> Suggested Interfaces* menu option.

Note:

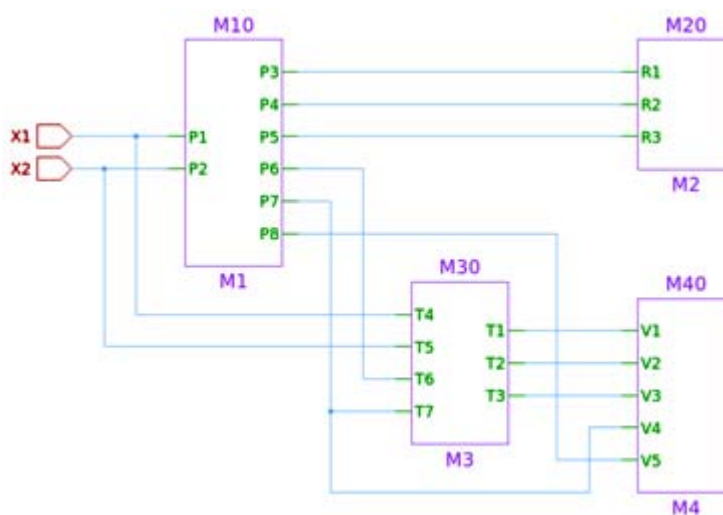
Use this argument only if the -connections argument of this Tcl command is set to true.

Note:

Run the `infer_interfaces -connections true -suggested` command before running the `create_report` to generate suggested interfaces.

Examples

Consider the connectivity shown in the following schematic:



Now consider that you run the following Tcl command:

```
infer_interfaces -connections true -fanout 3-min_ports 2
```

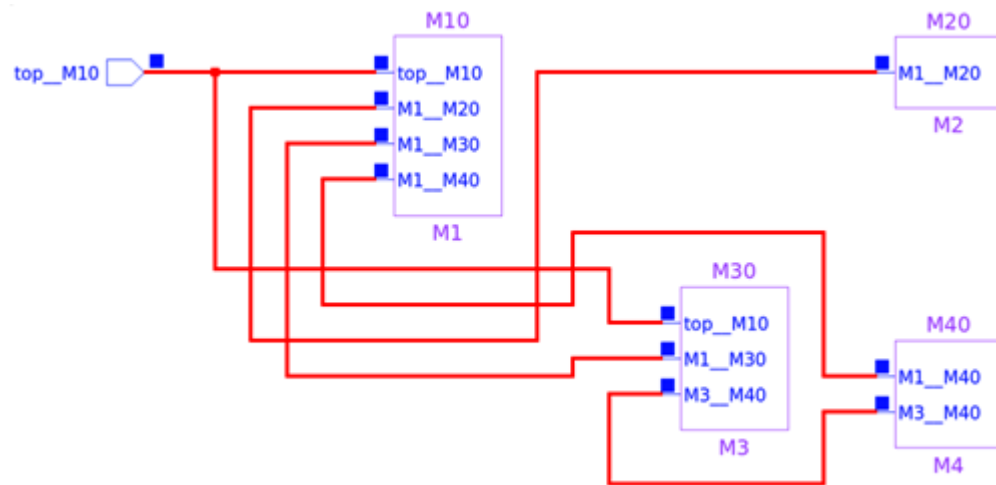
When you run the above Tcl command, GenSys infers two interface libraries, `top_M1` and `M1_M2`, having logical ports as `{L_X1, L_X2}` and `{L_P3, L_P4, L_P5}`, respectively. Following are the details:

- M1 will have the following four interface instantiated on it:
 - M1_M20 (master busdef M1_M2)
 - M1_M30 (master busdef top_M1)
 - M1_M40 (master busdef top_M1)
 - top_M10 (master busdef top_M1)
- M2 will have one interface instantiated as M1_M20 (master busdef M1_M2).
- M3 will have the following three interfaces instantiated on it:

- M1_M30 (master busdef top_M1)
- M3_M40 (master busdef M1_M2)
- top_M10 (master busdef top_M1)
- M4 will have the following two interfaces instantiated on it:
 - M1_M40 (master busdef top_M1)
 - M3_M40 (master busdef M1_M2)
- The top-level design will have only the top_M10 (master busdef top_M1) interface instantiated on it.

Therefore, interface libraries are re-used wherever possible.

Initially, when connectivity was the ad-hoc connectivity, there were 13 connections in total. However, when the connectivity is abstracted in terms of interfaces, there are only five interface connections, as shown in the following figure, after running the `infer_interface` Tcl command):



Therefore, a lot of connectivity abstraction can take place by using the `infer_interface` Tcl command.

insert_row

Inserts a new row in the table

Usage

```
insert_row    NULL | prev | next | { -<col-name> <value> }
```

Description

The `insert_row` command inserts a new row (with blank cells) in the table at the specified location.

You can use the `insert_row` command in the following ways:

```
insert_row
```

Inserts a new row after the current row.

```
insert_row next
```

Inserts a new row AFTER the current row.

```
insert_row prev
```

Inserts a new row BEFORE the current row.

```
insert_row -size 8
```

Inserts a new row AFTER the row where the size column value is 8.

```
insert_row -size 8 -MSB 5
```

Inserts a new row AFTER the row where the size column value is 8 and the MSB column value is 5.

You can set a current row by using the [select_row](#) command. On successful run of the `insert_row` command, the newly added row becomes the current row.

In case of a newly inserted row in an IP-XACT table, GenSys inserts a corresponding node in the XML tree of the IP-XACT Viewer pane. This node can be cross-probed to the IP-XACT table.

Notes

GenSys always searches down from the current row. Thus, you should use the `-<col-name>` argument carefully considering the current row. Consider the following example:

Name	LSB	MSB
P1	1	3

Name	LSB	MSB
P2	2	5
P3	0	4
P4	1	5

If you are currently at row 1 (Name=P1) (using the [select_row](#) command or using the GUI controls) and run the following command, a new row is inserted after row 2 (Name=P2):

```
insert_row -MSB 5
```

However, if you are currently at row 3 (Name=P3) (using the [select_row](#) command or using the GUI controls) and run the following command, a new row is inserted after row 4 (Name=P4):

```
insert_row -MSB 5
```

Note:

You can specify the `-<col-name>` argument multiple times so that the row search is more focused.

Note:

The `-<col-name>` argument has precedence over the named row specification.

load

Loads designs, components, or interfaces.

Usage

```
load [ -as <MRS | AUTORTL | IPXACT | CSV> ]  
      <file-name-list>
```

Description

The load command loads the designs, components, and interfaces from specified XML format files in the GenSys library.

The last found object becomes as the current object.

Arguments

The load command has the following arguments:

`-as <MRS | AUTORTL | IPXACT | CSV>`

(Optional) Specifies the MRS , AUTORTL, CSV, or IP-XACT files that needs to be loaded.

By default, MRS files are loaded.

However, please note the following points:

- Currently, CSV files can be dumped or restored only for an individual GenSys table, not for entire design, component, or Interface. Also, CSV restore can be done only through TCL command not through GUI.
- When a CSV file is saved on Windows platform, a special character, ^M, gets added to the file. These characters might cause parsing errors. Therefore, it is recommended to use the dos2unix command to remove these characters before loading the CSV files into GenSys.

`<file-name-list>`

Specifies space-separated list of names of XML format files.

Important Notes

The process of loading a component is as follows:

1. GenSys searches for the component with exact VLNV in the root first and if found then loads it from there.
2. If not found in the root, GenSys searches the component in the library for exact match in any of the formats (XML, IP-XACT, or Tcl). If found, then load it from the library otherwise, flag error message 1717 that component with specified VLNV not found.

The process of loading an interface is as follows:

1. GenSys searches for an interfaceDef with exact VLNV in the root and if found then loads it from the root.
2. If not found in the root, GenSys searches for the exact VLNV match in any of the formats (XML, IP-XACT, or Tcl). If found, GenSys loads it from the library.
3. If still not found, GenSys searches for the highest version interfaceDef with same name in the root, If found, loads it and also gives a warning message 1098 that no interfaceDef with exact VLNV found and loading the highest version file from root.

4. If highest version file is not found in the root, GenSys searches for the highest version file of any format (XML, IP-XACT, or Tcl) from the library. If found, loads it from library and also gives a warning message 1098 that no interfaceDef with exact VLNV is found and loading the highest version file from library.
5. If still not found, GenSys searches for the same named interfaceDef in the root and if found loads it and also gives a warning message 1099 that no interfaceDef with same or higher version found loading interface with same name from the root.
6. If named match not found in the root search for the name based match in any format (XML, IP-XACT, or Tcl) from the library. If found, loads it from the library and also gives the warning message 1099 that no interfaceDef with same or higher version found loading interface with the same name from library.
7. If still not found, GenSys gives an error message 1717 that interfaceDef with specified VNLV not found.

Note:

1098/1099 messages flagged for library also contain the time stamp and type of file (IP-XACT, MRS, or TCL) getting loaded along with the list of other files (with time stamp and type) lying in the library.

Note:

User can specify which files (IP-XACT, XML, TCL) to load from the library using the [set_log_dir](#) TCL command. This is useful when files of same VLNV exist in all formats.

Note:

In case multiple objects matching the search criteria exist (this will happen if we have resorted to the last name matching only), then it picks up the object which is in the directory which was given first in the [set_library_path](#) TCL command. If there are multiple files with same VLNV in the same directory, GenSys loads the file with the latest timestamp. For more details, refer to [set_library_path](#).

After loading components and designs, the load command prints statistics for the loaded file. An example of these details is given below:

```
%load newdes.xml
INFO[0666] : Load statistics for design 'newdes' -
CPU time - 0.19 seconds, Memory usage - 0 MB, Instances - 2, Connections
- 1
%load top_des.xml
INFO[0666] : Load statistics for design 'top_des' -
CPU time - 4.77 seconds, Memory usage - 10 MB, Instances - 27,
Connections - 6
```

Note:

Currently, these details are not printed for interfaces.

Examples

The following example loads the objects described in the MRS file named abc.xml:

```
load abc.xml
```

load_rtl

Loads RTL of design or components

Usage

```
load_rtl <list-of-RTL-files>
[ -component ]
[ -import_glue ]
[ -autosave ]
[ -f <file-name-with-spyglass-options> ]
[ -hierarchy_view_name <hierarchy-modelView> ]
[ -hdl_view_name <synthesis-modelView> ]
[ -merge_views <TRUE | FALSE> ]
[ -vendor <vendor-name> ]
[ -version <version-num> | <Tcl-function> ]
[ -library <library-name> ]
[ -import_all <TRUE | FALSE> ]
[ -stop_level <int-value> ]
[ -ui ]
[ -instance_threshold <value> ]
[ -project <project-file-name> ]
[ -auto_blackbox <TRUE | FALSE> ]
```

Description

The load_rtl command loads the design or components from specified RTL files in the GenSys library. You can also provide the InterfaceLib files to infer interfaces on components.

Before loading RTL, it is important to clean the RTL using SpyGlass.

Arguments

The load_rtl command has the following arguments:

<list-of-RTL-files>

Specifies space-separated name list of RTL files.

`-component`

(Optional) Specifies that RTL files being imported are for a component.

When this argument is used, the design RTL file is also imported as a component in GenSys with only information on ports/parameters. This flow is used to package RTL IP to MRS/IPXACT view.

If this argument is not specified, it is assumed that design RTL files are being imported.

When this argument is specified, component IP packaging occurs. For details, see *GenSys User Guide*.

`-import_glue`

(Optional) Specifies that along with port and connectivity information glue logic is also to be imported. By default, this argument is set to TRUE.

`-autosave`

(Optional) Specifies if the loaded design/component are to be saved in XML format. If this option is not used, the loaded design/component is not saved.

`-f <file-name-with-spyglass-options>`

(Optional) Specifies name of the file that contains the list of command-line options for SpyGlass. These are the same command-line options that were passed to SpyGlass to clean the design before loading RTL into GenSys.

This file can contain RTL files and options, such as `+incdir`, `-top`, `-stop`, `-y`, and `-v`.

`-hierarchy_view_name <hierarchy-modelView>`

(Optional) Sets the hierarchy model view.

This argument changes the model view for all the designs. For example, if you specify the value of this argument as `hier1`, GenSys sets the hierarchy model view as `hier1` for all the designs.

During RTL import, GenSys adds two model views namely "Hierarchy" and "Synthesis" to the component/design imported from the RTL view. This is as per IPXACT. This option changes "Hierarchy" to the name specified through this option.

`-hdl_view_name <synthesis-modelView>`

(Optional) Sets the hierarchy model view.

This argument changes the model view for all the designs. For example, if you specify the value of this argument as `hdl1`, GenSys sets the synthesis model view as `hdl1` for all the designs.

`-merge_views <TRUE | FALSE>`

(Optional) If the value of this argument is set to TRUE, then while loading the RTL of the specified design/sub-system, GenSys checks for an existing design/sub-system in the library that has the same VLVN information (or VLN information if the `-version` argument is not specified) of the object being loaded.

If GenSys finds an existing design/sub-system that has the same VLVN/VLN information of the design/sub-system being loaded, it copies the default values and interface-mapping of the existing design/sub-system to the new design/sub-system being loaded.

By default, the value of this argument is set to FALSE, and the `load_rtl` command continues its original behavior.

Note:

GenSys issues a warning and does not perform the merging if there is a mismatch between the ports and parameters of the existing object and new object being loaded. The "`-merge_views TRUE`" option is added for specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name of the design. If you do not specify this argument, GenSys considers the default value as novendor.

`-version <version-num> | <Tcl-function>`

(Optional) Specifies the version number of the design. If this argument is not specified, GenSys considers the default value as noversion.

If the `-import_all` argument is used, you should specify a Tcl function name in this argument. This Tcl function should return a new version number. For details, see [-import_all <TRUE | FALSE>](#).

RTL does not contain any vendor/version information. Therefore, after loading RTL, the design/component created have "noversion/novendor" set as version/vendor field. With these two options, you can choose to have the version/vendor information attached with the imported component/design. This is also applicable to the library as well.

`-library <library-name>`

(Optional) Specifies the library name of the design. If this argument is not specified, GenSys considers the default value as nolibrary.

`-import_all <TRUE | FALSE>`

(Optional) If this argument is set to TRUE, GenSys forcefully performs a clean import of an object even if an object of the same VLN (Vendor, Library, and Name) is already present in the Library Browser.

If you set this argument to TRUE, you must specify a Tcl function name in the `-version` argument of this Tcl command. This Tcl function should return a version number of the object being imported.

Given the VLN of the object being imported, all versions present for that VLN and version that is currently instantiated should be passed as arguments to this function.

Please note the following points:

- If you do not specify the `-version` argument, GenSys invokes the default Tcl function, `VLNV_next_version`.
- If you specify a Tcl function name in the `-version` argument but do not define that Tcl function, GenSys does not perform the RTL import process for that object and reports an error.
- If the specified Tcl function returns the same version as that of the existing object, GenSys does not perform the RTL import of that object and considers the object already present in the Library Browser.
- If the specified Tcl function returns a new version, GenSys performs the complete import of the RTL hierarchy of the object, and assigns the new version to that object.

Note:

If you specify the `-import_all` and `-merge_views TRUE` arguments together in the `load_rtl` Tcl command, the `-import_all` argument is given priority.

`-stop_level <int-value>`

(Optional) Sets a level till, that is total number of hierarchies till which RTL import should occur.

Consider that SoC RTL has 10 levels of hierarchies. If `-stop_level` is specified as 4, all subsystems/IPs lying below level 4 with reference to the top-level design is not imported.

`-ui`

(Optional) Displays the Specify Stop Modules dialog.

Use this dialog to specify sub-systems beyond which RTL import should not occur. For details on this dialog, refer to the *Configuring Sub-Systems as Stop Modules* topic of *GenSys User Guide*.

`-instance_threshold <value>`

(Optional) If the number of instances in the subsystem exceeds the value specified, the subsystem is converted into a black box and the hierarchy below the subsystem is ignored.

By default, the value is set to 100. Therefore, during the RTL import process, if the number of instances in a subsystem is more than 100, that subsystem is converted into a black box and the hierarchy below it is ignored.

Note:

The value specified for this argument overrides the value of the `instance_threshold` argument specified in the [set_gensys_options](#) Tcl command.

`-project <project-file-name>`

(Optional) Specifies a project file to be read by GenSys.

For information on project files, refer to the *Atrenta Console User Guide* of the SpyGlass help set.

`-auto_blackbox <TRUE | FALSE>`

(Optional) Specifies modules with unsupported constructs as black boxes.

There are several RTL constructs that are currently not supported by GenSys. RTL profiler in GenSys now detects such modules with unsupported constructs and generates a report. However, it is good practice to make such subsystems as black boxes. To enable this, the following option has been added to the `load_rtl` Tcl command:

`-auto_blackbox <TRUE | FALSE>`

The default value of this argument is FALSE.

When you set this argument to TRUE, GenSys sets the `-profile_rtl` argument of the `load_rtl` Tcl command to TRUE to enable the RTL profiler. After the RTL is loaded, GenSys identifies the modules that have unsupported constructs. These modules are marked as stops (`-stop` argument of `load_rtl` Tcl command) and the RTL is imported again.

A message on the Log Window indicates that the RTL is being imported again to implement the automatic black boxing feature.

Note:

To use this argument, make sure the `preserve_configuration` option is set to FALSE in the GenSys RTL import options. Otherwise, the run is aborted.

Returns

List of names of design/components loaded from the RTL files. This list can be used as shown in the following example:

```
set tlist [load_rtl -component module1.v master.v slave.v]
puts "no of element: [llength $tlist]\n"foreach item $tlist
{ puts "\nimported module:\t $item"}
```

Examples

The following example loads the RTL files for a component:

```
load_rtl com.v -interfaces=intf1.xml -component
```

The following example loads the RTL files for a design:

```
load_rtl dsgn.v
```

The following example loads the RTL files for a component and a design with their glue logic:

```
load_rtl comp2.v dsgn2.v -import_glue
```

make_rule

Create rules to match interfaces on RTL modules being imported

Description

The `make_rule` command create rules that are used to match interfaces on the RTL modules being imported.

Usage

```
make_rule
  -name <rule-name>
  [ -substr <num>:<num> ]*
  [ -prefix <str> -postfix <str> ]
  | [ -regexp <regular-expression> ]
```

Arguments

The `make_rule` command has the following arguments:

`-name <rule-name>`

Specifies the name of the rule to be created.

`-substr <num>:<num>`

(Optional) Specifies the subset of input string, which is used for interface matching.

A substring is created from the logical port name. For example, consider a logical port name, InterfacePort1. If you want to create a substring, IPort1, from this logical port name, specify the following:

```
-substr 0:0  
-substr 9:13
```

You can also use wildcards to extract substrings. The output of the above example can also be generated by using:

```
-substr 0:0  
-substr -5:$
```

In this case, \$ represents the last character of the logical name and -5 indicates the last five characters. The ^ character represents the first character.

```
-prefix <prefix> -postfix <postfix>
```

(Optional) Specifies the prefix and postfix string which is applied to the input string that will be matched with the actual port names or with the name of the interface library to be instantiated on the design/component.

```
-regexp <regular-expression>
```

(Optional) Specifies a Perl regular expression which will be applied to the input string used for matching actual port names or interface library names to be instantiated on a design/component.

This argument uses the substring specified by the -substr argument to create a regular expression. If you do not specify any substring, \$1 will be considered as the input.

Note:

You can either specify the -regexp argument or specify the -prefix and -postfix arguments.

Example

Consider an example in which an interface has a logical port, L1_atr_1. The actual port of this interface is A_L1_atr_1_port. Now, consider that you create the following rule:

```
make_rule -name R1 -regexp {A.*_$1.*$2.*} -substr ^:1 -substr 6:$
```

Note:

Here, \$1 means the first substring (-substr ^:1) and \$2 means the second substring (-substr 6:\$).

Now, when you apply the rule, R1, by using the [apply_rule](#) Tcl command, GenSys uses this rule to create a string from the logical port name. GenSys then maps those logical ports that

match the criteria specified by the R1 rule with their actual ports. For example, GenSys uses the rule, R1, to create a string by using the logical port name, L1_atr_1, in the following manner:

1. The -substr ^:1 argument creates a substring, L1.
2. The -substr 6:\$ argument creates a substring, _1.
3. The regular expression specified by -regexp {A.*_L1.*_1.*} argument creates a string, A.*_L1.*_1.* (. means any number of characters or no characters).

GenSys uses this input string (A.*_L1.*_1.*) to match with the actual port name, A_L1_atr_1_port. Since this input string (A.*_L1.*_1.*) matches with the actual port name (A_L1_atr_1_port), GenSys maps the logical port, L1_atr_1, with the actual port, A_L1_atr_1_port.

Similarly, the following example creates a rule, R2:

```
make_rule -name R2 -regexp {A.*_L1.*_1.*}
-substr ^:1 -substr -2:$
```

Here, -substr -2:\$ creates a substring by using the last two characters from the given input string.

match_interface

Specifies the properties of the interface being instantiated on a design/component after applying rule(s) on that interface

Usage

```
match_interface
{
  -component <design/component-name> }*
{
  -interface <interface-lib-name>
  [ -vendor <vendor> ]
  [ -library <library> ]
  [ -version <version> ] }*
[ -type <master | slave | system> ]
[ -mirrorcheck ]
-greedy <match-percentage>
-bestfit
-interface_naming <MATCHING_PATTERN_BASED | rule-name>
[ -case_insensitive_match <TRUE | FALSE> ]
```

Description

The `match_interface` command specifies the properties of the interface being instantiated on a design/component after applying rule(s) on that interface. For example, you can specify the type of interface (master, slave, system) to be considered while inferring interfaces on a component. You can also specify the name of the instance of the interface being instantiated on a component.

Arguments

The `match_interface` command has the following arguments:

`-component <design/component-name>`

Specifies the name of the design/component being imported through RTL import on which interfaces are being inferred.

Specify this argument multiple times if you want to specify more than one design/component.

`-interface <interface-lib-name>`

Specifies the name of the interface whose logical port names need to be matched.

Specify this argument multiple times if you want to specify more than one interface.

`-vendor <vendor>`

(Optional) Specifies the vendor of the interface library.

`version <version>`

(Optional) Specifies the version of the interface library.

`-library <library>`

(Optional) Specifies the library of the interface library.

`-type <master | slave | system>`

(Optional) Specifies the type of the interface to be matched.

By default, the value of this argument is set to `system`.

`-mirrorcheck`

(Optional) Specifies whether mirrored version should be used for the interfaces.

`-greedy <match-percentage>`

Specifies the match percentage. This argument accepts a decimal value in the range of 0.0 to 1.0.

GenSys generates a match percentage for each interface that needs to be instantiated on the specified design/component. GenSys calculates the match percentage in the following manner:

$$\text{Match percentage} = \frac{\text{Number of ports matched on an interface}}{\text{Total number of ports on an interface}}$$

GenSys then runs a loop in which it keeps on finding the interface whose match percentage is greater or equal to the criterion specified by the `-greedy` argument. Once it finds such an interface, it exits the loop, and instantiates that interface on the specified design/component.

`-bestfit>`

Instantiates the interface that has the largest number of matching ports. For example, consider two interfaces, Intf1 and Intf2. For the interface, Intf1, consider that 9 out of 10 ports are matching. For the interface, Intf2, consider that 11 out of 20 ports are matching. In this case, GenSys instantiates the interface, Intf2, since it has the largest number of matching ports.

GenSys considers the `-bestfit` argument, by default, if you do not specify the `-greedy` argument.

`-interface_naming <MATCHING_PATTERN_BASED | rule-name>`

You can either specify:

- `<rule-name>`: The name of the rule created by the [make_rule](#) Tcl command. GenSys uses this rule on interface library names.
- `MATCHING_PATTERN_BASED`: GenSys applies the rule, which was used to match all logicals of an interface with all corresponding physical ports, on the interface library name and the resultant value is the interface class name. This results in more explanatory interface class names. See [Example 2](#).

`-case_insensitive_match <TRUE | FALSE>`

Specifies whether the matching pattern should be case sensitive. By default, this argument is set to `FALSE`, and the interface matching pattern is case sensitive.

Examples

Example 1

Consider the following Tcl command:

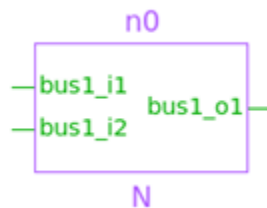
```
match_interface -component comp -interface intf2
               -interface intf1 -greedy .9
```

This command instantiates that interface among intf1 and intf2 whose match percentage is greater or equal to the greedy criterion, .9.

Example 2

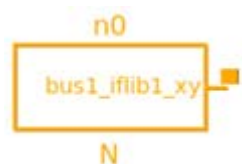
Consider a component N shown below, with the following physical ports bus1_i1, bus1_i2, and bus1_o1.

An interface library named iflib1_xy has the following logical ports: i1, i2, and o1.



While inferring interfaces, you can name the generated interfaces to include bus1 in the interface name:

```
make_rule -name r1 -regexp {.*_$1}
apply_rule -rule r1 -interface iflib1_xy -component N
match_interface -interface iflib1_xy -component N
               -interface_naming MATCHING_PATTERN_BASED
infer_interfaces
```



The interface iflib1_xy is matched with ports of N with rule r1. Therefore, the interface name is generated by applying the matched rule r1 against the interface library name. Any .* is replaced by the captured pattern in the port matching stage, bus1, and \$1 is replaced by interface library name.

Similarly, if a rule has prefix and postfix pattern, the same pattern is applied to the interface library name and if that rule is used to infer interface on a component.

mod_sch_print

Prints the modular schematic

Usage

```
mod_sch_print
[ -prn_cmd <command> ]
[ -file <file-name> ]
[ -color_mode <mono | color | colorinv> ]
[ -orient <auto | landscape | portrait> ]
[ -sheet_size <size> ]
[ -scale <full | fullfit | current> ]
[ -with_highlight ]
```

Description

The mod_sch_print command prints the modular schematic.

Note:

This command can be used only if the -batchSch command-line option is specified while running GenSys in the batch mode.

Arguments

The mod_sch_print command has the following arguments:

-prn_cmd <command>

(Optional) Specifies the print command of the printer. For example, the print command for Linux platform is lpr.

-file <file-name>

(Optional) Specifies the name of the file.

`-color_mode`

(Optional) Specifies the color mode of the page. This argument accepts one of the following values:

mono, color, or colorinv

`-orient`

(Optional) Specifies the page orientation. This argument accepts one of the following values:

auto, landscape, or portrait

`-sheet_size <size>`

(Optional) Specifies the size of the sheet. You can specify any of the following values:

A	A	A	A	A	B	C	D	E	F	G
4	1	2	3							

`-scale`

(Optional) Specifies the scale of the page. You can specify any of the following values:

Value	Purpose
full	Prints/saves the full schematic at the current scale. GenSys considers the previously set scale as the current scale in this case. For example, the current scale set by using the <code>-factor</code> argument of the preceding mod_sch_set_zoom_factor Tcl command is used.
fullfit	Prints/saves the full schematic ignoring the current scale
current	Prints the current window on the currently scaled schematic

`-with_highlight`

(Optional) Enables the printing of highlight information. By default, the highlighted objects are printed in the normal mode.

mod_sch_set_plane_view

Sets the plane view of the Modular Schematic

Usage

```
mod_sch_set_plane_view
    -plane_name <plane-name> <true | false>
```

Description

The `mod_sch_set_plane_view` command specifies whether the connections of the specified plane should be visible in the printed modular schematic.

Note:

This command can be used only if the `-batchSch` command-line option is specified while running GenSys in the batch mode.

Note:

This command is effective only when schematic has been loaded.

Arguments

The `mod_sch_set_plane_view` command has the following arguments:

```
-plane_name <plane-name> <true | false>
```

Specifies whether the connections of the specified plane should be visible in the Modular Schematic window. If this argument is set to false, the connections of the specified plane gets hidden.

mod_sch_set_zoom_factor

Sets the zoom factor of the modular schematic

Usage

```
mod_sch_set_zoom_factor
    -factor <value> | -fullfit
```

Description

The `mod_sch_set_zoom_factor` command sets the zoom factor of the printed modular schematic.

Note:

While running GenSys in the batch mode, this command can be used only if the `-batchSch` command-line option is specified

Arguments

The `mod_sch_set_zoom_factor` command has the following arguments:

`-factor <value>`

Specifies the zoom factor. This argument accepts an integer value.

The specified zoom factor is absolute (and not relative). If you specify `mod_sch_set_zoom_factor -factor 0.5` multiple times, only the first specified command changes the size. In contrast, the north-east upstroke, tagged as zoom 0.5, in the schematic is relative to the current size.

`-fullfit`

Fits the entire schematic view in the visible area of the modular schematic.

Note:

You must specify either `-factor` or `-fullfit` argument with this Tcl command. If you specify both these arguments, GenSys considers the `-fullfit` argument.

mod_sch_show

Populates the modular schematic

Usage

```
mod_sch_show
  -design <des-name>
  | -component <comp-name>
```

Description

The `mod_sch_show` command populates the modular schematic, to be printed, with the specified design or component. This command can be specified in the batch mode.

Note:

This command can be used only if the `-batchSch` command-line option is specified while running GenSys in the batch mode.

Arguments

The `mod_sch_show` command has the following arguments:

`-design <des-name>`

Specifies the name of the design for which the modular schematic view needs to be displayed.

`-component <comp-name>`

Specifies the name of the component for which the modular schematic needs to be printed.

move_instance

Moves the specified instance across the hierarchy

Usage

```
move_instance
  -instance <instance-name>
  -from <source-subsystem>
  -to <destination-subsystem>
  [ -remove_floating_ports <TRUE | FALSE> ]
  [ -port_naming_subroutine <Perl-function> ]
  [ -preserve_parameter <TRUE | FALSE> ]
```

Description

The `move_instance` command moves the specified instance from the source hierarchy to the destination hierarchy. In addition, this Tcl command restores the connections of the specified instance by moving those connections along with the instance into the destination hierarchy. This might lead to the creation of some intermediate ports, if required.

Note the following points:

- On moving instances containing interface connections, GenSys converts all the interface connections associated with that instance to ad-hoc connections.

- Modifying hierarchy by using this command results in creation of ports in intermediate hierarchies for connectivity rerouting.

Use the [set_default_port_name](#) command to set a default naming convention for such ports.

These ports are tracked internally within GenSys and automatically removed if further hierarchy modifications makes these ports as floating.

Arguments

The `move_instance` command has the following arguments:

`-instance <instance-name>`

Specifies the name of the instance that is to be moved.

`-from <source-subsystem>`

Specifies the hierarchical name of the subsystem which contains the instance to be moved.

This hierarchical path should be with reference to the top-level design name.

`-to <destination-subsystem>`

Specifies the hierarchical name of the destination subsystem to which the instance needs to be moved.

This hierarchical path should be with reference to the top-level design name.

`-remove_floating_ports <TRUE | FALSE>`

(Optional) Removes hanging or redundant ports that are generated because of moving IPs.

Set this argument to `FALSE` to retain such ports.

When this option is set to `TRUE`, GenSys also removes user ports that become redundant, i.e. floating after the instance is moved. Otherwise, only GenSys generated internal floating ports are removed after instance move.

`-port_naming_subroutine <Perl-function>`

(Optional) Specifies a port-naming convention for the ports generated by the `move_instance` Tcl command. To provide a port-naming convention, write a Perl file containing a Perl subroutine, which is specified to this argument. However, it is recommended to use the [set_default_port_name](#) Tcl command to control port names.

Inputs of the subroutine: Two adhoc connection type items (*adhoc1*, *ahhoc2*)

Output of the subroutine: Port name

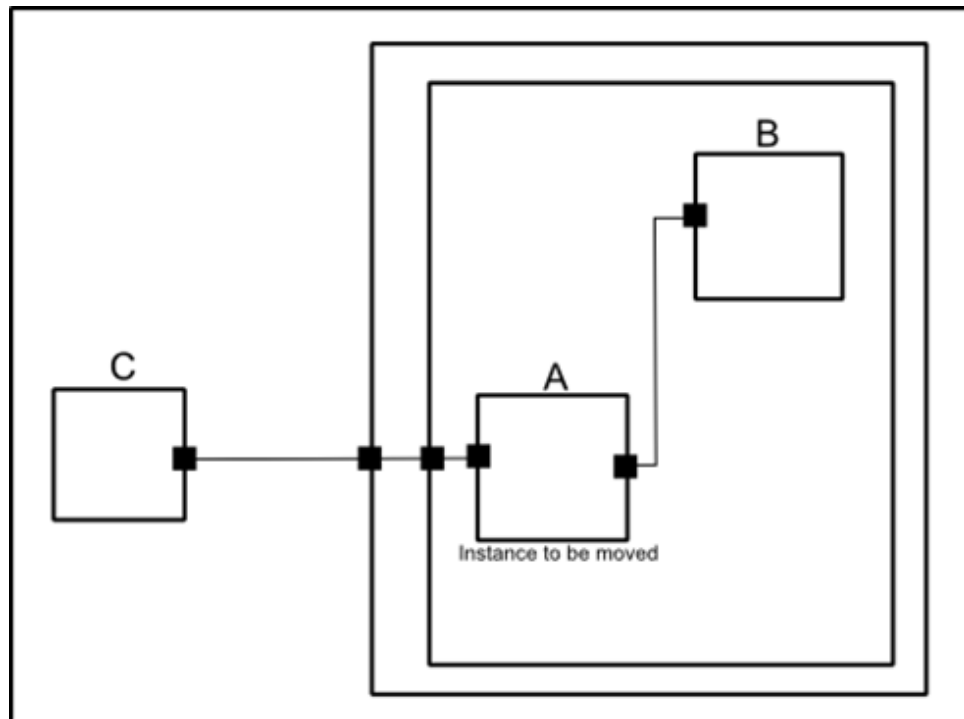
While invoking Gensys, specify that Perl file by using the gensys -I command. Then, specify the subroutine name to the -port_naming_subroutine argument of the move_instance Tcl command.

See [Example 3](#).

Note that the specified port-naming convention is applicable for the connections going from the pin of the instance to be moved to the pin of an instance at the same level or a level below in the design hierarchy.

However, the specified port-naming convention is not applicable for the connections going out through the parent design boundary of the moving instance to the pin/port of an instance/design that is at a higher level than the level of the instance to be moved.

For example, in the following scenario, the port-naming convention is applicable for the connections between A and B, but not between A and C:



`-preserve_parameter <TRUE | FALSE>`

(Optional) Specifies whether to preserve parameterized expressions of the pins of the instance being moved.

By default, this argument is set to FALSE.

When you set this argument to TRUE, parameterized expressions are preserved in terms of the diverge point of the movement. Only the expressions containing parameters that are controlled from the diverge point are preserved. To preserve parameters in the destination side, new parameters are added to the destination side.

See [Example 4](#).

Examples

Example 1

Consider the following Tcl command:

```
move_instance -instance i1 -from TOP::h1 -to TOP
```

The above command moves the instance, i1, from the source subsystem, TOP.h1, to the destination subsystem, TOP.

Example 2

Consider a Tcl file containing the following Tcl commands:

```
new component c1
add_port -name p1 -direction IN
  add_port -name p2 -direction OUT
add_port -name p3 -direction INOUT
save
new component c2
add_port -name p1 -direction OUT
add_port -name p2 -direction IN
add_port -name p3 -direction INOUT
save
new design H
add_port -name hp1 -direction IN
add_port -name hp2 -direction OUT
add_port -name hp3 -direction INOUT
add_instance -name i1 -master c1
add_connection -instance i1 -pin p1 -port hp1
add_connection -instance i1 -pin p2 -port hp2
add_connection -instance i1 -pin p3 -port hp3
```

```

save
new design TOP
add_instance -name h1 -master H
add_instance -name i2 -master c2
add_connection -instance i2 -pin p1 -instance h1 -pin
hp1add_connection -instance i2 -pin p2 -instance h1 -pin
hp2add_connection -instance i2 -pin p3 -instance h1 -pin
hp3
save
move_instance -instance i1 -from TOP::h1 -to TOP

```

When you run the above commands, GenSys results in the following final connections at the TOP level:

```

add_connection -instance i2 -pin p1 -instance i1
-pin p1
add_connection -instance i2 -pin p2 -instance i1
-pin p2
add_connection -instance i2 -pin p3 -instance i1
-pin p3

```

Example 3

Consider the following Perl file that contains the `get_port_name` subroutine to define a port-naming convention:

```

use genesisapis;
sub get_port_name
{
    my $adhoc1 = shift;
    my $adhoc2 = shift;
    -----
    return port_name;
}
1;

```

After invoking GenSys, specify the following Tcl command to implement the subroutine-specified port-naming convention while moving instances:

```

move_instance -instance inst -from top
-to top::midInst::botInst
-port_naming_subroutine get_port_name

```

Example 4

This example shows the impact of setting the `-preserve_parameter` argument to `TRUE`. Consider the following move operation to the design shown in the following figure.

```

move_instance
-instance p

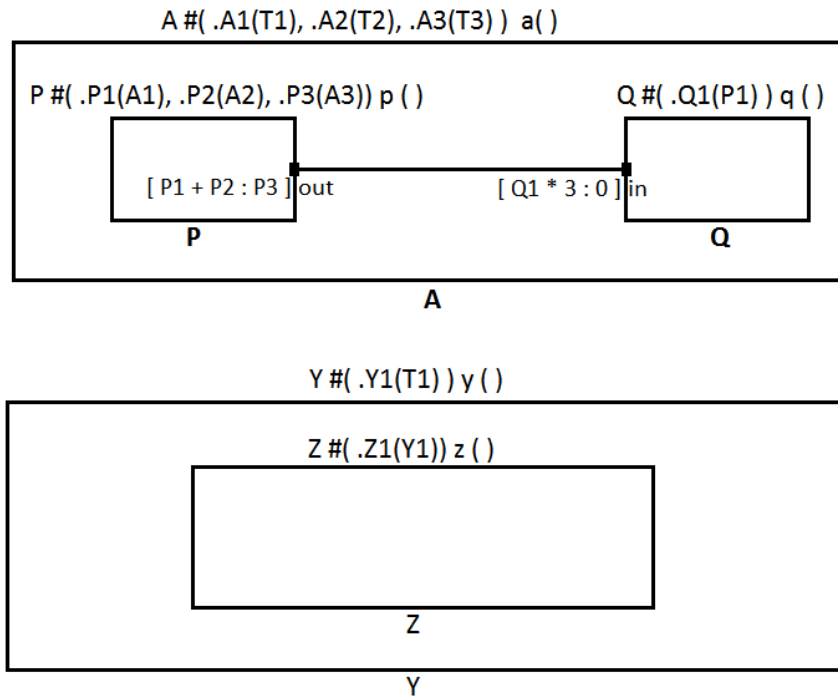
```



```
-from top::a -to top::y::z
-preserve_parameter TRUE
```

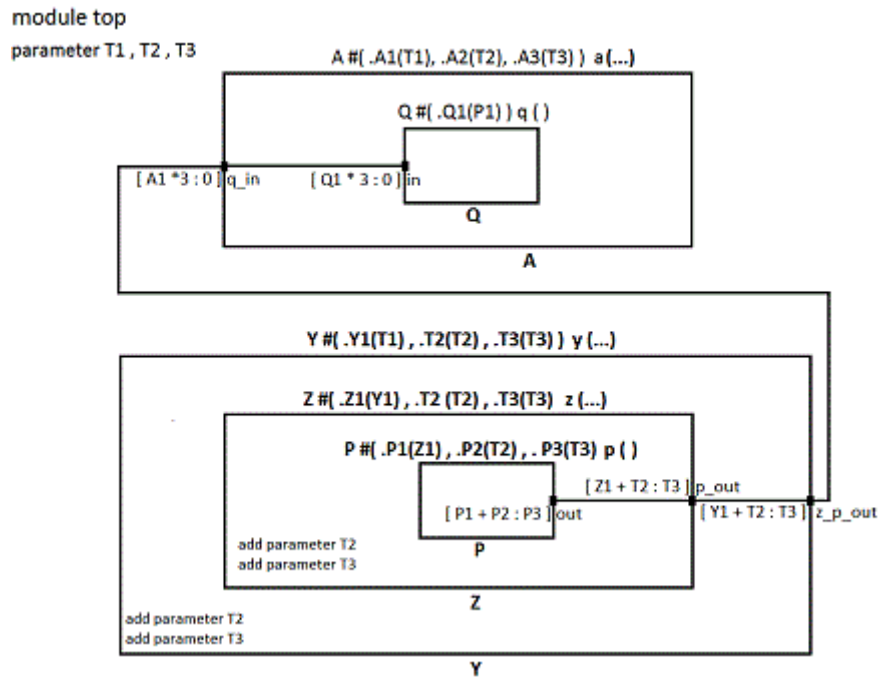
module top

parameter T1 , T2 , T3



The common point of the movement is top and is defined as the diverge point. This diverge point is reference point while preserving parameters. The figure below shows the impact of running the Tcl command:

- Ports Created on Source Side: port `q_in` on design A
- Ports Created on Destination Side: Port `p_out` on design Z and port `z_p_out` on design Y
- Reused Parameters: Parameter `A1` in design A , `Y1` in design Y, and `Z1` in design Z
- New Parameters Added to Preserve Parameters: Parameters `T2`, `T3` to design Y, and parameters `T2`, `T3` to design Z
- Parameters Overridden: Finally, the parameters of the moved instance are overridden by the corresponding parameters of the destination design.



new

Creates a new design, interface, component, or design configuration file

Usage

```

new design <name>
  | interface <name>
  | component <name>
  | designconfiguration <name>
[ -description <description> ]
[ -short_name <short-name> ]
[ -vendor_name <vendor-name> ]
[ -version_num <version-num> ]
[ -library_name <library-name> ]
[ -componentType <comp-type> ]
[ -visibility <visibility-condition> ]

```

Description

The new command creates a new design, component, interface, or design configuration MRS file.

The new object is loaded and becomes the active object.

The object's MRS file is created as *<name>.xml* in the current working directory. The specified object name *<name>* is also added in the Name::name column of the new object's table.

Arguments

The new command has the following arguments:

`design <name>`

Specifies the name of the new design.

`interface <name>`

Specifies the name of the new interface.

`component <name>`

Specifies the name of the new component.

`designconfiguration <name>`

Specifies the name of the new design configuration.

`-short_name <short-name>`

(Optional) Specifies the short name of the new object.

`-description <description>`

(Optional) Specifies the description of the new object.

This argument also accepts unicode characters.

`-vendor_name <vendor-name>`

(Optional) Specifies the vendor name for the new object.

`-version_num <version-num>`

(Optional) Specifies the version number of the new object.

`-library_name <library-name>`

(Optional) Specifies the name of the library of the new object.

`-componentType <comp-type>`

The `-componentType` option is applicable to components and designs only.

(Optional) Specifies the component type of the new component as one of the following:

BUSLOGI C	CPU	MEMORY
OTHERS	PERIPHER AL	SUBSYST EM
GLUE		

Note:

The `-componentType` argument internally sets the `comp_type` attribute on that component or design. The `comp_type` attribute is an internally generated reserved attribute by GenSys, which is used to set the component type.

Note:

If you add an attribute `comp_type` to a component/design using the [add_config](#) command, the value of `comp_type` attribute will be set as the component type for that component/design. In such a case, a warning is also displayed.

Note:

If a value for `comp_type` has already been set, it should be overridden by the new value.

```
new component mypad -componentType PAD_CELL
```

The above command sets the component type and `comp_type` attribute to `PAD_CELL`.

```
add_attribute -name comp_type -value "MUX_CELL"
```

The above command resets the component type and `comp_type` attribute to `MUX_CELL` and a warning is displayed.

```
add_attribute -name comp_type -value "PAD_CELL "
```

The above command resets the component type and `comp_type` attribute to `PAD_CELL` and a warning is displayed.

`-visibility <visibility-condition>`

(Optional) Specifies the comma-separated list of conditions under which the object would be visible to the generators.

Examples

1. The following example creates a new component, abc:

```
new component abc
```

2. The following example creates a component (with the allow_create option set to Y), comp1, with one vector port:

```
new component comp1 -allow_create Y
add_port -name pin1 -msb 4 -lsb 0
save
```

3. The following example creates a design, des, and instantiates a component whose allow_create option set to Y:

```
new component comp -allow_create Y
new design des
add_port -name P1 -lsb 0 -msb 0
add_instance -name inst1 -master comp
add_connection -instance I1 -pin R1 -port P1
....
saveall
```

While creating a design using the new command, the saveall command should be used instead of the save command to save all the components along with the design. This is required because if the allow_create option for the components is set to Y then the number of ports on them may have been changed with the addition of new connections in the design. Therefore, it is required to save the changes in the master components along with the design.

optimize_ports

Optimizes ports generated on subsystem boundaries

Usage

```
optimize_ports
[-merge <TRUE | FALSE>
[-compress <FALSE | TRUE>
[-split <FALSE | TRUE>
```

Description

The `optimize_ports` command optimizes ports generated on subsystem boundaries as part of various hierarchy manipulation and connection creation operations. For more information, see the following sections:

- [Prerequisite](#)
- [Type of Optimization](#)
- [Exceptions](#)
- [Arguments](#)
- [Examples](#)

Prerequisite

Before you use this Tcl command, make sure the ports are generated by following four hierarchy manipulation Tcl commands:

- [group_design](#)
- [move_instance](#)
- [reroute_connection](#)
- [add_connection](#) (hierarchical connections)

Type of Optimization

You can perform the following optimizations on GenSys generated ports:

- **Merge:** Sink ports that are connected to a common driver outside or inside the boundary are merged so that only one port is connected to the common driver.

Default behavior is to select one of the port names arbitrarily. Naming of the merged port will honor naming set by [set_default_port_name](#) command, only if it is a driver-based expression.

Use the `merge <TRUE | FALSE>` argument to implement this type of optimization.

Also see [Example 1 - Merge](#).

- **Compress:** Vector ports might have holes, such as bits unconnected from both internally to the boundary and externally. These holes are removed in this optimization and the port size is reduced.

Use the `compress <FALSE | TRUE>` argument to implement this type of optimization.

Also see [Example 2 - Compress and Split](#).

- **Split:** A vector port is split into multiple vector/scalar ports when there are holes in it. For example, if there are two holes in a vector port, three new ports are created to replace the original port.

Naming for new ports generated by split can be controlled by the expression given to `set_default_port_name`, only if the expression is sink-based for driver ports, or driver-based for sink ports.

Use the `split <FALSE | TRUE>` argument to implement this type of optimization.

Also see [Example 2 - Compress and Split](#).

Exceptions

The following types of ports are not fully optimizable: Parameterized, complex SV, and interface. In addition, these ports cannot be compressed or split.

Arguments

The `optimize_ports` command has the following arguments:

`merge <TRUE | FALSE>`

(Optional) Specifies the merge optimization.

Default is TRUE and merge optimization is performed.

For merging to be successful, make sure the two ports have the same complex/SV type.

You cannot specify this argument with the split argument.

Also see [Example 1 - Merge](#).

`compress <FALSE | TRUE>`

(Optional) Specifies the compress optimization.

Default is FALSE and compress optimization is performed. To enable it, set this argument to TRUE.

You cannot specify this argument with the split argument.

Also see [Example 2 - Compress and Split](#).

```
split <FALSE | TRUE>
```

(Optional) Specifies the split optimization.

Default is FALSE and compress optimization is performed. To enable it, set this argument to TRUE.

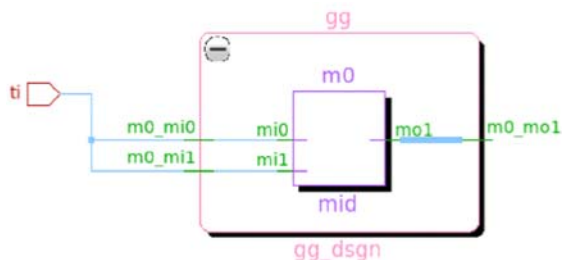
Also see [Example 2 - Compress and Split](#).

Examples

Example 1 - Merge

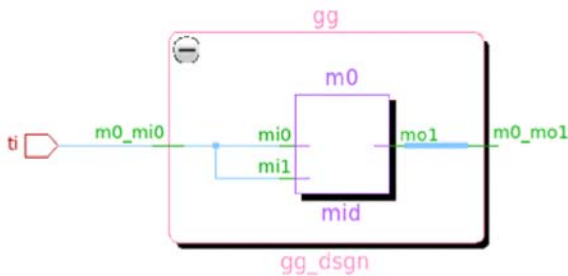
This example shows the impact of the merge argument.

A [group_design](#) operation was performed on instance m0. Pins m0_mi0 and m0_mi1 are driven by port ti.



After running the following command, both inputs are merged to pin m0_mi0, and m0_mi1 has been removed.

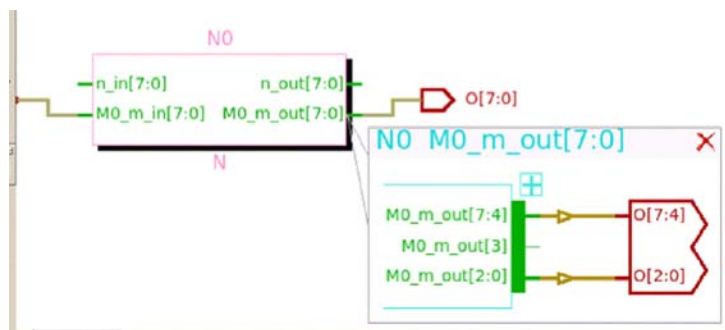
```
optimize_ports -merge true
```

Example 2 - Compress and Split

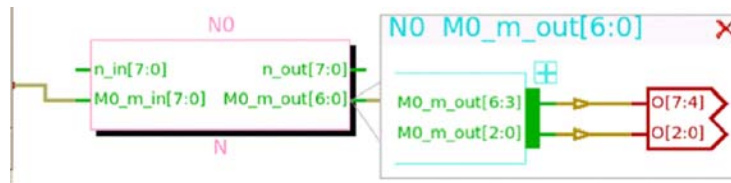
This example shows the impact of the compress argument.

In this example, port M0_m_out[7:0] was created by [add_connection](#) with hole in bit 3. A hole gets created due to unconnected bits. For example, consider a vector port of five bits. Now if you connect two lower bits and two upper bits of this port and then run the [set_group_options](#) Tcl command, the middle bit remains unconnected resulting in a hole."

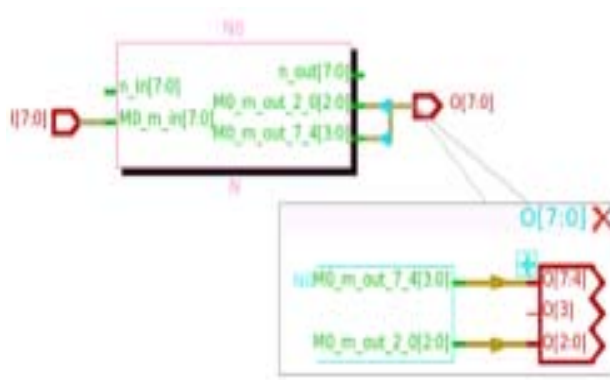


After running the following command, the reduced port and connectivity is seen in the following figure.

```
optimize_ports -compress true
```



Similarly, the impact of the split argument is that the original vector port is split into two new ports, as shown in the following figure:



pop_default_datasource

Removes the top element from the default datasource stack

Usage

```
pop_default_datasource
```

Description

The `pop_default_datasource` command removes the top element from the default datasource stack.

print_to_perf_log

Prints the given details to gensys.perflog.

Usage

```
print_to_perf_log  
[ -text <text> ]  
[ -time_mem ]
```

Description

The `print_to_perf_log` command prints the given message and/or time and memory details to the file, `gensys.perflog`, in the log directory.

You can choose to print a string or elapsed time and memory usage details or both.

Note:

The `gensys.perflog` file is created when GenSys is run with `--perflog` command-line option.

Arguments

The `print_to_perf_log` command has the following arguments:

`-text <text>`

(Optional) Specifies the text message to be printed to `gensys.perflog`.

`-time_mem`

(Optional) Specifies that time and memory details need to be printed to `gensys.perflog`. Time is the time elapsed since GenSys started and memory usage is memory used (in terms of megabytes) at a given point of time.

Examples

The following example prints a message:

```
print_to_perf_log -text 'This is some text\n'
```

The following example prints time and memory details:

```
print_to_perf_log -time_mem
```

The following example prints both a given message and time and memory details:

```
print_to_perf_log -text 'This is some text\n' -time_mem
```

push_default_datasource

Pushes the default datasource information on the stack

Usage

```
push_default_datasource
[ -personname <name> ]
[ -changetype <type> ]
[ -changereason <reason> ]
[ -changemethod <method> ]
[ -changedate <date> ]
[ -datafrozen <TRUE | FALSE> ]
[ -creationdate <date> ]
```

Description

The `push_default_datasource` command pushes the default datasource information on the stack. GenSys maintains this stack to keep the default datasources information. The datasource, which is on the top of the stack, is used in the current generator run.

If you want to use your own datasource information for a particular generator run, you first need to create your own generator specific datasource information and then push it on the stack by using the `push_default_datasource` Tcl command. Then, execute the generator with your own datasource information. After execution of the generator, pop your own datasource information from the stack by using the [pop_default_datasource](#) Tcl command. This way, the use of the previous datasource is continued for the rest of the things.

Arguments

The command has the following arguments:

`-personname <name>`

(Optional) Specifies the datasource name.

`-changetype <type>`

(Optional) Specifies the type of datasource.

`-changereason <reason>`

(Optional) Specifies the reason for making changes in the datasource.

`-changemethod <method>`

(Optional) Specifies the method that make the change.

`-changedate <date>`

(Optional) Specifies the date on which the change occurs.

`-datafrozen <TRUE | FALSE>`

(Optional) Specifies whether the changes made should be frozen.

`-creationdate <date>`

(Optional) Specifies the date on which the datasource gets created. If you do not specify the creation date then the current system date is considered.

refresh_library

Refreshes the library browser.

Usage

```
refresh_library
```

Description

The `refresh_library` command refreshes the library browser.

You can use this command to refresh the library browser after deleting/adding files from/to the directory.

refresh_table

Refreshes the specified table

Usage

```
refresh_table <hierarchical-table-name>
```

Description

The `refresh_table` command refreshes the specified table. This command accepts the hierarchical name of the table that needs to be refreshed.

Consider the following example:

```
new design d
add_port -name p1
refresh_table design.Ports
```

The above command refreshes the Ports table of the design, d.

Consider another example in which you have attached your own schema with the design. Now, the design has a table, PRP, that is coming from the schema provided by you. In this case, you can refresh that table by specifying the following command:

```
refresh_table design.PRP
```

refresh_ui

Refreshes the GUI

Usage

```
refresh_ui
```

Description

The `refresh_ui` command refreshes the GUI.

This command works even if the `-incremental_refresh` argument of the [set_ui_options](#) command is set to false.

rename

Renames the specified design or component

Usage

```
rename <object-name> -as <new-name>
```

Description

The rename command renames the specified design or component.

Arguments

The rename command has the following arguments:

<object-name>

Specifies the name of the design or component to be renamed.

-as <new-name>

Specifies the new name of the object.

Examples

The following example renames the comp3 component to COMP3:

```
rename comp3 -as COMP3
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/examples/Genesis/case3 directory. The rename command works only for component and design.

rename_instance

Renames a component or subsystem instance present in the active design

Usage

```
rename_instance -name <current-name-of-instance> -newname  
<new-name-of-instance>
```

Description

The rename_instance command renames the specified instance present in the active design.

Arguments

The `rename_instance` command has the following arguments:

`-name <current-name-of-instance>`

Specifies the current name of the component or subsystem instance being renamed.

`-new_name <new-name-of-instance>`

Specifies the new name for the component or subsystem instance being renamed.

Examples

The following example renames instance I1 of design D with I1_NEW:

```
select_design D
rename_instance -name I1 -new_name I1_NEW
```

reroute_connection

Reroutes existing connections through new instances or instance ports

Usage

```
reroute_connection
  -source <source-hier-path>
  -dest <dest-hier-path>
  -through <hier-inst-names/instance-ports>
  [ -intf_group {<group-name-for-interface-rerouting>} ]
  [ -exclude_src <source-pins-to-be-excluded> ]
  [ -exclude_dest <destination-pins-to-be-excluded> ]
  [ -source_port <TRUE | FALSE> ]
  [ -dest_port <TRUE | FALSE> ]
  [ -all <TRUE | FALSE> ]
```

Description

The `reroute_connection` command reroutes existing connections through new instances/ports while reusing the existing port names as much as possible.

For details on rerouting connections, see [Examples of Rerouting GenSys Connections through Via Points](#).

Arguments

The `reroute_connection` command has the following arguments:

`-source <source-hier-path>`

Specifies the hierarchical path of the source instance pin/port/interface with reference to the top design.

`-dest <sink-hier-path>`

Specifies the hierarchical path of the destination instance pin/port/interface with reference to the top design.

`-through <hier-list-names/instance-ports>`

Specifies a space-separated list of hierarchical instance names/ports with reference to the top design.

`-intf_group {<group-name-for-interface-rerouting>`

(Optional) Specifies the group name for [Rerouting SystemVerilog Interfaces](#).

`-exclude_src <source-pins-to-be-excluded>`

(Optional) Specifies a space-separated list of source pins that should be ignored while rerouting connections. See [Example 10 - Rerouting Connections After Specifying Excluded Source and Destination Pins](#).

`-exclude_dest <destination-pins-to-be-excluded>`

(Optional) Specifies a space-separated list of destination pins that should be ignored while rerouting connections. See [Example 10 - Rerouting Connections After Specifying Excluded Source and Destination Pins](#).

`-source_port <TRUE | FALSE>`

(Optional) Set this argument to `TRUE` so that rerouting does not occur through the pins specified by the `-source` argument. Rerouting occurs through the corresponding component of the instance specified to the `-source` argument. That is, rerouting occurs through the design ports of the component.

`-dest_port <TRUE | FALSE>`

(Optional) Set this argument to TRUE so that rerouting does not occur through the pins specified by the `-dest` argument. Rerouting occurs through the corresponding component of the instance specified to the `-dest` argument. That is, rerouting occurs through the design ports of the component.

`-all <TRUE | FALSE>`

(Optional) When this option is specified all connections between source and destination instances are rerouted by the value specified in the `-through` argument. See [Example 1](#).

Examples

Refer to the [Examples of Rerouting GenSys Connections through Via Points](#).

Example 1

To reroute connections between the `ufpb_grp_wrap` and `ufrs_grp_wrap` instances through the `feedthru_inst` instance of the top-level design `cpu`, specify the following commands:

```
new design feedthru_master
select_design cpu #cpu is the top level design
add_instance -name feedthru_inst -master feedthru_master
reroute_connection -source {cpu.ufrs_grp_wrap} -dest {cpu.ufpb_grp_wrap}
-through {cpu.feedthru_inst} -all TRUE
```

Note:

Make sure you specify the complete hierarchical path from the top-level design for source, destination and feedthrough instances.

reroute_fanout_node

Dissolves the specified output pin by rerouting connections internal to a design

Usage

```
reroute_fanout_node
  -source <output-pin>
  [ -dest <hierarchical-dest> ]
```

Description

The `reroute_fanout_node` command dissolves the specified output pin and reroutes connections so that the fan-out nets are rerouted internally to the sub-system and ports are formed for each fan-out net.

For details on rerouting connections, see [Rerouting GenSys Connections through Via Points](#).

Arguments

The `reroute_fanout_node` command has the following arguments:

`-source <output-pin>`

Specifies the output terminal that should be dissolved.

`-dest <hierarchical-dest>`

(Optional) Specifies the sink of the fan-out net that needs to be rerouted.

Examples

See [Example 3 - Using the `reroute_fanout_node` Tcl Command](#) (usage of the `reroute_fanout_node` command with the `-dest` argument) and [Example 4 - `reroute_fanout_node` Tcl Command without `-dest` Argument](#) (usage of the `reroute_fanout_node` command without the `-dest` argument) of [Examples of Rerouting GenSys Connections through Via Points](#).

reset_rules

Resets all the rules earlier set on the specified components for interface matching

Usage

```
reset_rules
[ {-component <component-name>}* ]
```

Description

The `reset_rules` command resets all the rules that were earlier set on the specified component(s) being imported through RTL import.

Arguments

The `reset_rules` command has the following arguments:

`-component <component-name>`

(Optional) Specifies the name of the component on which you want to reset all the earlier specified rules.

You can specify this argument multiple times if you want to reset rules on more than one component.

If you do not specify this argument, GenSys resets the previously set rules on all the components.

`reset_visibility`

Resets the visibility of all objects (registers/bitfields and ports) to true

Usage

```
reset_visibility
```

Description

The `reset_visibility` command resets the visibility of all those objects (registers/bitfields and ports) to true.

Executing this command sets the initial conditions for the new generator.

`restore_session`

Restores a GenSys session

Usage

```
restore_session  
  -state <dir-name>
```

Description

The `restore_session` command restores a session saved by the [save_session](#) command.

When you run this Tcl command, GenSys closes the current session and loads the session from the directory specified by this command.

All unsaved changes of the current session are lost while restoring/loading another session.

Arguments

The `restore_session` command has the following arguments:

```
-state <dir-name>
```

Specifies the name of the directory under which a session was restored by using the [save_session](#) command.

Example

The commands specified in the following steps explain the purpose of the `save_session` command:

1. Creating components and instantiating them in a design

```
new component comp1 -version_num 1.0  
-vendor_name Atrenta -library_name lib1 add_port -name p1  
-direction OUT -lsb 0 -msb 10  
new component comp2 -vendor_name atrenta  
-version_num 2.0 -library_name lib1 add_port -name p2 -direction IN  
-lsb 0 -msb 10  
new design d  
add_instance -name inst1 -master comp1  
add_instance -name inst2 -master comp2 -vendor atrenta -version 2.0  
-library lib1
```

2. Saving the session of step 1

```
save_session -state first_session
```

In this step, GenSys saves the current session at the following path:

<current-working-directory>/gensys_dir/gensys_states/first_session

3. Saving the session of step 2

```
add_instance -name inst4 -master comp1
delete_instance -name inst1

save_session -state second_session
```

In this step, GenSys saves the current session in the `second_session` directory with latest changes, such as addition and deletion of instances. This directory is saved under the current working directory.

Now, the following directory structures are present:

```
<current-working-directory>/gensys_dir/gensys_states/first_session
<current-working-directory>/gensys_dir/gensys_states/second_session
```

4. Restoring a saved session

```
add_port -name prt1
restore_session -state first_session
```

In this step, GenSys loads (restores) the session saved in the `first_session` directory.

All unsaved changes, such as addition of the `prt` port in the current session are lost before restoring the session saved in the `first_session` directory.

5. Overwriting a saved session

```
add_instance -name inst5 -master comp2
save_session -state first_session -overwrite
```

In this step, GenSys saves the current session in the `first_session` directory with the latest changes, such as addition of the `inst5` instance.

In this case, the `first_session` directory containing the session saved in the `step2` is overwritten with the latest changes in the current session.

restruct_rtl

Modifies design hierarchy

Usage

```
restruct_rtl
  -file <file-name>
  [ -ui ]
```

Description

The `restruct_rtl` command modifies the design hierarchy by running the file containing the hierarchy manipulation Tcl commands, such as `move_instance`, `group_design`, `uniquify_instance`, and `clone_ip`.

Arguments

The `restruct_rtl` command has the following arguments:

`-file <file-name>`

Specifies the file containing hierarchy manipulation Tcl commands.

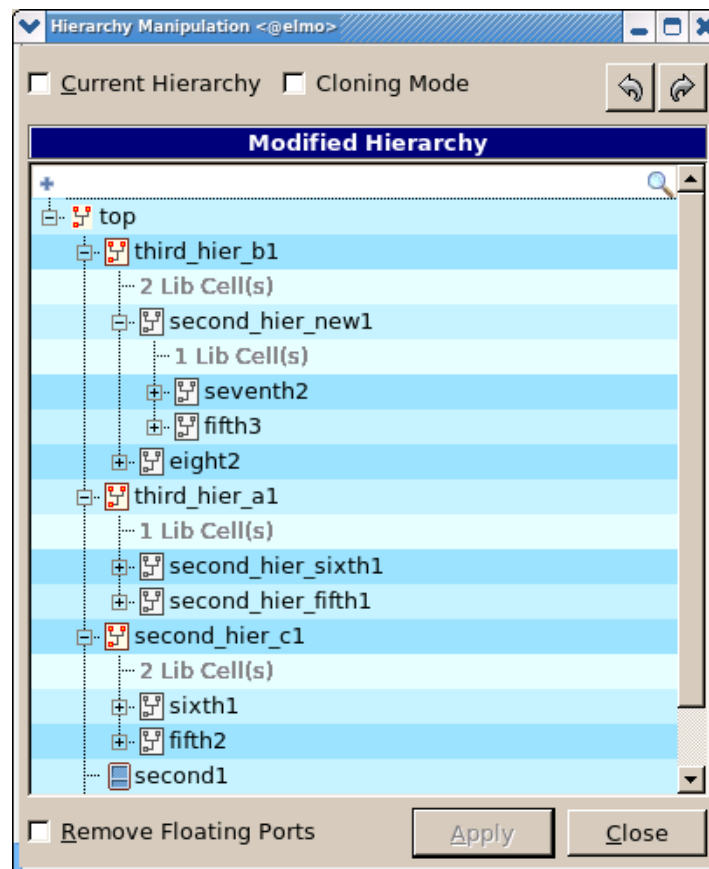
`-ui`

(Optional) Opens the Hierarchy Manipulation widget showing the hierarchy changes.

Review the changes and then apply them in the widget.

Example

Consider the design hierarchy shown in the following Hierarchy Manipulation widget:



Now consider that you specify the following command:

```
restruct_rtl -file top.tcl -ui
```

Where top.tcl contains the following commands:

```
new design MASTER_sec_pu
new design MASTER_th_h_1_pu
new design MASTER_sec_h_3_pu
new design MASTER_fir_pu
new design MASTER_th_h_2_pu

select_design top
set_physical_units -names {MASTER_sec_pu MASTER_th_h_1_pu
  MASTER_sec_h_3_pu MASTER_fir_pu MASTER_th_h_2_pu}
select_design top
add_instance -name sec_pu -master MASTER_sec_pu
add_instance -name th_h_1_pu -master MASTER_th_h_1_pu
add_instance -name sec_h_3_pu -master MASTER_sec_h_3_pu
add_instance -name fir_pu -master MASTER_fir_pu
```



```

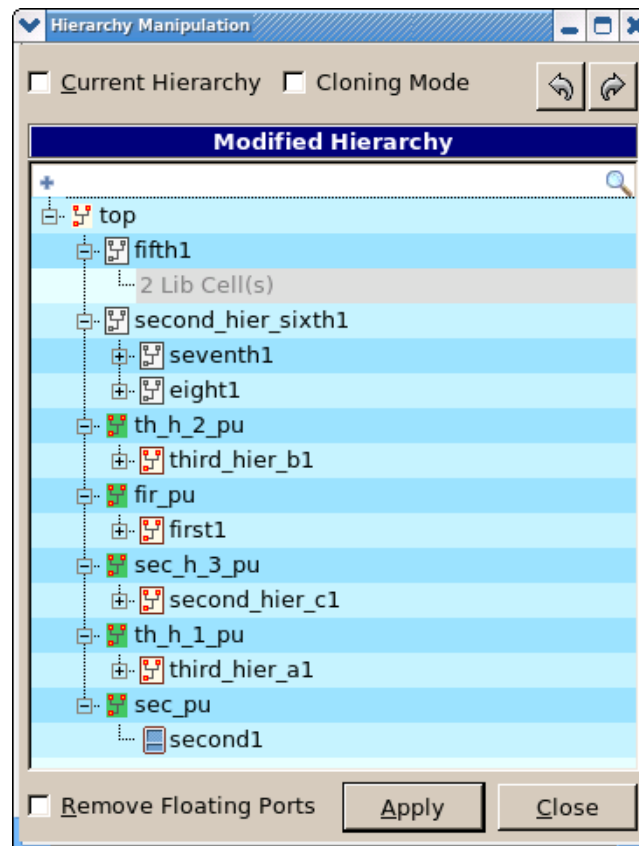
add_instance -name th_h_2_pu -master MASTER_th_h_2_pu

# moving unassigned(to pu) instance/hierarchies
move_instance -instance second_hier_sixth1 -from top::third_hier_a1 -to
top
move_instance -instance fifth1 -from
top::third_hier_a1::second_hier_fifth1 -to top

# moving other hierarchies now
move_instance -instance second1 -from top -to
top::sec_pu
move_instance -instance third_hier_a1 -from top -to
top::th_h_1_pu
move_instance -instance second_hier_c1 -from top -to
top::sec_h_3_pu
move_instance -instance first1 -from top -to
top::fir_pu
move_instance -instance third_hier_b1 -from top -to
top::th_h_2_pu

```

The following Hierarchy Manipulation widget appears:



Review the changes in the above dialog before applying them.

Note:

In the above dialog, you cannot undo the changes done by the `restruct_rtl` command. However, you can undo and redo the changes made over these changes.

run

Runs the specified Tcl script file or generator

Usage

```
run <file-name> | <gen-name>
    [ -verbose ]
```

Description

The run command runs the specified Tcl script file *<file-name>* or the specified generator *<gen-name>*.

Arguments

The run command has the following arguments:

<file-name> | *<gen-name>*

Specifies the Tcl script file name or the generator name.

`-verbose`

You can specify the `-verbose` option only if the Tcl script file is specified.

(Optional) Dumps the Tcl commands executed from the Tcl script file *<file-name>* into a log file under the GenSys log directory. All the dumped Tcl commands in the log file are commented.

Note:

The standard mechanism to load Tcl commands is the Tcl source command. However, the GUI is updated for each Tcl command executed when using the source command. Thus, the run command is the preferred mechanism as it additionally suppresses the GUI update of intermediate commands, resulting in much better performance.

Examples

The following example runs the Tcl script file named mytcl1.tcl:

```
run mytcl1.tcl
```

The following example runs the generator named myGen01:

```
run myGen01
```

run_autoconnect

Runs all the rules created by using the `add_autoconnect_rule` Tcl command

Usage

```
run_autoconnect
[ -name <rules-list> ]
[ -elaborate <Y | y | N | n> ]
[ -preserve_parameter <TRUE | FALSE> ]
[ -export_parameter <FALSE | TRUE> ]
```

Description

The `run_autoconnect` command runs the rules created by using the [add_autoconnect_rule](#) Tcl command. Based on the rules executed, GenSys generates connections automatically.

Arguments

The `run_autoconnect` command has the following arguments:

`-name <rules-list>`

(Optional) Specifies a space-separated list of rules that you want to run for generating automatic connections.

By default, GenSys runs all the rules created by using the [add_autoconnect_rule](#) Tcl command.

You can use regular expressions with this argument. The following examples shows the usage of this argument:

```
run_autoconnect -name { rule1 rule2 }
run_autoconnect -name { rule1 b.* }
```

`-elaborate <Y | y | N | n>`

(Optional) Elaborates the design and then generates connections.

Set this argument to n or N if you do not want to perform elaboration.

`-preserve_parameter <FALSE | TRUE>`

(Optional) Preserves parameters of the exported pin that is created while exporting connections.

If the pin being exported is parameterized and the parameters controlling the exported pin are mapped to the parent design parameters, you can enable GenSys to:

- Create a parameterized exported port on the parent design boundary such that the exported port is controlled by the parent design parameters.
- Assign the same global parameter, macro, or constant to the exported port if the port before exporting is controlled by a global SystemVerilog parameter, macro, or a constant.

Parameters are preserved under the following conditions:

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, parameters are not exported. To export parameters, set this argument to TRUE.

`-export_parameter <TRUE | FALSE>`

(Optional) Specifies whether during export connections are being created, must the parameters controlling the exported pin that are not mapped to the parent design parameters be exported to the parent design and start controlling from parent design.

Parameters are exported and preserved under the following conditions:

- The complete pin is exported, not a slice.
- The exported port does not already exist in the parent design boundary.

By default, the value of this argument has been set to FALSE. This option can be TRUE only when the `-preserve_parameter` argument is also TRUE.

Example

This example shows the impact of the `export_parameter` and `preserve_parameter` arguments. The following table shows the original RTL and the RTL generated after running the `run_autoconnect` Tcl command with the `-preserve_parameter` and the `-export_parameter` arguments.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top(); submodule sub(); endmodule module submodule(out); parameter P = 1; output [P:0] out; leaf l(); endmodule module leaf(); endmodule </pre>	<pre> add_autoconnect_rule -src_inst sub -src_pin out -export_hier TRUE run_autoconnect -export_parameter TRUE -preserve_parameter TRUE </pre>	<pre> module top (out); parameter sub_P = 1; output [sub_P:0] out; submodule #(.P(sub_P)) sub (out); endmodule </pre> <p>Note: The parameter P of submodule is exported as sub_P to the top module and being controlled from top.</p>

run_creategroup

Runs the group specified using the set_group command

Usage

```
run_creategroup [ <partition-name> ]
```

Description

The run_creategroup command triggers grouping after you have set hierarchies by using the [set_group](#) command.

Arguments

The run_creategroup command has the following arguments:

<partition-name>

Specifies the partition name.

run_scr_checks

Runs SCR checks to validate correctness of IP-XACT files

Usage

```
run_scr_checks [ -hierarchy <true | false> ]
```

Description

The run_scr_checks command runs Semantic Consistency Rules (SCR). For details on these rules, refer to GenSys User Guide.

By default, the above checks are automatically run while saving IP-XACT files. However, if you want to execute these checks at any time during the session, other than IP-XACT save, run this Tcl command.

SCR checks are used to validate the correctness of IP-XACT files.

Arguments

The run_scr_checks command has the following arguments:

-hierarchy <true | false>

(Optional): Set this argument to true to run SCR checks over the entire hierarchy loaded in GenSys.

By default, this argument is set to false.

save

Saves the designs/components/interfaces as XML, Tcl, IP-XACT format files

Usage

```
save <design-object> [ -top ] [ -blackbox ] [ -file [ <file-name> ]  
] [ -version <version-num> ] [ -vendor <vendor-name> ] [ -library  
<library-name> ] [ -ipxact_version <version-num> ]
```

Description

The save command saves the specified object (design, component, and interface) in its original format under its original directory.

However, if you are saving a newly created object for the first time, GenSys considers the format and directory for that object based on the settings specified by using the [set_save_template](#), [set_default_save_format](#), and [set_default_save_dir](#) Tcl commands.

Saving in a Canonical XML Format

While saving a design/component/interface in a canonical XML format, GenSys saves the object information as per the schema of the table and NOT by using the hard-coded tags as it is done in the case of MRS.

Note:

The canonical XML files have the GenesisXmlVersion as 2.0, instead of 1.0 as in the case of MRS files.

While saving an object in canonical XML format, GenSys uses the canonical names of table/column for row headers. Canonical names can be specified by using the CANONICAL_NAME tag in GenSys schemas. If canonical names are not specified, GenSys uses the MRS names.

The following example creates a canonical name, Port:

```
TABLE{   NAME :: Ports   MRSNAME :: Ports   CANONICAL_NAME ::
Port    GRPID :: 4      ...}
```

Here, Port will be used as a row header for the Ports table in the canonical XML file. The canonical dump for the Ports table in this case will look like the following:

```
<Ports>   <Port>       <_Comment_Flag>false</_Comment_Flag>   <name>p1</
name>      <logical>p1</logical>   <direction>IN</direction>   <LSB>0</
LSB>      <MSB>7</MSB>   <status>NON_INTERFACE_PORTS</
status>    <alignment>LSB</alignment>   </
Port>   <Port>       <_Comment_Flag>false</_Comment_Flag>   <name>p2</
name>      <logical>p2</logical>   <direction>OUT</direction>   <LSB>0</
LSB>      <MSB>7</MSB>   <status>NON_INTERFACE_PORTS</
status>    <alignment>MSB</alignment>   </Port></Ports>
```

If a table contains only one row, GenSys does not dump the row header in the canonical XML file.

Saving in an IP-XACT Format

When a COM interface is saved in the IP-XACT format, GenSys translates the COM object into IP-XACT format and then saves it into IP-XACT XML format. Hence, now you can load an IP-XACT XML interface file into GenSys, make some changes in GenSys and save it again in IP-XACT format. All the changes will be reflected in the new IP-XACT XML file.

When a COM design is saved in the IP-XACT format, two files are created to save the component and design details separately.

The design details include details of instances, ports, connections, parameters, interfaces, and VLNV. These details are saved in a file with name as specified with the -file argument of the save command. If a file name is not specified, a file with same name as of design is created.

The component details include details of ports, interfaces, parameters, registers, and VLNV. The file in which component details are saved has the name as specified with the -file argument with Comp_ prefix.

When a COM component is specified in IP-XACT format using the save command, only component details of a design are saved.

Limitations of Saving COM Objects in an IP-XACT Format

Following are some of the limitations in saving COM objects in IP-XACT format:

- Currently, tieoff, intentional open, logical, and splice interface connections cannot be saved in IP-XACT format using the save command.
- Attributes are not saved in the IP-XACT file except for InterfaceLib. In case of InterfaceLib, attributes are mapped to parameters.
- Complex parameters are not saved in the IP-XACT file.
- Some of the MRS fields do not have corresponding IP-XACT field. Few examples are listed below:

COM Object	MRS field for which IP-XACT field is not available
Port	Port Type Optional

COM Object	MRS field for which IP-XACT field is not available
BitField	ResetValue Few values of AccessType field. For example, RESERVED. IP-XACT access type has only three values: RO, WO, RW.
Component	Component Type

- If user-defined MRS fields have spaces or special characters, GenSys gives validation error in IP-XACT.

Arguments

The save command has the following arguments:

<design-object>

Name of the design/component/interface to be saved.

-blackbox

(Optional) Specifies that the design is to be saved as a blackbox in Tcl or XML format. The saved blackbox will have ports and interfaces on its boundary corresponding to normal or exported ports/interfaces within the top-level design.

While creating ports/interfaces for a blackbox, GenSys uses elaborated values of parameterized ports/interfaces if they use parameters of the instances inside the design. If parameterized ports/interfaces use parameters of the design itself, then they are kept as is.

By default, the blackbox view of the design is saved in a file named <design-name>.cmd. You can also provide a different name using the -file argument. You can also specify the directory in which you want to save the object by using the -dir argument. In this case, it is important to specify the -blackbox argument after the -dir argument.

The blackbox of a design contains ports, interfaces, attributes, and parameters corresponding to its design view.

Note:

The VLNV of both the design and its blackbox view is same.

Note:

Currently, you can only save a design as a blackbox.

`-file <file-name>`

(Optional) Specifies the file to which the object will be saved.

Note:

Only the file name should be used with the `-file` argument. The directory in which the files need to be saved should be specified by using `<dir>` argument. Also, be careful that you specify a unique name with the `-file` argument, otherwise, an existing file might get overwritten.

`-version <version-num>`

(Optional) Specifies the version num of the design to be saved.

`-vendor_name <vendor-name>`

(Optional) Specifies the vendor name of the design to be saved.

`-library <library-name>`

(Optional) Specifies the library name of the design to be saved.

`-ipxact_version <version-num>`

(Optional) Specifies the IP-XACT version number. You can specify the value of this argument as 1.2 or 1.4.

If you have not specified any version and the object being saved is created by loading the IP-XACT file then the object will be saved in the same version as the previous version. However, if the object being saved is an MRS/TCL object then such object will be saved in the IP-XACT version 1.4.

Examples

The following example saves component comp3 in an XML file named component3.xml:

```
save comp3 -file component3.xml
```

save_autoconnect_rules

Saves details of autoconnect rules in Tcl format

Usage

```
save_autoconnect_rules [ -file <file-name> ] [ -dir <directory-name> ]
```

Description

The `save_autoconnect_rules` command saves details of autoconnect rules in a Tcl format.

Arguments

The `save_autoconnect_rules` command has the following arguments:

`-file <file-name>`

(Optional) Specifies the name of the Tcl file in which the details of the auto-connect rule should be saved.

`-dir <directory-name>`

(Optional) Specifies the directory in which the Tcl file should be saved.

save_session

Saves the current session

Usage

```
save_session -state <dir-name> -overwrite
```

Description

The `save_session` command saves the current session.

When you run this command, GenSys takes a snapshot of the entire database and saves it in a hierarchical NVLV database format.

You can later restore/load the saved session by running the [restore_session](#) command.

Arguments

The `save_session` command has the following arguments:

`-state <dir-name>`

Specifies the name of a directory under which the current session should be saved.

Following is the path of this directory (*<dir-name>*):

<current-working-directory>/gensys_dir/gensys_states/<dir-name>

-overwrite

Overwrites the already existing directory if its name matches with the directory name specified by the -state argument.

Example

The commands specified in the following steps explain the purpose of the save_session command:

1. Creating components and instantiate them in a design

```
new component comp1 -version_num 1.0 -vendor_name Atrenta
-library_name lib1 add_port -name p1-direction OUT -lsb 0 -msb 10
new component comp2 -vendor_name atrenta-version_num 2.0 -library_name
lib1 add_port -name p2 -direction IN -lsb 0 -msb 10
new design d
add_instance -name inst1 -master comp1
add_instance -name inst2 -master comp2 -vendor atrenta -version 2.0
-library lib1
```

2. Saving the session of step 1

```
save_session -state first_session
```

In this step, GenSys saves the current session at the following path:

<current-working-directory>/gensys_dir/gensys_states/first_session

3. Saving the session of step 2

```
add_instance -name inst4 -master comp1delete_instance -name inst1
save_session -state second_session
```

In this step, GenSys saves the current session in the second_session directory with latest changes, such as addition and deletion of instances. This directory is saved under the current working directory.

Now, the following directory structures are present:

<current-working-directory>/gensys_dir/gensys_states/first_session

<current-working-directory>/gensys_dir/gensys_states/second_session

4. Overwriting a saved session

```
add_port -name prt1
save_session -state first_session -overwrite
```

In this step, GenSys saves the current session in the first_session directory with the latest changes, such as addition of the prt port.

In this case, the `first_session` directory containing the session saved in the `step2` is overwritten with the latest changes in the current session.

Also see [Example](#) of the `restore_session` command.

saveall

Saves all active objects

Usage

```
saveall    [ -top ]
```

Description

The `saveall` command saves all top level objects (designs, interfaces, and components).

By default, newly created objects are saved as XML files. However, if you set the format using the [set_default_save_format](#) command, the `saveall` command saves in the specified format. Existing objects are saved in the original format.

This Tcl command is used as an alternative to the switches, `-dumptcl`, `-dumpxml`, and, `-dumpspirit`.

Arguments

The `saveall` command has the following arguments:

- top

(Optional) Specifies that all top level objects should be saved.

Examples

The following example saves all active objects files as MRS files:

```
saveall
```

sdi_end_update

End updates to the most recent table

Usage

`sdi_end_update`

Description

The `sdi_end_update` command end updates to the most recent table referenced by the [sdi_start_update](#) Tcl command.

Note:

Start and end pairs nest. Therefore, you can start updating table A, then start updating table B, and then end update to table B, which will take you back to table A.

sdi_start_update

Start row updates

Usage

`sdi_start_update`

Description

The `sdi_start_update` command reports that subsequent [sdi_update_row](#) commands will apply to the current table.

sdi_update_row

Updates information in a row

Usage

`sdi_update_row`

Description

The `sdi_start_update` command updates information in a row.

Unlike the `update_row` command, this command will create the row if it does not exist.

select_component

Selects the specified component

Usage

```
select_component <comp-name> [ -vendor <vendor-name> ] [ -version  
<version-num> ] [ -library <library-name> ]
```

Description

The `select_component` command selects the specified component. You may also select a component by the vendor name, component version and/or library name.

Arguments

The `select_component` command has the following arguments:

`<comp-name>`

Specifies the name of the component to be selected.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name for the component to be selected.

Note:

If you do not specify the `-vendor` argument, GenSys selects the specified object of the first found vendor.

`-version <version-num>`

(Optional) Specifies the version number of the component to be selected.

Note:

If you do not specify the `-version` argument, GenSys selects the specified object of the first found version.

`-library <library-name>`

(Optional) Specifies the library name for the component to be selected.

Note:

If you do not specify the -library argument, GenSys selects the component of the first-found library.

Note:

If none of the -version, -vendor , and -library argument specified, GenSys selects the specified object of the first found vendor, version, and library.

Examples

The following example selects the component, comp1:

```
select_component comp1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_design

Selects the specified design

Usage

```
select_design <des-name> [ -vendor <vendor-name> ] [ -version  
<version-num> ] [ -library <library-name> ]
```

Note:

The -vendor/-version/-library options should be used only when <des-name> is specified.

Description

The select_design command selects the specified design *<des-name>*. You may also specify the vendor name, library name, and/or version of the design that needs to be selected.

Arguments

The select_design command has the following arguments:

<des-name>

Specifies the name of the design to be selected.

-vendor <vendor-name>

(Optional) Specifies the vendor name for the design to be selected.

Note:

If you do not specify the -vendor argument, GenSys selects the specified design of the first found vendor.

-version <version-num>

(Optional) Specifies the version number of the design to be selected.

Note:

If you do not specify the -version argument, GenSys selects the specified design of the first found version.

-library <library-name>

(Optional) Specifies the library name for the design to be selected.

Note:

If you do not specify the -library argument, GenSys selects the specified design of the first found library.

Note:

If none the -version, -vendor , and -library arguments are specified, GenSys selects the specified design of the first found vendor, version, and library.

Examples

The following example selects the design, dsgn1:

```
select_design dsgn1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_instance

Selects the specified instance

Usage

```
select_instance <inst-name>
```

Description

The select_instance command selects the specified instance *<inst-name>*.

Examples

The following example selects the instance, I1:

```
select_instance I1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_interface

Selects the specified interface instance

Usage

```
select_interface <interface-inst-name>
```

Description

The select_interface command selects the specified interface instance *<interface-inst-name>* under the currently active design/component.

Examples

The following example selects the interface instance named IF1 under the currently active design/component:

```
select_interface IF1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_interfaceinst

Sets the specified interface of a given component instance as an active interface instance

Usage

```
select_interfaceinst -instance <instance-name> -interface  
<interface-inst-name>
```

Description

The select_interfaceinst command sets the specified interface of a given component instance as an active interface instance.

Arguments

The select_interfaceinst command has the following arguments:

`-instance <instance-name>`

Specifies the instance whose pin is to be selected. You can specify a hierarchical name to this argument.

`-interface <interface-inst-name>`

Specifies the interface name which you want to select.

Example

The following examples shows the usage of this Tcl command:

```
select_interfaceinst -instance inst1 -interface intf1  
select_interfaceinst -instance sub::inst1 -interface intf1
```

select_interfaceLib

Selects the specified interface library

Usage

```
select_interfaceLib <interface-lib-name> [ -vendor <vendor-name> ] [
-version <version-num> ] [ -library <library-name> ]
```

Description

The `select_interfaceLib` command selects the specified interface library *<interface-lib-name>*.

The interface library is the library of the top-level interface.

You can optionally specify the vendor name, library name, and version of the interface library that needs to be selected.

Arguments

The `select_interfaceLib` command has the following arguments:

`<interface-lib-name>`

Specifies the name of the master interface library to be selected.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name for the master interface library to be selected.

Note:

If you do not specify the `-vendor` argument, GenSys selects the specified Interface library of the first found vendor.

`-version <version-num>`

(Optional) Specifies the version number of the master interface library to be selected.

Note:

If you do not specify the `-version` argument, GenSys selects the specified Interface library of the first found version.

`-library <library-name>`

(Optional) Specifies the library name for the master interface library to be selected.

Note:

If you do not specify the `-library` argument, GenSys selects the specified Interface library of the first found library.

Note:

If none of the `-version`, `-vendor`, and `-library` arguments are specified, GenSys selects the specified Interface library of the first found vendor, version, and library.

Examples

The following example selects the interface library named intf1:

```
select_interfaceLib intf1
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_port

Sets the specified port as the active port

Usage

```
select_port -name <port-name> [ -logical ]
```

Description

The select_port command sets the specified port as the active port.

If this Tcl command fails to set the specified port as the active port, the active object remains the same which was prior to the execution of this command

Arguments

The select_port command has the following arguments:

-name <port-name>

Specifies the name of the port, which needs to be set as the active port.

-logical

(Optional) Specifies that the port, which is being set as active, is an interface port.

If you do not specify this argument, GenSys searches the specified port either in the active design or in the active component, whichever is present, but not in the active interface.

select_row

Selects a row of the current table

Usage

```
select_row first | last | next | prev | { -<col-name> <value> }* |  
-row <row-number>
```

Description

The select_row command selects the specified row of the current table.

You can use the select_row command in the following ways:

```
select_row first
```

Select the first row.

```
select_row last
```

Selects the last row.

```
select_row next
```

Selects the row AFTER the current row.

```
select_row prev
```

Selects the row BEFORE the current row.

```
select_row -size 8
```

Selects the first row AFTER the current row where the size column value is 8.

```
select_row -size 8 -MSB 5
```

Selects the first row AFTER the current row where the size column value is 8 and the MSB column value is 5.

```
select_row -row 1
```

Selects the second row in the table.

Note:

Row numbers in a table start from 0.

Notes

GenSys always searches down from the current row. Thus, you should use the `-<col-name>` argument carefully considering the current row. Consider the following example:

Name	LSB	MSB
P1	1	3
P2	2	5
P3	0	4
P4	1	5

If you are currently at row 1 (Name=P1) and run the following command, row 2 (Name=P2) is selected:

```
select_row -MSB 5
```

However, if you are currently at row 3 (Name=P3) and run the following command, row 4 (Name=P4) is selected:

```
select_row -MSB 5
```

Note:

You can specify the `-<col-name>` argument multiple times so that the row search is more focused.

Note:

The `-<col-name>` argument has precedence over the named row specification.

Examples

The following example selects the first row in the current table:

```
select_row -row 0
```

The following example selects the row AFTER the current row in the current table:

```
select_row next
```

The following example selects the row BEFORE the current row in the current table:

```
select_row prev
```

Note:

To refer to the testcases of the above examples, go to \$SPYGLASS_HOME/ examples/ Genesis/case0 directory.

select_subtable

Selects the specified subtable

Usage

```
select_subtable <subtable-name>
```

Description

The select_subtable command selects the subtable with the specified subtable name *<subtable-name>*.

In the selected table, the last row is set as the current row.

Examples

The following example selects the subtable TypeValue:

```
select_subtable TypeValue
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case6 directory.

select_table

Selects the specified table

Usage

```
select_table <table-name>
```

Description

The `select_table` command selects the specified table *<table-name>*.

You need to specify the full hierarchical name for the table, for example, `component.Ports`.

In the selected table, the last row is set as the current row.

Examples

The following example selects the subtable named `logical_ports` of table `interface`:

```
select_table interface.logical_ports
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case0` directory.

The following example selects the subtable named `Interface` of table `Connections` of the design `des1`:

```
select_table design\[des1\].Connections.Interface
```

Note:

Escape the object names using the backslash character (`\`).

select_terminal

Sets the specified pin of a given instance as an active terminal

Usage

```
select_terminal -instance <instance-name> -pin <pin-name>
```

Description

The `select_terminal` command sets the specified pin of a given instance as an active terminal.

Arguments

The `select_terminal` command has the following arguments:

`-instance <instance-name>`

Specifies the instance whose pin is to be selected. You can also specify a hierarchical name.

`-pin <pin-name>`

Specifies the name of the pin that you want to select.

Example

The following examples shows the usage of this Tcl command:

```
select_terminal -instance inst1 -pin p1
select_terminal -instance
sub::inst1 -pin p2
```

set_active_object

Sets the named object as active

Usage

```
set_active_object -object <design | component | interfaceLib |
  componentInstance | interfaceInstance> -name <name> [ -vendor
<vendor-name> ] [ -version <version-num> ] [ -library <library-name> ]
```

Description

The `set_active_object` command sets the specified object as active from the currently loaded objects. If there are multiple objects with the same name, specify the version, vendor, and library of the object.

Arguments

The `set_active_object` command has the following arguments:

`-object`

Specifies the type of object that is set as active. You can specify the object type as `design`, `component`, `interfaceLib`, `componentInstance`, or `interfaceInstance`.

`-name`

Specifies the name of the object, which needs to be set as active.

If the object is in a particular hierarchy, specify the hierarchical name of the object by using the hierarchy separator set by the [set_hierarchy_separator](#) command. For details, see [Example](#).

`-vendor <vendor-name>`

(Optional) Specifies the vendor name of the object.

`-version <version-num>`

(Optional) Specifies the version number of the object.

`-library <library-name>`

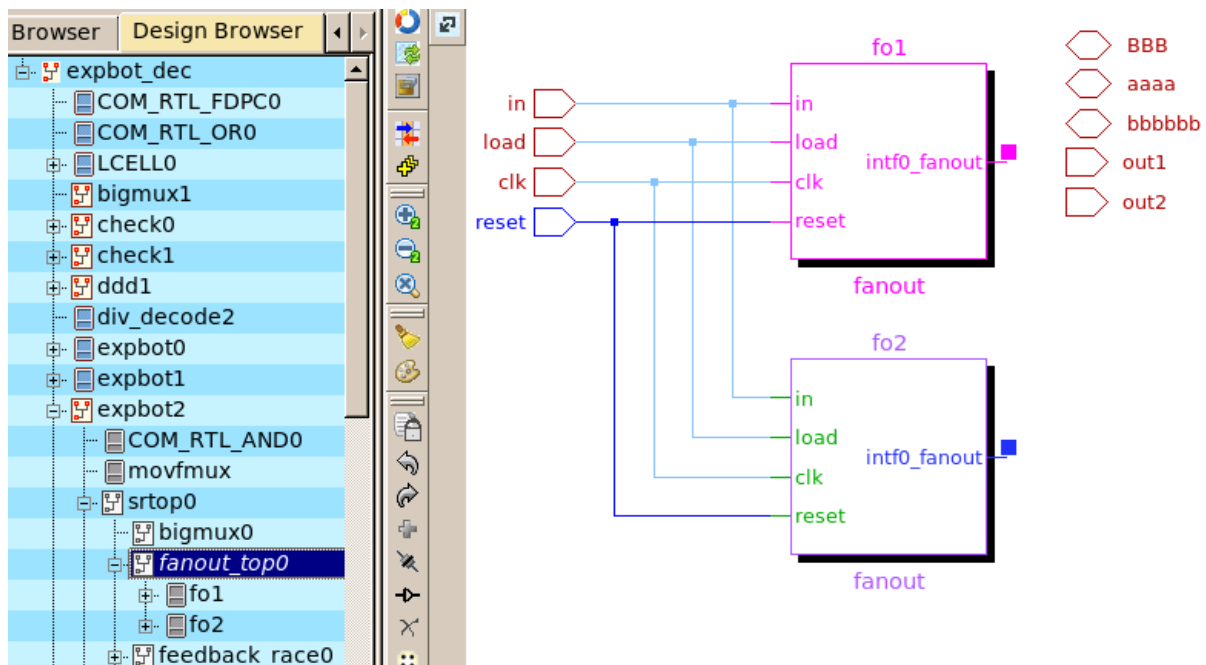
(Optional) Specifies the name of library of the object.

Example

Consider that you specify the following commands:

```
set_hierarchy_separator "::"
set_active_object -object design -name
expbot2::srtop0::fanout_top0
```

On running the above command, the `fanout_top0` object is set as the active object, as shown in the following figure:



set_attribute

Assigns the specified value to an attribute

Usage

```
set_attribute -name <attribute-name> -value <attribute-value>
```

Description

The set_attribute command assigns the specified value to the specified attribute if it is present in the currently active base object.

Arguments

The set_attribute command has the following arguments:

-name <attribute-name>

Specifies the name of an attribute.

-value <attribute-value>

Specifies the value of the specified attribute.

set_autoconnect_options

Specifies various settings while generating automatic connections

Usage

```
set_autoconnect_options [ -allow_interface_fanout <TRUE | FALSE> ] [
-reset ] [ -name <list of space separated rule names> | -all ] [
-hierarchical <TRUE | FALSE> ] [ -consider_order <TRUE | FALSE> ] [
-exclude_inst <instances-list> ] [ -exclude_pins <instance-pin-names>
] [ -exclude_intf <instance-interface-names> ] [ -exclude_component {
-name <comp-name> -vendor <comp-vendor> -version <comp-version>
-library <comp-library> } ]* [ -exclude_intflib { -name
<intflib-name> -vendor <intflib-vendor> -version <intflib-version>
-library <intflib-library> } ]* [ -export_hier <TRUE | FALSE> ] [
-allow_pin_fanout <TRUE | FALSE> ] [ -allow_loopback <TRUE | FALSE> ] [
-strict <TRUE | FALSE> ] [ include_connected_drivers <TRUE | FALSE> ]
```

Description

The `set_autoconnect_options` command specifies various settings while generating automatic connections.

For details on generating automatic connections, refer to *GenSys User Guide*.

Arguments

The `set_autoconnect_options` command has the following arguments:

`-allow_interface_fanout` <TRUE | FALSE>

(Optional) Enables an interface to drive multiple interfaces while generating automatic connections.

By default, this argument is set to FALSE, and an interface is allowed to drive only one interface. Set this argument to TRUE to enable an interface to drive multiple interfaces.

`-reset`

(Optional) Resets all the arguments of this Tcl command to their default values.

`-name` { <space-separated-list-of-rule-names> }

(Optional) Specifies a space-separated list of rules on which the settings specified by this Tcl command will be applicable.

`-all`

(Optional) Applies settings specified by this Tcl command on all rules mentioned in the Autoconnect table.

Note:

You must specify either the `-name` or `-all` argument of this Tcl command. If you do not specify either of these arguments, GenSys reports an error.

`-hierarchical` <TRUE | FALSE>

(Optional) Purpose this argument is same as the `-hierarchical` <TRUE | FALSE> argument of the [add_autoconnect_rule](#) Tcl command.

`-consider_order` <TRUE | FALSE>

(Optional) Purpose this argument is same as the `-consider_order` <TRUE | FALSE> argument of the [add_autoconnect_rule](#) Tcl command.

`-exclude_inst <instances-list>`

(Optional) Purpose this argument is same as the `-exclude_inst {instances-list}` argument of the `add_autoconnect_rule` Tcl command.

`-exclude_pins <instance-pin-names>`

(Optional) Purpose this argument is same as the `-exclude_pins {instance-pin-list}` argument of the `add_autoconnect_rule` Tcl command.

`-exclude_intf <instance-interface-names>`

(Optional) Purpose this argument is same as the `-exclude_intf {instance-interface-list}` argument of the `add_autoconnect_rule` Tcl command.

`-exclude_component { -name <comp-name> -vendor <comp-vendor> -version <comp-version> -library <comp-library> }`

(Optional) Purpose this argument is same as the `-exclude_intf {m0 inst1@m[0|1|3]}-exclude_component { -name <comp-name> -vendor <comp-vendor> -version <comp-version> -library <comp-library> }` argument of the `add_autoconnect_rule` Tcl command.

`-exclude_intflib { -name <intflib-name> -vendor <intflib-vendor> -version <intflib-version> -library <intflib-library> }`

(Optional) Purpose this argument is same as the `-exclude_intflib { -name <intflib-name> -vendor <intflib-vendor> -version <intflib-version> -library <intflib-library> }` argument of the `add_autoconnect_rule` Tcl command.

`-export_hier <TRUE | FALSE>`

(Optional) Purpose of this argument is the same as the `-export_hier <TRUE | FALSE>` argument of the `add_autoconnect_rule` Tcl command.

`-allow_pin_fanout <TRUE | FALSE>`

(Optional) Purpose of this argument is the same as the `-allow_pin_fanout <TRUE | FALSE>` argument of the `add_autoconnect_rule` Tcl command.

`-allow_loopback <TRUE | FALSE>`

(Optional) Purpose of this argument is the same as the `-allow_loopback <TRUE | FALSE>` argument of the `add_autoconnect_rule` Tcl command.

`-strict <TRUE | FALSE>`

(Optional) Specifies if automatic connections should be generated only between ports/pins of the same bit width.

By default, this argument is set to FALSE and GenSys connects pins or ports even if their bit-width does not match.

Set this argument to TRUE so that GenSys does not connect pins or ports of different bit widths.

```
-include_connected_drivers <TRUE | FALSE>
```

(Optional) When `-include_connected_drivers` is set to TRUE then the autoconnect feature considers the fan-out connections for the driver pins/ports even if they have existing connections.

We recommend using the autoconnect rule for pin-to-pin adhoc connections and not for interface connections.

Examples

Example 1

The following command enables an interface to drive multiple interfaces while generating automatic connections and this setting is made applicable to all rules present in the Autoconnect table:

```
set_autoconnect_options -allow_interface_fanout TRUE -all
```

Example 2

The following command enables an interface to drive multiple interfaces while generating automatic connections and this setting is made applicable only to the rule1 rule present in the Autoconnect table:

```
set_autoconnect_options -allow_interface_fanout TRUE -name {rule1}
```

Example 3

The following command enables an interface to drive only one interface while generating automatic connections and this setting is made applicable only to the rule1 and rule2 rules present in the Autoconnect table:

```
set_autoconnect_options -allow_interface_fanout FALSE -name { rule1 rule2  
}
```

set_autoconnect_report_file

Sets the name of the file that is generated after running the `run_autoconnect` command

Usage

```
set_autoconnect_report_file [ -name <file-name> ] [ -reset <true | false> ]
```

Description

The `set_autoconnect_report_file` command sets the name of the file that is generated after running the [run_autoconnect](#) Tcl command.

This file contains details of the status of connections based on the executed rules.

By default, the name of the generated file is `AutoconnectReport.log`.

Arguments

The `set_autoconnect_report_file` command has the following arguments:

`-name <file-name>`

(Optional) Specifies the name of the file in which the auto-connect report will be generated.

`-reset <true | false>`

(Optional) Specifies whether the file name specified by the last executed `set_autoconnect_report_file` Tcl command should be reset to the default name, that is `AutoconnectReport.log`.

By default, this argument is set to `FALSE` and GenSys does not reset the file name. Set this argument to `TRUE` to reset the file name back to its default name.

set_default_port_name

Controls port names by specifying a naming convention for GenSys-generated ports

Usage

```
set_default_port_name -exported_port_name #f <perl-function1> |  
{Expr(<expression1>)} hierarchical_port_name #f <perl-function2> |  
{Expr(<expression2>)}
```

Description

The `set_default_port_name` command controls names of GenSys-generated ports by specifying a naming convention for such ports.

The need to control port names arises in case the RTL boundary of a subsystem is frozen. In such cases, it is important to keep the names of GenSys-generated ports same as what is present in the RTL boundary.

Port naming convention specified by this command is used to set the name of ports generated by the operations through the [export_pin](#), [export_interface](#), [export_unconnected](#), [move_instance](#), and [group_design](#) commands. It also works on rerouting.

Arguments

The `set_default_port_name` command has the following arguments:

`-exported_port_name`

Defines a naming convention for GenSys-generated ports.

You can define the naming convention by specifying an expression.

This argument works when the operation is exporting a port on upper boundary.

{Expr(<expression1>)}

To create an expression, use a combination of the following keywords:

Keywords	Description
si	Represents source instance name
sp	Represents source port name
sintf	Represents source interface name

Keywords	Description
upper(<expression>)	Changes the text generated from <expression> in upper case.
slp	Represents source logical port name
sintfinst	Represents source interface instance name
sd	Represents source port direction
xyz	Represents any string in the expression. For example, in the following expression, atrenta is the string: {Expr(#si_#sp_#sintf_atrenta_#upper(#si_#sp)_#slp_#sintfinst_#sd)}

To understand the usage of this argument, see [Example 1](#).

-hierarchical_port_name

Defines a port naming for GenSys-generated ports while grouping, rerouting, and moving instances ([move_instance](#)).

You can define the naming convention by specifying an expression.

{Expr(<expression2>)}

To create an expression, use a combination of the following keywords:

Keywords	Description
si	Represents source instance name
di	Represents destination instance name
sp	Represents source port name
dp	Represents destination port name
sintf	Represents source interface name
dintf	Represents destination interface name

Keywords	Description
upper(<expression>)	Changes the text generated from <expression> in upper case.
slp	Represents source logical port name
dlp	Represents destination logical port name
sintfinst	Represents source interface instance name
dintfinst	Represents destination interface instance name
sd	Represents source port direction
dd	Represents destination port direction
xyz	Represents any string in the expression. For example, in the following expression, atrenta is the string: {Expr(#si_#sp_#sintf_atrenta_#upper(#si_#sp)_#slp_#sintfinst_#sd)}

To understand the usage of this argument, see [Example 2](#).

Example

Example 1

Consider that you specify Tcl commands, as shown in the following steps:

- Step 1: Setting a global naming convention for GenSys-generated ports by using the following command:

```
set_default_port_name -exported_port_name
{Expr(#si_#sp_#sintf_#upper(#si_#sp)_#slp_#sintfinst_#sd)}
```
- Step 2: Creating a component and adding a port on it.

```
new component compladd_port -name p1 -direction OUT -lsb 0 -msb 10save
```
- Step 3: Creating a new design and instantiating the above-created component in that design.

```
new design dadd_instance -name inst1 -master compl
```

- Step 4: Exporting the unconnected port in the design to the design boundary.

```
export_unconnected -name d
```

- Step 5: Generating Verilog.

```
run GenerateVerilog
```

After performing the above steps, the port exported on the design boundary has the name `inst1_p1_INST1_P1_out`. This name is generated based on the expression specified in Step 1.

Following is the generated Verilog file containing this name:

```
module compl (p1);output [10:0]  p1;endmodule
module d (inst1_p1_INST1_P1_out);output [10:0]  inst1_p1_INST1_P1_out;
compl  inst1 (                .p1(inst1_p1_INST1_P1_out)  //0:11
);endmodule
```

Example 2

Consider that you specify the following Tcl commands:

```
set_default_port_name-hierarchical_port_name{Expr(#si_#sp_#di_#dp)}new
component compadd_port -name plsave
new design dadd_port -name dladd_instance -name inst -master
compadd_connection -port d1 -instance inst -pin plgroup_design -instances
inst -name grouprun GenerateVerilog
```

When you run the above Tcl commands, GenSys creates the `inst1_p1_d_d1` on the group design.

set_default_save_dir

Specifies a directory in which new objects should be saved

Usage

```
set_default_save_dir <dir-name>
```

Description

The `set_default_save_dir` command specifies a directory in which new objects should be saved. The [set_save_template](#) command dictates how the object is saved inside the directory.

Existing objects are always saved back in their original location. Neither the `set_default_save_dir` nor the `set_save_template` is applied while saving existing objects.

If you do not use this Tcl command, new objects are saved in the current working directory.

Note:

This Tcl command is not applicable for existing objects as they are saved in their original location.

When `export_data` is run, `save_template` is used for saving existing objects and new objects. In `export_data`, the `set_default_save_dir` command is not used, instead the objects are exported into the directory specified. The same is applied to the `export_all` command.

Arguments

The `set_default_save_dir` command has the following arguments:

<dir-name>

(Mandatory) Specifies the directory in which new objects should be saved.

Examples

In the following example, the `myDes` is saved into `/tmp/save_dir/ipxactDir/ven1/lib1/1.1`.

```
set_default_save_format ipxact
set_save_template "ipxact" "ipxactDir/%vendor/%version/%library/%obj.xml"
set_default_save_dir /tmp/save_dir
new design myDes -vendor_name "ven1" -library_name "lib1" -version_num
1.1
save
```

Note:

The `/tmp/save_dir/ipxactDir` must exist.

set_default_save_format

Specifies the default format of the files being saved

Usage

```
set_default_save_format <ipxact | IPXACT | mrs | MRS>
```

Description

The `set_default_save_format` command specifies the default format in which the files should be saved.

By default, GenSys saves the files in an MRS format if you do not specify the format of those files while saving them.

If you want to save files in an IP-XACT format by default, specify the following command:

```
set_default_save_format IPXACT
```

In the above case, GenSys saves the files in IP-XACT version 1.2 or 1.4 depending upon the default IP-XACT version set by using the [set_ipxact_version](#) Tcl command. If you have not set any default IP-XACT version by using the `set_ipxact_version` Tcl command, GenSys considers the default IP-XACT version as 1.4.

Note:

Please note the following points:

- The `set_default_save_format` command can be specified in the `.gensys.setup` file or in the Command window in GenSys GUI. However, it is recommended to specify this command in the `.gensys.setup` file.
- If, by default, you want to save files in an IP-XACT format, it is recommended to specify the `set_ipxact_version` Tcl command followed by the `set_default_save_format` Tcl command in the `.gensys.setup` file.

set_default_tieoff_prefix

Sets the default tieoff prefix tag.

Usage

```
set_default_tieoff_prefix <tag>
```

Description

The `set_default_tieoff_prefix` command sets the default tieoff prefix tag *<tag>* for default tieoff connections.

Signal name generated in RTL for default tieoff connections has the following syntax:

<tag>_<binary equivalent of default value>

The default value of the prefix tag is GENESIS_DEFAULT_TIEOFF. Any other prefix tag can be specified using the `set_default_tieoff_prefix` command.

Consider the example given below.

```
new component compadd_port -name P1 -direction IN -lsb 0 -msb
2-default_value 3'b101save compnew design topadd_instance -name I1
-master compsave top
```

In this case, if `set_default_tieoff_prefix` command is not used, name of the default tieoff connection will be set to GENESIS_DEFAULT_TIEOFF_101. If the default tieoff connection is set to AUTOGEN_DEFAULT_TIEOFF using the `set_default_tieoff_prefix` command, name of the default tieoff connection will be set to AUTOGEN_DEFAULT_TIEOFF_101.

Note:

The `set_default_tieoff_prefix` command should be called after setting the `-defval_rtdump` option of the `set_rtl_option` command to TRUE. Otherwise, this command will not have any effect.

set_default_vlv

Specifies the default vendor, library, and version of the newly created object

Usage

```
set_default_vlv [ -vendor <vendor-name> ] [ -library <library-name>
] [ -version <version-num> ] [ -reset <Y | N> ]
```

Description

The `set_default_vlv` command specifies the default vendor, version, and library of the newly created object, such as design, component, interface, and design configuration.

While creating an object, if you do not specify the vendor, library and version (VLV) for that object, GenSys considers the VLV specified by this Tcl command.

Arguments

The `set_default_vlv` command has the following arguments:

`-vendor <vendor-name>`

(Optional) specifies the default vendor name.

`-library <library-name>`

(Optional) specifies the default library name.

`-version <version-num>`

(Optional) specifies the default version number.

`-reset <Y | N>`

(Optional) Set the value of this argument to Y (or y) to reset the default vendor, version, and library.

When you reset VLV, GenSys does not apply the specified default settings of this command to the new objects that are created later.

Consider the following example:

```
set_default_vlv -vendor def_ver -version def_ver -library def_libnew
design dsaveset_default_vlv -reset Ynew design d2save
```

In the above case, GenSys applies the default settings for design d, but not for design d2.

set_elab_option

Sets the elaboration options.

Usage

```
set_elab_option    -align < LSB | MSB > [ -exact_align < TRUE | FALSE > ]
```

Description

The `set_elab_option` command sets the elaboration options for the elaboration phase.

Note:

The `set_elab_option` command should not be used in RTL restructuring and the assembly flow. Preferably, you should set the `set_elab_option` in the `.gensys.setup` file.

Arguments

The `set_elab_option` command has the following arguments:

`-align <LSB | MSB>`

Specifies the default alignment, if the alignment is neither specified for port nor specified in connection.

Note:

If the `-align` option is not set in the GenSys setup file, the default value of `-align` will be MSB. If you set the `set_elab_option` command as `-align LSB`, the default alignment will be LSB.

The LSB or MSB value specifies whether the default alignment should be with respect to LSB or MSB.

`-exact_align < TRUE | FALSE >`

(Optional) Connect exact indices of logical ports at both the ends of an interface connection.

By default value, this argument is set to FALSE. Set the argument to TRUE to connect exact indices of logical ports at both the ends of an interface connection. The following example illustrates the use of this argument.

Consider a design, `des`, that has two instances, `I1` and `I2`. Instance `I1` has interface `intf1` and instance `I2` has interface `intf2`. Now, suppose that an interface connection is to be made between the logical ports, `L1[22:2]` and `L1[23:0]` of interfaces, `I1.intf1` and `I2.intf2`, respectively.

If you do not set the `exact_align` argument to TRUE, GenSys connects the logical ports of the two interfaces in the following manner:

For MSB Aligned Connection

`L1[22:2]` of `intf1` gets connected to `L1[23:3]` of `intf2`

For LSB Aligned Connection

`L1[22:2]` of `intf1` gets connected to `L1[20:0]` of `intf2`

However, if you set the `-exact_align` argument to TRUE, GenSys connects the exact indices of the logical ports at both the ends:

`L1[22:2]` of `intf1` gets connected to `L1[22:2]` of `intf2`

set_extends_vlnv

Sets VLNV information for an extended interface library

Usage

```
set_extends_vlnv [ -name <name> ] [ -version <version> ] [ -vendor  
<vendor> ] [ -library <library> ]
```

Description

The `set_extends_vlnv` command sets the version, name, library, and vendor of an extended interface library.

Arguments

The `set_extends_vlnv` command has the following arguments:

`-name <name>`

(Optional) Specifies the name of the extended interface library.

`-version <version>`

(Optional) Specifies the version of the extended interface library.

`-vendor <vendor>`

(Optional) Specifies the vendor of the extended interface library.

`-library <library>`

(Optional) Specifies the library of the extended interface library.

set_gensys_define

Modifies the value of visibility variables

Usage

```
set_gensys_define -names {<space-separated-list-of-variables>} -values  
{<space-separated-list-of-values>}
```

Description

The `set_gensys_define` command modifies the value (TRUE or FALSE) of the variables added by using the [add_hier_connection](#) Tcl command.

Arguments

The `set_gensys_define` command has the following arguments:

`-names` {<space-separated-list-of-variables>}

Specifies the names of visibility variables whose values need to be modified.

`-values` {<space-separated-list-of-values>}

Specifies the space-separated list of values (TRUE or FALSE) for the visibility variables.

Example

Example 1

The following commands show how you can change the value of previously added visibility variables M1 and M2:

```
add_gensys_define -names {M1 M2 M3} -values {TRUE FALSE TRUE}
set_gensys_define -names {M1 M2} -values {FALSE TRUE}
```

Example 2

Consider that you specify the following command in the `.gensys.setup` file:

```
add_hier_connection -names {M1 M2 M3} -values {TRUE FALSE TRUE}
```

Now consider that you run the following Tcl commands:

```
set_default_vlv -vendor atrenta -library lib -version 1.0load_rtl
-component comp.v -uisavenew design desadd_parameter -name para7 -type
INT -value 30 -rtl Y -rtl_visibility {[d M1]}add_parameter -name para8
-type INT -value 40 -rtl Y -rtl_visibility {[d M2]}add_parameter -name
para9 -type INT -value 20 -rtl Y -rtl_visibility {[d M3]}add_instance
-name comp0 -master comp-rtl_visibility {[d M1]}add_instance -name comp1
-master comp-rtl_visibility {[d M2]}add_instance -name comp2 -master
comp-rtl_visibility {[d M3]}add_port -name Pd1 -lsb 0 -msb {[p para7]}
-direction IN -rtl_visibility {[d M2]}add_port -name Pd2 -lsb 0 -msb
{[p para8]} -direction OUT -rtl_visibility {[d M3]}add_port -name Pd3
-lsb 0 -msb {[p para9]} -direction INOUT -rtl_visibility {[d
M1]}set_verilog_dir V11run GenerateVerilogset_gensys_define -names {M1 M2
M3} -values {FALSE TRUE FALSE}set_verilog_dir V12run GenerateVerilog
```

For the above example, the following `design.v` is generated in the V11 directory:

```

module des (Pd2,          Pd3);parameter    para7    =    30;parameter
para9    =    20;output    [40:0]    Pd2;inout    [para9:0]    Pd3;    comp
comp0 ( //Instance                .p1(), //IO[2:0]
.p2(), //IO[2:0]                .p4(), //IO[2:0]
.p5(), //IO[2:0]                .p6() //IO[2:0]
);    comp    comp2 ( //Instance                .p1(), //IO[2:0]
.p2(), //IO[2:0]                .p4(), //IO[2:0]
.p5(), //IO[2:0]                .p6() //IO[2:0]
);endmodule

```

In the above RTL, note that the comp1 instance, para8 parameter, and pd1 port is not visible. This is because the M2 visibility variable, which is set to FALSE in the .gensys.setup file, is applied on these objects.

However, after generating Verilog in the V11 directory, the values of the visibility variables set in the .gensys.setup file is changed. Therefore, when the Verilog is again generated (this time in the V12 directory), the following design.v is generated:

```

module des (Pd1);parameter    para8    =    40;input    [30:0]    Pd1;
comp    comp1 ( //Instance                .p1(), //IO[2:0]
.p2(), //IO[2:0]                .p4(), //IO[2:0]
.p5(), //IO[2:0]                .p6() //IO[2:0]
);endmodule

```

In the above RTL, note that all the instances, ports, and parameters except comp1, pd1, and para8 are hidden. This is because the visibility variables applied on those hidden instances, ports, and parameters are set to FALSE.

set_gensys_options

Sets global GenSys options

Usage

```

set_gensys_options [ -enable_scr_checks <Y | N> ] [
-export_atrenta_vendor_extensions <Y | N> ] [ -instance_threshold
<value> ] [ -write_mapping_file <Y | N> ] [ -increment_version <TRUE |
FALSE> ] [ -datasource <Y | N> ] [ -preserve_unsynth_constructs <TRUE |
FALSE> ] [ -use_model <IPXACT | RTL | MIXED> ] [ -selective_netlisting
<Y | N> ] [ -enable_group_glue [ <TRUE | FALSE> ] [ -tcldatabase <Y |
N> ] [ -netlist <Y | N> ] [ -write_connectivity_check_sgdc_file <TRUE |
FALSE> ] [ -connectivity_check_sgdc_filename <file-name> ] [
-preserve_global_parameter <TRUE | FALSE>]

```

Description

The `set_gensys_options` command sets global GenSys options.

Note:

This command should be specified in the `.gensys.setup` file.

Arguments

The `set_gensys_options` command has the following arguments:

`-enable_scr_checks <Y | N>`

(Optional) Specifies whether the following Semantic Consistency Rules (SCR) should run while loading IP-XACT files:

SCR-1.1	SCR-1. 2	SCR-1. 3	SCR-1. 4	SCR-1. 5	SCR-1. 8
SCR-1.9	SCR-1. 10	SCR-1. 11	SCR-2. 13	SCR-2. 14	SCR-2. 15
SCR-5.1	SCR-5. 2	SCR-5. 3	SCR-5. 4	SCR-5. 5	SCR-5. 6
SCR-5.1 1	SCR-6. 9	SCR-6. 10	SCR-6. 15	SCR-6. 16	SCR-6. 17
SCR-6.2 1	SCR-6. 22	SCR-6. 23	SCR-6. 24	SCR-6. 25	SCR-6. 26
SCR-6.2 7					

Set this argument to Y to execute the above SCR checks while loading IP-XACT files. By default, this argument is set to N.

`-export_atrenta_vendor_extensions <Y | N>`

(Optional) Controls exporting of Atrenta-specific vendor extensions to IP-XACT files generated in GenSys.

To eliminate Atrenta-specific vendor extensions from IP-XACT files, set this argument to N. By default, this argument is set to Y.

`-instance_threshold <value>`

(Optional) During RTL import, if the number of instances in the subsystem exceeds the value specified, the subsystem is converted into a black box and the hierarchy below the subsystem is ignored. This is done to black box large netlist subsystems and reduce RTL import runtime.

By default, the value is set to 100. Therefore, during the RTL import process, if the number of instances in a subsystem is more than 100, that subsystem is converted into a black box and the hierarchy below it is ignored.

Note:

The value specified for this argument is overridden by the value of the `instance_threshold` argument specified in the [load_rtl](#) Tcl command.

Note:

the `instance_threshold` argument specified in the [load_rtl](#) Tcl command.

`-write_mapping_file <Y | N>`

(Optional) Set this argument to Y before loading RTL ([load_rtl](#)) if you want to use the [write_formality_script](#) Tcl command.

This argument should be set to TRUE only when you want to dump formality mapping file using the `write_formality_script` command.

`-increment_version <Y | N>`

(Optional) Increments the version of a design or a component whenever it is saved after modification.

Consider that you have imported an RTL of module UART. Now, if you save, the version is saved as "noverison". After making changes with GenSys, if this option is set to TRUE, GenSys dumps the version as "noverison.1". For an IPXACT IP, if the imported IP has version 1.0, it is dumped as 1.1 after modifications.

`-datasource <Y | N>`

(Optional) Specifies if the back-reference information of a connection should be added to a connection object.

When you set this argument to N, the connection back-reference information, such as Tcl file and line number from where the connection was given, is not added to the connection object in GenSys. Therefore, RTL health check and other generators do not report the back-reference information.

Note:

After setting this argument to N, if you still want to add some back-reference information to a connection, specify that information by using the [conn_backref](#) `<backref-string>` argument of the [add_connection](#) command.

By default, this argument is set to Y and a back-reference information is attached to a connection object. However, note that if a custom generator report huge number (say thousands) of such connections, the tool will become slow in finding the back-reference information for each connection.

Therefore, you should set this argument to N at the beginning of generator and set it to Y at the end.

```
-preserve_unsynth_constructs <TRUE | FALSE>
```

(Optional) Before loading RTL, set this argument to TRUE so that if the loaded RTL contains unsynthesizable constructs inside pragma and `ifdef blocks, GenSys preserves such constructs in the generated RTL.

By default, the above command is set to FALSE and unsynthesizable constructs are not preserved.

```
-use_model <IPXACT | RTL | MIXED>
```

(Optional) Specifies the mode (IP-XACT, RTL, or mixed) in which GenSys should run.

Specify IPXACT mode when all the SOC and IPs are in IPXACT format only. RTL mode when all the IPs and SOC are in RTL. Mixed mode when some of the IPs are in IPXACT view while others are in RTL.

While changing the mode, ensure that no object is loaded in GenSys. Else, GenSys reports an error.

In each mode, certain set of options are valid. If you try setting an option that is not valid in a particular mode, GenSys reports a warning and change that option to the valid value.

For details, see [Setting GenSys Operating Modes](#).

```
-selective_netlisting <Y | N>
```

(Optional) Enables the surgical netlisting feature in which SpyGlass generates the RTL when you invoke RTL-generation Tcl commands.

Note:

To enable this feature, use the `gen_rtl_sn1` license. If you do not use this license, GenSys invokes the default netlister for RTL generation.

```
-enable_group_glue [ <TRUE | FALSE>
```

(Optional) Set this argument to TRUE to enable the feature on glue logic solidification in which a glue logic is converted into a module that can be moved across hierarchies.

By default, this argument is set to FALSE.

All the behavioral code, i.e. always/process/generates/array of instances, is imported in GenSys as a pseudo GLUE block that gets dissolved during RTL generation. However, you can solidify the GLUE block using the `group_glue` TCL command when this option is set to TRUE. This enables glue block movement.

```
-tcldatabase <Y | N>
```

(Optional) Set this argument to Y to enable fast loading of Tcl database.

By default, this argument is set to N. In the default mode, GenSys works in the incremental elaboration mode in which when you run the `add_connection` Tcl command, GenSys incrementally elaborates that connection.

However when you load a Tcl database that contains thousands of `add_connection` Tcl commands, incremental elaboration of each command will be lot more slower than doing a complete elaboration at the end.

When you set this argument to Y, GenSys does not elaborate connections at once. Instead, you need to run the `elaborate` explicitly. Also, GenSys does not create a table view of connectivity. You will have to run the "`refresh_table Design_Connections`" command to explicitly populate the connection table. This will result in faster loading of Tcl database.

```
-netlist <Y | N>
```

(Optional) Enables the netlist flow.

In the netlist flow, if any RTL construct is encountered during RTL import or during MRS read, GenSys reports an error and aborts RTL import.

An error appears if you try to do the following on the netlist design:

- RTL health check
- SDI implementation
- Adding or setting parameters (`add_parameter/set_parameter`)
- Preserve configurations (using the `-preserve_configurations` argument of the `set_rtlimport_option` Tcl command)
- Presence of glue logic
- Presence of glue/generate/array of instance
- IP-XACT save/restore
- Invoking GenSys products (`gensys -register` or `gensys -io`)
- Presence of macros and include files
- Creating and setting partitions on tables (`create_partition/set_partition`)

- Usage of GenSys tabular view (because the netlist flow works only in the schematic mode)
- Usage of Tcl commands, such as `add_interface_mapping`, `infer_interfaces`, and `infer_interface_connections`

`-print_macro_definition <TRUE|FALSE>`

(Optional) Configures GenSys to generate ``define` macros just before module definitions.

To generate ``define` macros, set the `-print_macro_definition` argument of the `set_gensys_options` Tcl command to `TRUE`. By default, this argument is set to `FALSE`.

GenSys generates ``define` macros that are used in the LSB/MSB indices of:

- Newly created ports by hierarchy manipulation operations (This is shown in the code in green in the example below).
- Wires connected to driver ports of the same LSB/MSB parameterize indices (This is shown in the code in blue in the example below).

In both the above cases, ``define` macros are generated just before the relevant module definition.

`-write_connectivity_check_sgdc_file <TRUE | FALSE>]`

(Optional) When this option is set to true, GenSys starts tagging each connection addition and deletion command executed in GenSys and internally creates SGDC constraints required to verify those connection addition and deletion commands.

For more information on connectivity verification, see the GenSys User Guide and [Example 2](#).

`-connectivity_check_sgdc_filename <file-name>`

(Optional) Use this argument to specify the name of the generated SGDC file. If the file name is not provided, GenSys writes the SGDC constraints in an internally generated file called `verify_connections.sgdc`. This file is located in the report directory.

For more information on connectivity verification, see the GenSys User Guide and [Example 2](#).

`-preserve_global_parameter <TRUE | FALSE>`

(Optional) When this option is set to `FALSE`, the tool does not preserve the ``define` and SystemVerilog global parameters during instance move. The default is `TRUE`, that is, the tool preserves the ``define` and SystemVerilog global parameters during instance move.

When you specify this option, you must also specify the `set_rtl_option -macros_in_wire_indexes` command as `FALSE`, as shown:

```
set_rtl_option -macros_in_wire_indexes FALSE
set_gensys_options -preserve_global_parameter FALSE
```

Examples

Example 1

Consider the following Tcl command:

```
move_instance -instance leaf_ins1 -from top::mid_ins::bot_ins -to top
-preserve_parameter TRUE
```

Based on the above Tcl command, the following table shows the difference between the source RTL and the generated RTL:

Source RTL	Generated RTL
<pre>module top (); mid mid_ins(); endmodule module mid (); bot bot_ins(); endmodule `define P1 5 module bot (); wire [`P1:0]w; leaf leaf_ins1(w); leaf leaf_ins2(w); endmodule module leaf (p); inout [`P1:0] p; endmodule</pre>	<pre>`ifndef P1 `define P1 5 `endif module top (); wire [`P1:0] w; mid mid_ins (.bot_ins_leaf_ins2_p(w)); leaf leaf_ins1 (.p(w)); endmodule `ifndef P1 `define P1 5 `endif module mid (bot_ins_leaf_ins2_p); inout [`P1:0] bot_ins_leaf_ins2_p; wire [`P1:0] w; bot bot_ins (.leaf_ins2_p(w)); assign w = bot_ins_leaf_ins2_p; endmodule `define P1 5 module bot (leaf_ins2_p); inout [`P1:0] leaf_ins2_p; wire [`P1:0] w; leaf leaf_ins2 (.p(w)); assign w = leaf_ins2_p; endmodule module leaf (p); inout [`P1:0] p; endmodule</pre>

Example 2

This example shows the SGDC constraints that are generated for a sample Tcl file called test.tcl. To enable SGDC constraints generation, use the following Tcl command:

```
set_gensys_options -write_connectivity_check_sgdc_file true
```

TCL Script - test.tcl

```
new design top
add_port -name clk -direction IN
add_instance -name ul -master morpheus8_core
add_connection -instance ul::controller0 -pin rdn -instance
ul::controller1 -pin add_in
add_connection -instance ul::controller0 -pin clk -port clk
add_connection -instance ul::controller0 -pin finish -tieoff 1'b1
add_connection -instance ul::controller0 -pin rdaddr -lsb 1 -msb 2
-tieoff 2'b10
add_hier_connection -source {top::ul::controller1::my_out[0:1]} -dest
{top::ul::controller0::my_sig[0:1]} -through top::ul::u2
delete_hier_connection -source clk -dest top::ul::controller0::clk
select_design morpheus8_core
delete_connection -instance controller0 -pin rdaddr -lsb 1 -msb 2 -tieoff
2'b10
select_design top
run GenerateVerilog
```

During RTL generation, GenSys generates a support file containing the SGDC constraints in the report directory. In this example, the `-connectivity_check_sgdc_filename` argument has not been specified in the `set_gensys_option` Tcl option. Therefore, by default, the generated SGDC file is called `verify_connections.sgdc` and it is located in the reports directory.

Generated SGDC Constraints File - verify_connections.sgdc

```
current_design top
require_path -from top.ul.controller1.add_in -to top.ul.controller0.rdn
require_value -name top.ul.controller0.finish -value 1
require_path -from top.ul.controller1.my_out[1] -to
top.ul.controller0.my_sig[1]
require_path -from top.ul.controller1.my_out[0] -to
top.ul.controller0.my_sig[0]
```

To verify these connections in the GenSys-generated RTL, use the generated SGDC file and the GenSys-generated RTL after assembly operations as an input for the SpyGlass Connectivity Verify product.

Notes

- GenSys increments the version of the saved file. However, the version of the currently loaded object remains unchanged.
- This command increments the version of only those objects that are modified after being loaded.

set_glue

Modifies the declarative and behavioral region of a glue block

Usage

```
set_glue -name <name> -language <Verilog | Vhdl> -decl_glue  
<decl-glue> -behav_glue <behav-glue>
```

Note:

This command is added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

Description

The set_glue command modifies declarative and behavioral regions of a glue block.

Arguments

The set_glue command has the following arguments:

-name <name>

Specifies the glue block to be updated.

-language <Verilog | Vhdl>

Specifies the language of the glue block.

```
-decl_glue <decl-glue>
```

Specifies the glue-block declarative region to be updated.

This region contains all the signal declarations corresponding to the glue-block ports and all the supporting logic, such as function definition and type.

```
-behav_glue <behav-glue>
```

Specifies the glue-block behavioral region to be updated.

For a typical always block in Verilog, this argument contains the decompiled always block only.

set_group

Sets various options while grouping instances in the design

Usage

```
set_group -instance {<space-separated-list-of-instances>} -hierarchy  
<hierarchy-name> [ -hier_inst_name <hierarchy-instance-name> ]
```

Description

The set_group command is used to set hierarchies. This command is used to populate the Partition table.

This TCL command works on the active partition.

Arguments

The set_group command has the following arguments:

```
-instance {<space-separated-list-of-instances>}
```

Specifies a space separated list of instances. Instance names have wildcard support.

```
-hierarchy <hierarchy-name>
```

Specifies the hierarchy name.

`-hier_inst_name <hierarchy-instance-name>`

(Optional) Specifies the instance name of the hierarchy created by using the group feature in GenSys. Specifying instance name facilitates further grouping of instance hierarchy with other component/design instances.

Examples

To place all IO buffer instances of pad HD0, HD1 etc. inside the HD hierarchy, you would specify the following command.

```
set_group -instance IO_PAD*HD* -hierarchy HD
```

set_group_mi_module

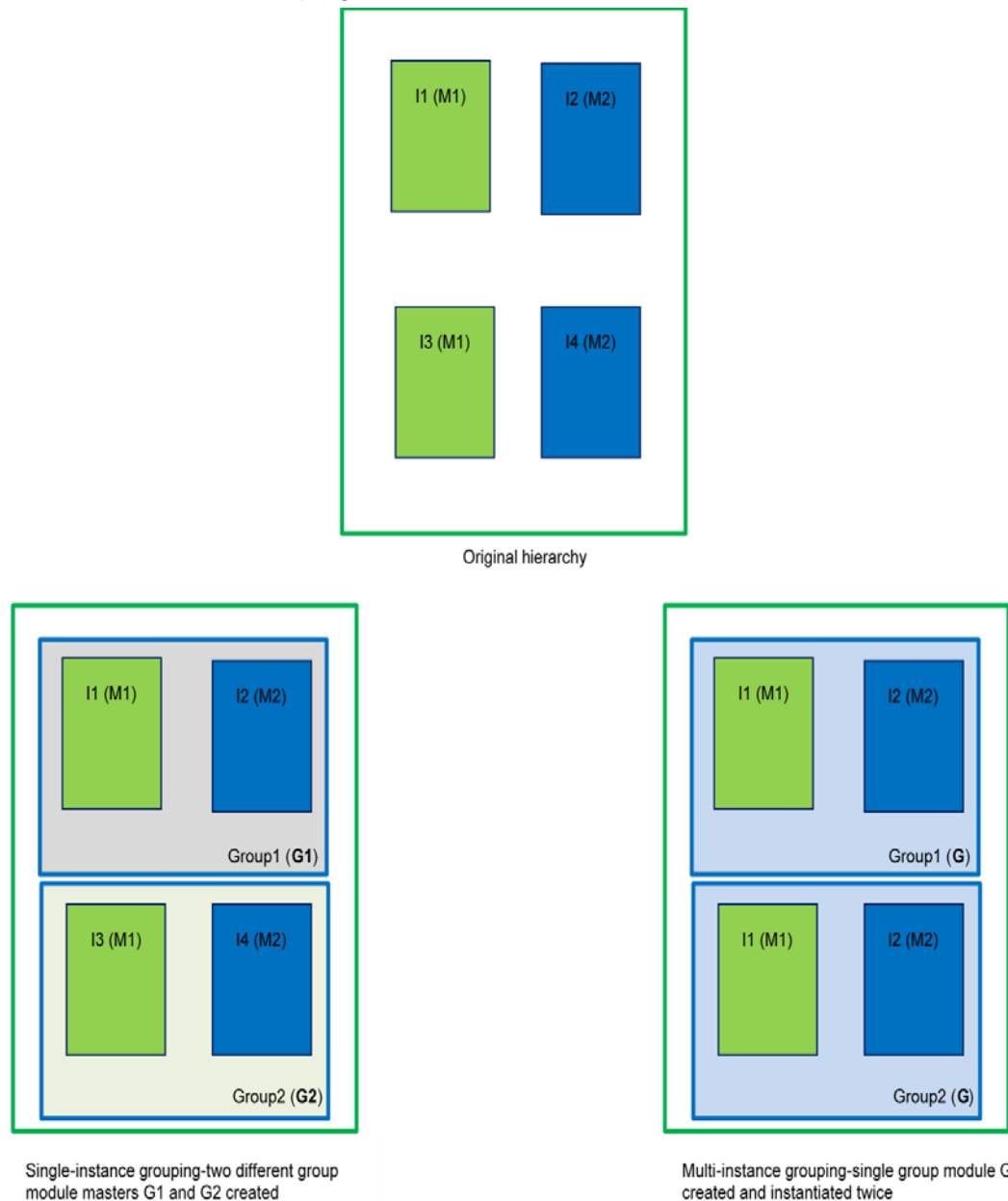
Specifies the multi-instance module (MIM) and the multiple instances of the multi-instance module for group operations

Usage

```
set_group_mi_module
-group_instances {<space-separated-list-of-group-instances>}
-mi_module_name {<module-name>}
```

Description

The `set_group_mi_module` command is used to create a single module master for multiple groups of similar instances and instantiate it multiple times in the design. Such grouping is called multi-instance grouping, and the generated module master is called a multi-instance module (MIM).

Figure 2-1 Multi-Instance Grouping

Use the `set_group_mi_module` command to group similar instances without creating a separate module master for each of the instance groups as shown in [Example - Design Restructuring Using Multi-Instance Grouping](#).

Conditions for Multi-Instance Grouping

Each of the following conditions must be met for multi-instance grouping:

- Number of instances in different group operations must be identical
- Master modules of these instances must match
- Master modules of these instances must be unique
- Instance parameters must match

If any of these conditions are not met, the tool generates an error and does not perform multi-instance grouping for those group operations. It creates modules that are instantiated only once.

Unsupported Scenarios

The tool does not support the following scenarios for multi-instance grouping:

- [Error Scenarios](#)

The prerequisite conditions for multi-instance grouping are not satisfied. In such cases, the tool generates error messages, ignores multi-instance grouping, and proceeds with single-instance grouping.

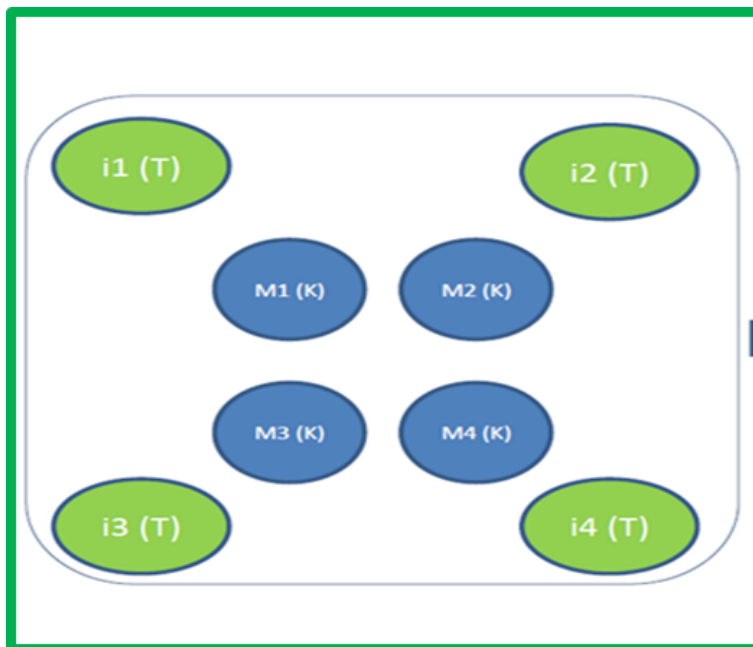
For more information, see [Conditions for Multi-Instance Grouping](#).

- [Unsupported Connection Scenarios](#)

If the tool encounters connection scenarios that are not supported for multi-instance grouping, the tool generates warning messages, ignores multi instance grouping, and proceeds with single-instance grouping.

Error Scenarios

Figure 2-2 Error Scenarios for Multi-Instance Grouping



In [Figure 2-2](#), i1, i2, i3, and i4 are instances of module T, and M1, M2, M3, and M4 are instances of module K. For the instances of [Figure 2-2](#), the following error scenarios can occur.

Error Scenario 1 - Number of Instances do not Match

```
set_group_mi_module -group_instances {G1 G2} -mi_module_name hier1
group_design -instances {i1} -name G1
group_design -instances {i3, M3} -name G2
```

In this case, the tool generates the following error message:

```
ERROR[MI-GROUPING-1]: Number of instances in group 'G2' does not match
with previous group 'G1'. Ignoring multi-instance grouping for 'G2'.
```

Error Scenario 2 - Master Modules of Instances do not Match

```
set_group_mi_module -group_instances {G1 G2} -mi_module_name hier1
group_design -instances {i1, i2} -name G1
group_design -instances {i3, M3} -name G2
```

In this case, the tool generates the following error message:

```
ERROR[MI-GROUPING-2]: Instance 'M3' of module 'K' grouped in 'G2' does
not match with any instance of previous group 'G1'. Ignoring
multi-instance grouping for 'G2'.
```

Error Scenario 3 - Master Modules of Instances are not Unique

```
set_group_mi_module -group_instances {G1 G2} -mi_module_name hier1
group_design -instances {i1, i2} -name G1
```

In this case, the tool generates the following error message:

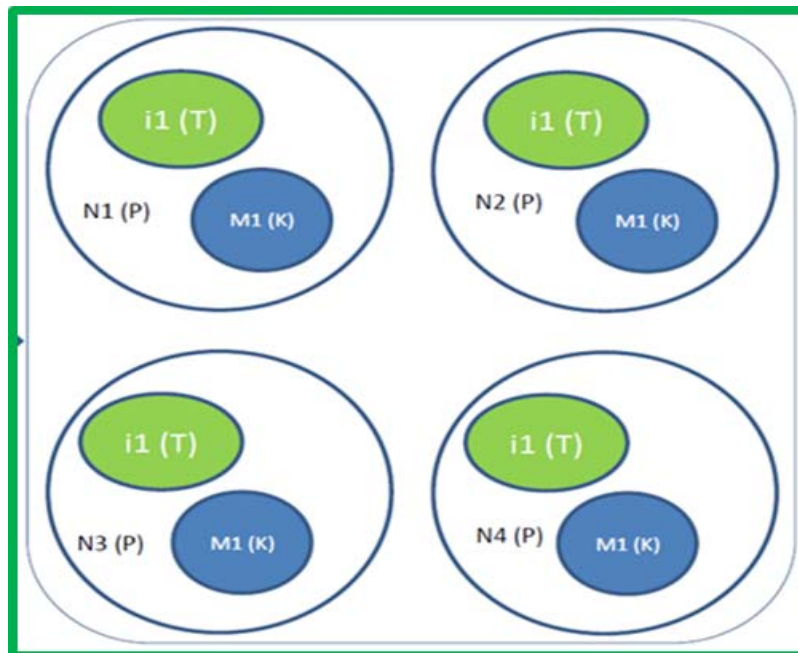
```
ERROR[MI-GROUPING-3]: Module 'T' has multiple instantiations 'i1' and
'i2' in group instance 'G1'. Ignoring multi-instance grouping for 'G1'.
```

The following script meets all conditions of multi-instance grouping for the design instances in [Figure 2-2](#).

```
set_group_mi_module -group_instances {N1 N2 N3 N4} -mi_module_name P
group_design -instances {i1, M1} -name N1
group_design -instances {i2, M2} -name N2
group_design -instances {i3, M3} -name N3
group_design -instances {i4, M4} -name N4
```

[Figure 2-3](#) shows the restructured design for this case.

Figure 2-3 Restructured Design With Multi-Instance Modules



Unsupported Connection Scenarios

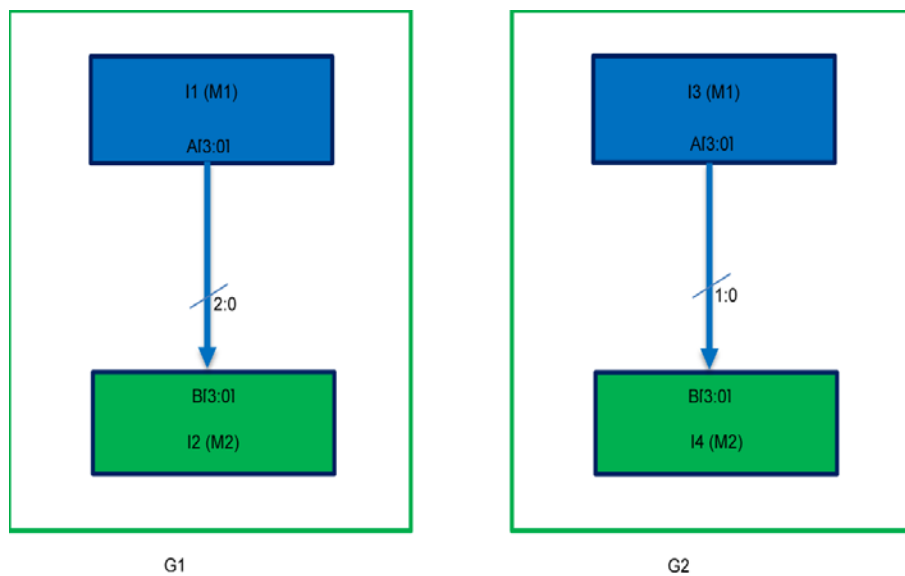
To create a multi-instance module, the tool might alter the interfaces and the internal connections of the different module masters being grouped. This ensures that the interfaces and connections of those modules become identical. The tool then generates the multi-instance module and performs multi-instance grouping.

If bit-level connections of vector ports are different, then the tool cannot create multi-instance modules. The tool then generates a warning message and only performs single-instance grouping.

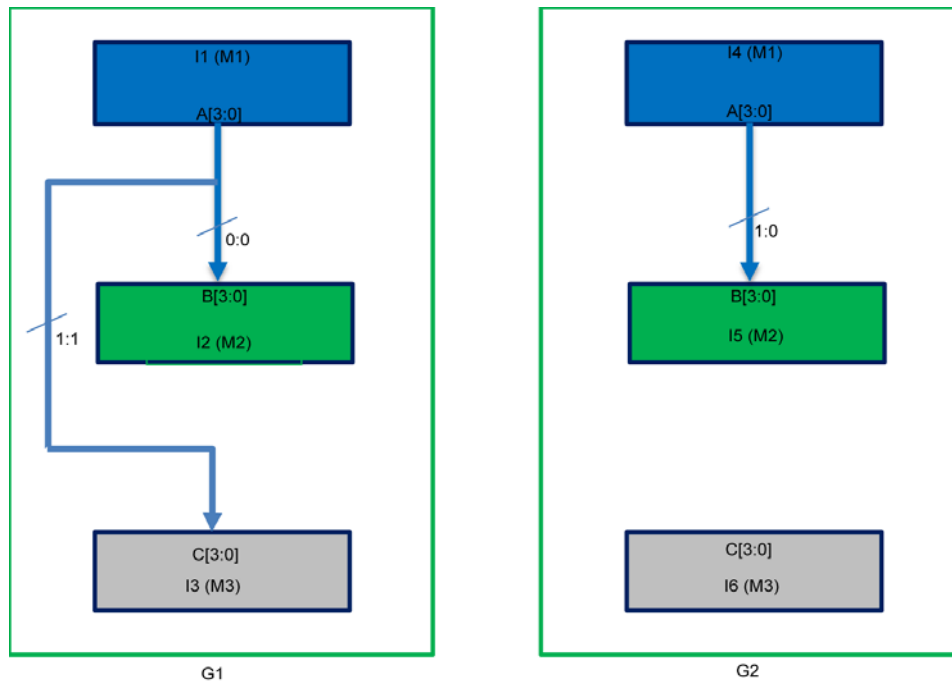
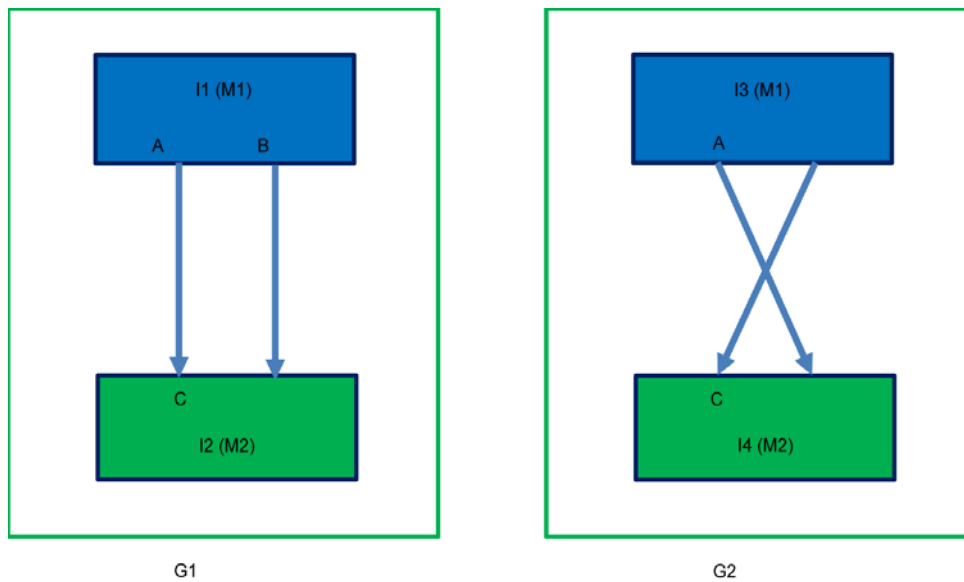
For example, in the unsupported connection scenario shown in [Figure 2-4](#), three bits of port A and port B are connected in the G1 group but only two bits of port A and port B are connected in the G2 group. The tool therefore generates the following warning message:

```
WARNING[MI-GROUPING-4]: Unsupported connection scenario. Connection
between terminal 'A' of instance 'I1' and terminal B of instance 'I2'
in group 'G2' cannot be altered to match connection in previous group
'G1'. Ignoring multi-instance grouping for 'G2'.
```

Figure 2-4 *Unsupported Connection Scenario - Bit Mismatch Between the Same Ports*



In [Figure 2-5](#), there is a mismatch in the number of bits that connect the ports in the two groups. In [Figure 2-6](#), the connections are completely different in the two groups.

Figure 2-5 Unsupported Connection Scenario - Bit Mismatch Between Ports*Figure 2-6 Unsupported Connection Scenario - Different Connections*

Supported Scenarios for Multi-Instance Grouping

The following figures show the similar single-instance modules and the corresponding multi-instance modules that are supported by the tool.

Figure 2-7 Multi-Instance Scenario 1

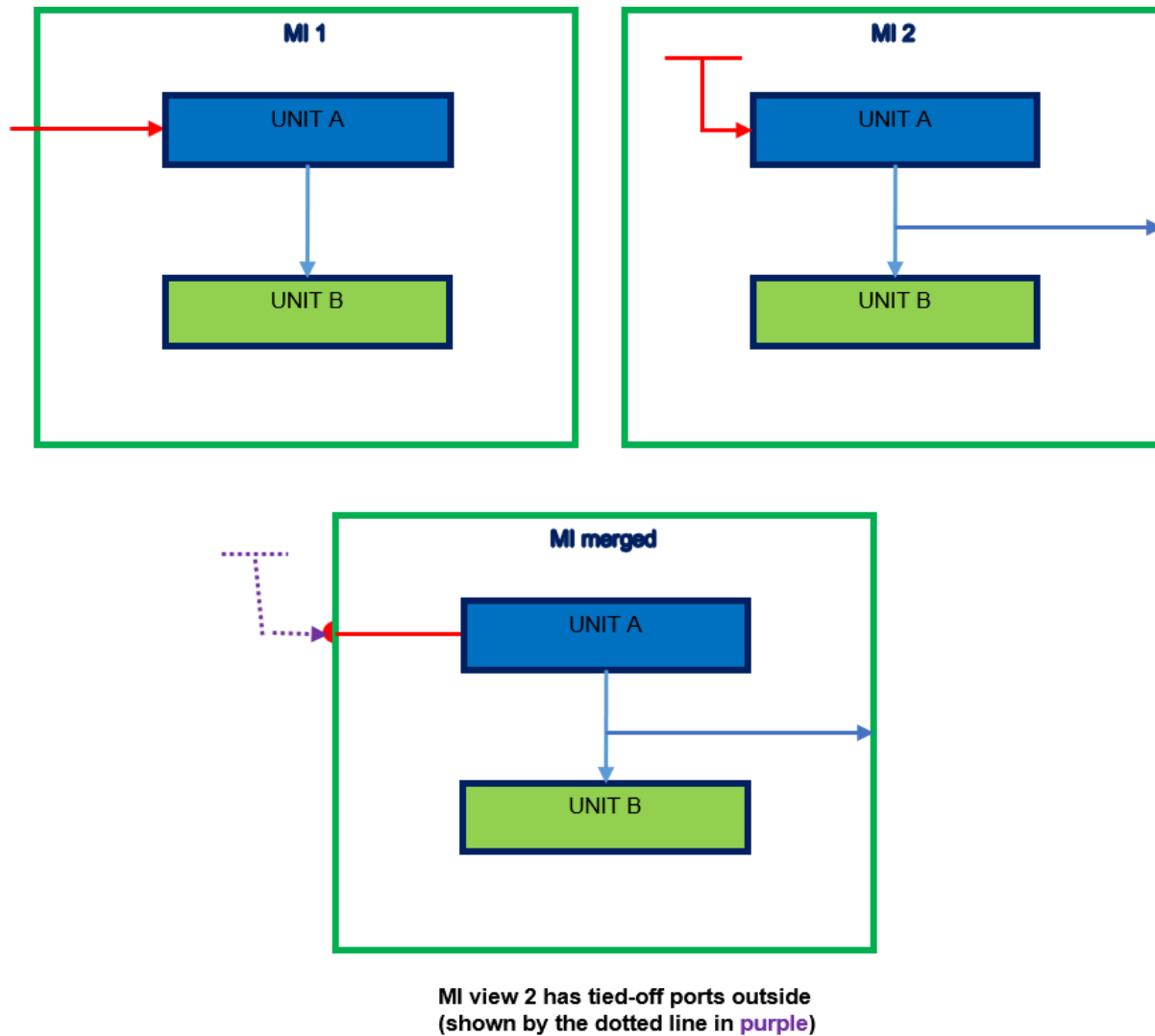


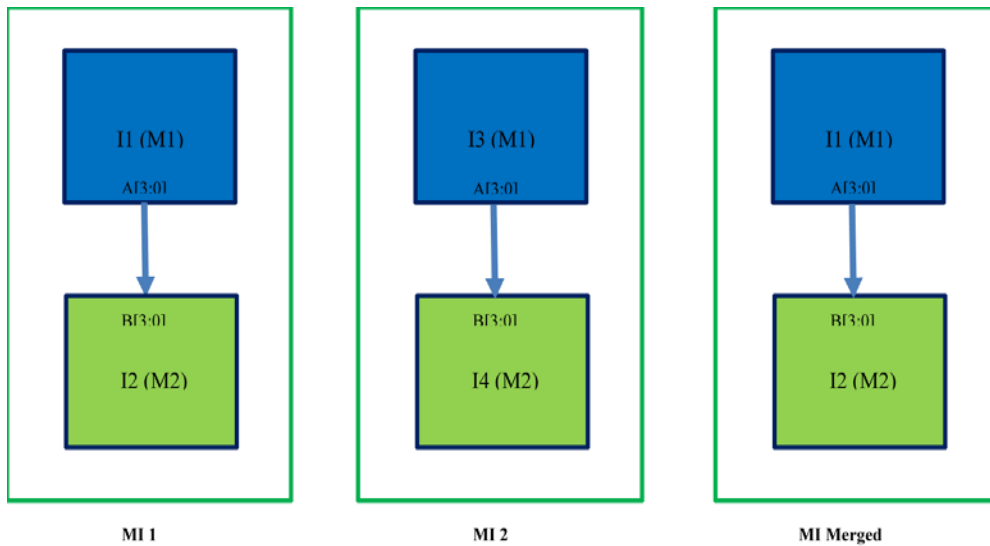
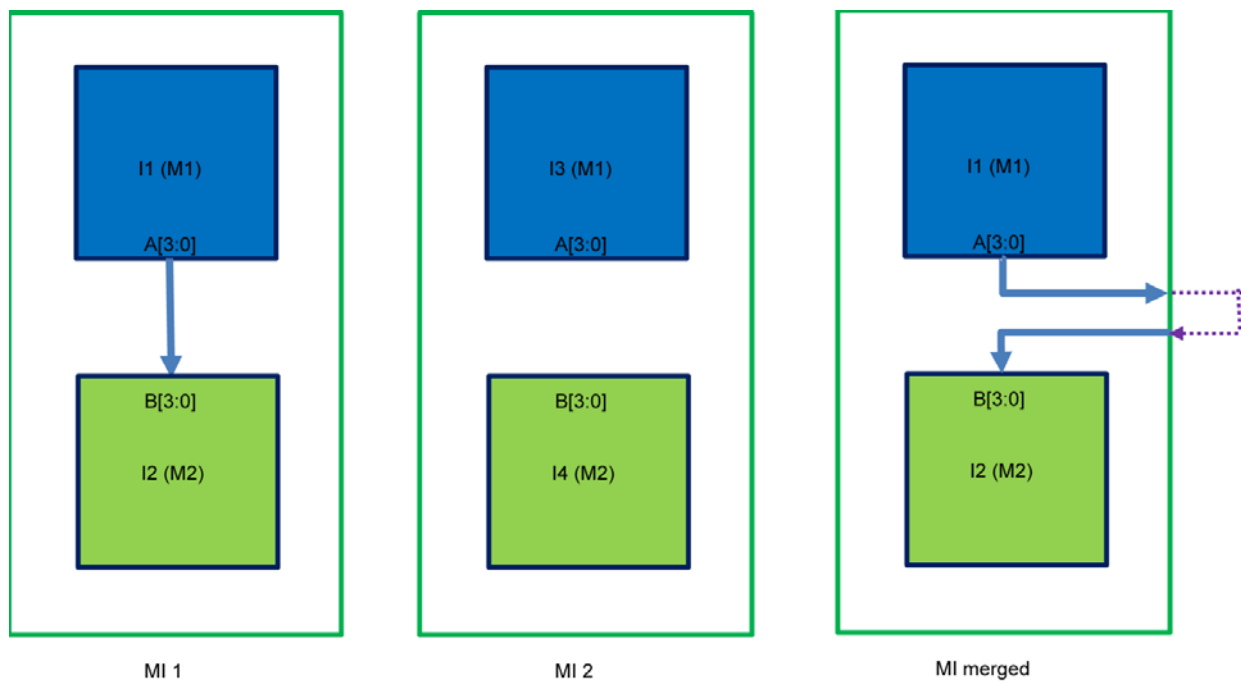
Figure 2-8 Multi-Instance Scenario 2*Figure 2-9 Multi-Instance Scenario 3*

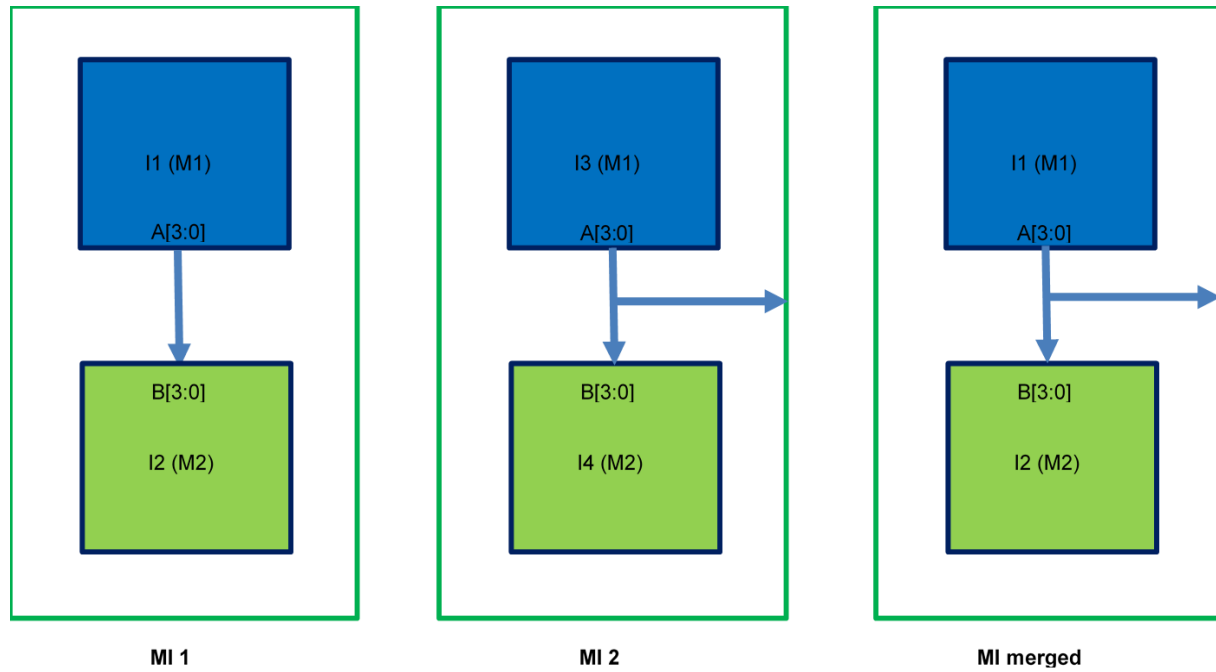
Figure 2-10 Multi-Instance Scenario 4

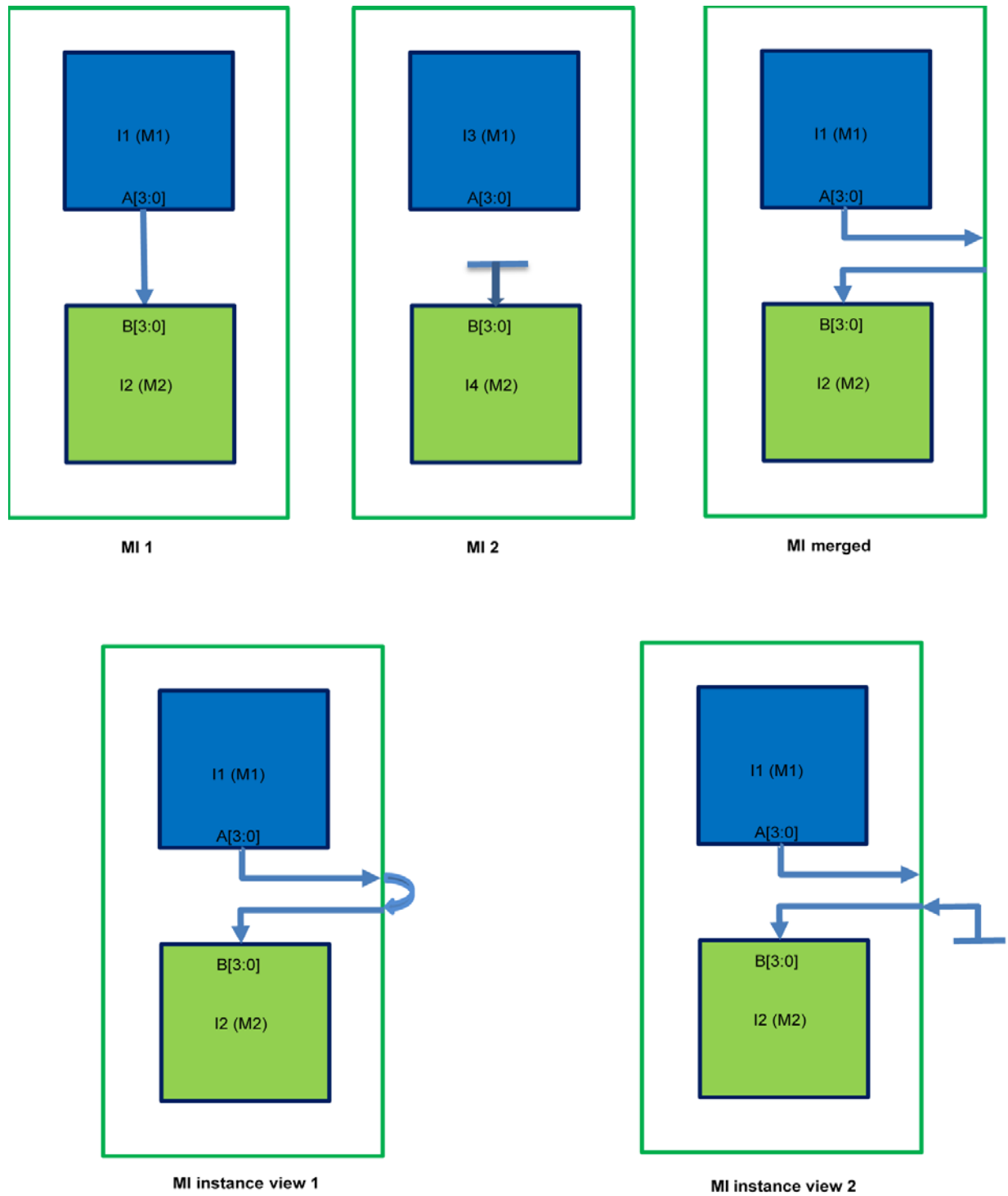
Figure 2-11 Multi-Instance Scenario 5

Figure 2-12 Multi-Instance Scenario 6

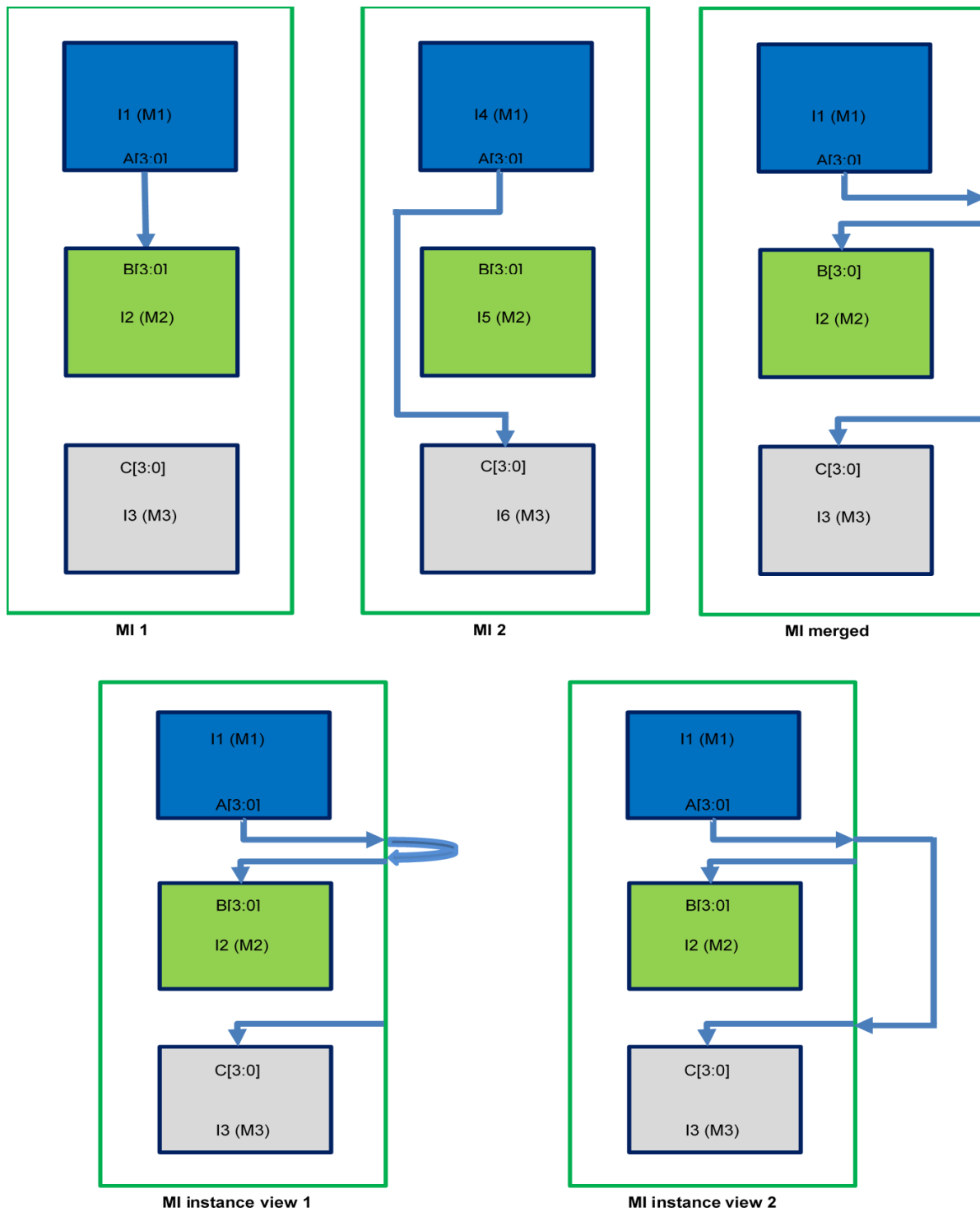
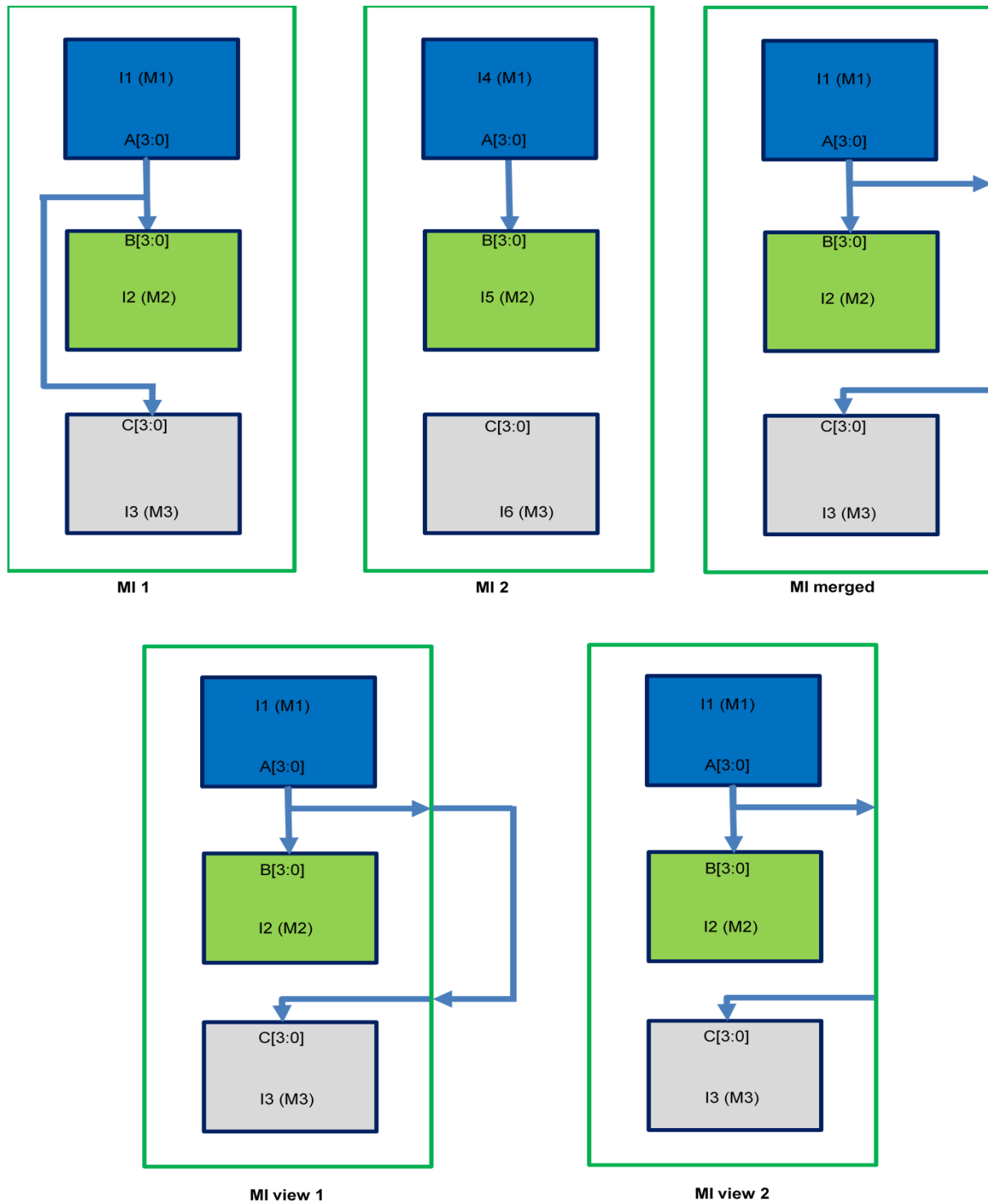


Figure 2-13 Multi-Instance Scenario 7



Arguments

The `set_group_mi_module` command has the following arguments:

`-group_instance {<space-separated-list-of-group-instances>}`

Specifies a space-separated list of group instances.

`-mi_module_name {<module-name>}`

Specifies the multi-instance module names for multi-instance grouping.

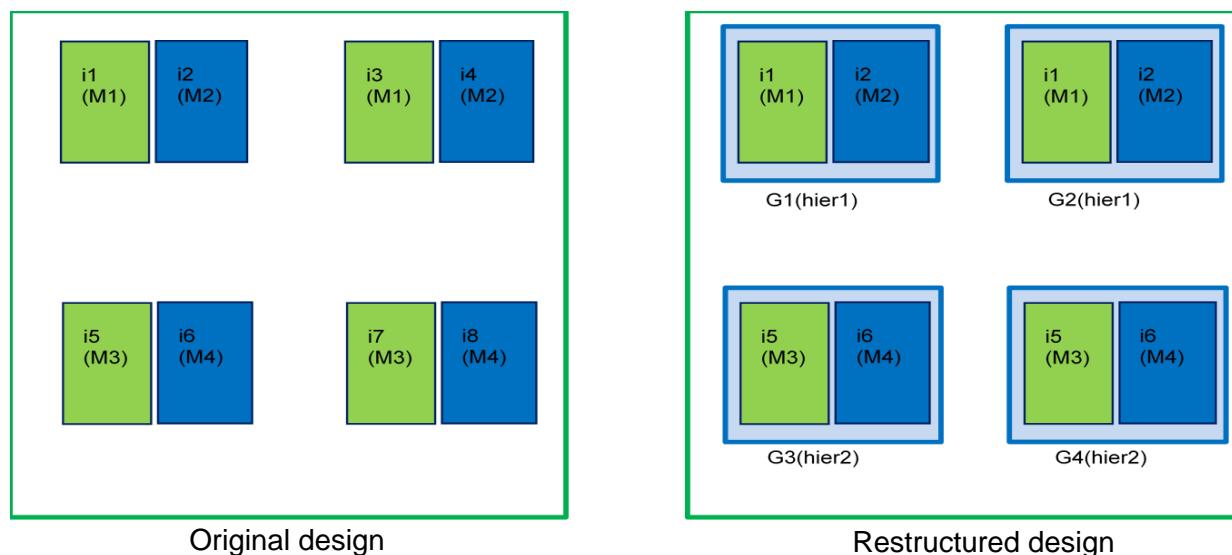
Example - Design Restructuring Using Multi-Instance Grouping

In the following example, i1 and i3 are instances of module M1, and i2 and i4 are instances of module M2. i5 and i7 are instances of module M3, and i6 and i8 are instances of module M4. The tool creates the multi-instance group modules, hier1 and hier2. The tool instantiates hier1 twice as G1 and G2, and hier2 as G3 and G4.

```
set_group_mi_module -group_instances {G1 G2} -mi_module_name hier1
group_design -instances {i1, i2} -name G1
group_design -instances {i3, i4} -name G2
```

```
set_group_mi_module -group_instances {G3 G4} -mi_module_name hier2
group_design -instances {i5, i6} -name G3
group_design -instances {i7, i8} -name G4
```

Figure 2-14 Design Restructuring Example Using Multi-Instance Grouping



set_group_options

Sets various options while grouping instances in the design

Usage

```
set_group_options [ -gen_ports <inmodule|outmodule|minport> ] [
  -tieoff_port <Y | N> ] [ -master_name <master-name> ] [ -port_naming
  <inmodule_based|perl_based|default> ] [ -port_naming_subroutine
  <perl-function> ] [ -interface_port_naming_subroutine
  <perl-subroutine> ] [ -stop_defaults <Y | N> ] [
  -export_unconnected <Y | N> ] [ -export_unconn_port_naming
  <Perl-function> ] [ -export_defaults <Y | N> ] [ -open <Y | N> ] [
  -inputs <Y | N> ] [ -outputs <Y | N> ] [ -reset <Y | N> ] [
  -rtl_library <library-name> ] [ -rtl_package <package-name> ] [
  -remove_holes <Y | N> ] [ -preserve_port_names <Y | N> ] [
  -preserve_interface_names <Y | N> ] [ -ordering_style 1 ]
```

Description

The `set_group_options` command sets various options while grouping instances in the design.

While grouping instances by using the [group_design](#) Tcl command, if you do not specify some arguments, GenSys picks the value for such arguments from this Tcl command.

Arguments

The `set_group_options` command has the following arguments:

`-gen_ports <inmodule|outmodule|minport|maxport>`

(Optional) Specifies the type of connections and necessary ports/ interfaces that will be created on the newly created subsystem. This argument can accept any of the following values:

Value	Description
inmodule	(Default) Specifies that in-module-based connections and necessary ports/interfaces will be created.

Value	Description
outmodule	Specifies that out-module-based connections, ports, or interfaces will be created. In this case, the number of ports on the group depends upon the number of distinct external ports connected with any instance terminal inside. Using this option minimizes the number of ports if driver is outside the group and maximizes the number of ports if driver is inside the group.
minport	Specifies that minimum number of adhoc ports are created on the group. In this case, ports get created based on the number of driver ports present (both external and internal to the group boundary) in the connectivity.

`-tieoff_port <Y | N>`

(Optional) Specifies whether tieoff ports should be created corresponding to the tieoff connections on the newly created sub-system.

By default, this option is set to N.

`-master_name <master-name>`

(Optional) Specifies the name of the master of group/subsystem.

If you do not specify this argument, GenSys considers the default master name for the group/subsystem as `<group-name>_Dsgn`.

Note:

If the master name of the group/subsystem conflicts with the name of an already existing design or component, GenSys reports an error and does not proceed for grouping.

`port_naming <default | inmodule_based | perl_based>`

(Optional) Specifies the port-naming convention to be followed. This argument can accept the following values:

- **default:** Specifies the default port naming convention in which a port is named as `<instance-name>_<pin-name>`. Here, `<instance-name>` and `<pin-name>` are derived from the instances (and their respective pins) inside or outside the group of an in-module or out-module based connection, respectively.

If the port name conflicts with an existing port name, GenSys assigns a unique port name as `<instance-name>_<pin-name>_<suffix>`, where `<suffix>` is a numerical value.

- `inmodule_based`: Specifies port names on the basis of instances inside the group. The port name is in the `<instance-name>_<pin-name>` format. If the port name conflicts with an existing port name, GenSys assigns a unique port name as `<instance-name>_<pin-name>_<suffix>`, where `<suffix>` is a numerical value.

This type of naming convention is useful if you want to name ports based on instances inside the group, but specify out-module-based connections or create maximum/minimum number of adhoc ports on the group.

- `perl_based`: This type of port naming convention enables you to specify your own Perl function by using the `port_naming_subroutine <perl-function>` -`port_naming_subroutine` argument. If you want to use the alias names of instances to name the ports of a group, you can provide alias names for instances as attributes inside the Perl function.

Note:

For IN-IN or OUT-OUT type of connections, a unique port gets created on the group.

Note:

While creating minimum/maximum ports on a group, it is important to identify the driver port. However, identification of the driver port is not possible if one side of the connection contains an INOUT port. Particularly for this type of connections, ports will be generated based upon instance-port combination inside the group.

```
port_naming_subroutine <perl-function>
```

(Optional) Specifies a Perl function that provides a logic to name ports of a group. This option is used when the `port_naming <default | inmodule_based | perl_based>` option is set to `perl_based`.

```
-interface_port_naming_subroutine <perl-function>
```

(Optional) Specifies the name of the Perl function that specifies the naming convention of the interface ports on the group instance. The first argument to the function is the interface connection type-item pointer inside the group and second argument is interface connection type-item pointer outside the group. For more details on interface port naming convention, refer to the *Specifying Interface Port Naming Conventions* topic of *GenSys User Guide*.

Note:

Currently, logical connections (nets) and cross-hierarchical connections are not handled.

```
-stop_defaults <Y | N>
```

(Optional) Specifies that the default values (tieoff or open) of unconnected ports of instances inside the group should not be propagated to the group boundary.

By default, the value of this argument is set to N, and the default values are propagated to the group boundary. Set the value of this argument to Y to stop the propagation of the default values to the group boundary.

`-export_unconnected <Y | N>`

(Optional) Specifies whether the unconnected terminals of all the instances within the group design should be exported to the group boundary as adhoc connections.

By default, the value of this argument is set to N, and the unconnected terminals are not exported to the group boundary as adhoc connections.

Set the value of this argument to Y to export all the unconnected terminals to the group boundary as adhoc connections. In this case, the name of the ports created corresponding to the open ports on the instance are inmodule based, and have the following name format:

`<instance-name>_<port-name>`

Ideally, this name should be unique and should not clash with any of the existing ports. However, if such clashes happen, GenSys uses those ports at the time of exporting.

For parameterized ports, exported ports are created on the top design with elaborated values of LSB and MSB.

`-export_unconn_port_naming <Perl-function>`

(Optional) Specifies the name of a Perl function using which you can control port names created by setting the `-export_unconnected` argument of this Tcl command to TRUE.

A pointer to an instance terminal having unconnected bits is passed as an argument to this Perl function.

`-export_defaults <Y | N>`

(Optional) Specifies whether ports with default connections (default tieoff or default open) should be exported to the group boundary.

By default, the value of this argument is set to Y, and such ports are exported to the group boundary.

Set the value of this argument to N if you do not want to export such unconnected ports with their default values to the group boundary.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-open <Y | N>`

(Optional) Specifies whether the intentional open ports should be exported to the sub-system boundary.

By default, the value of this argument is set to N, and all the intentional open ports are not be exported to the subsystem boundary.

Set the value of this argument to Y to export all the ports, which are marked as intentional open, to the subsystem boundary. In this case, the intentionally open ports will be exported

outside the boundary, and the new subsystem port that was created will be marked as intentional open.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-inputs <Y | N>`

(Optional) Specifies whether the unconnected input ports should be exported to the subsystem boundary.

By default, the value of this argument is set to Y, and the unconnected input ports are exported to the subsystem boundary by creating a port and connecting it to the unintentional input pin accordingly.

Set the value of this argument to N to turn off this default behavior.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-outputs <Y | N>`

(Optional) Specifies whether the unconnected open output and inout ports should be exported to the subsystem boundary.

By default, the value of this argument is set to Y, and the unconnected open output and inout ports are exported to the subsystem boundary by creating a port and connecting it to the unintentional output/inout pin accordingly.

Set the value of this argument to N to turn off this default behavior.

Note:

This argument is used only if the `-export_unconnected` argument is set to Y.

`-reset <Y | N>`

(Optional) Resets the values of all the arguments specified in this Tcl command.

Set the value of this argument to Y to reset values of all the arguments. By default, this argument is set to N.

`-rtl_library <library-name>`

(Optional) Specifies the library information of the component. This argument value is used when the `-component_decl` argument of the [set_rtl_option](#) Tcl command is set to FALSE. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE, the component declaration is not dumped in VHDL, but use library and package statements are dumped. In such case, the component declaration is picked from the library specified in this argument.

`-rtl_package <package-name>`

(Optional) Specifies the package information of the component. This argument value is used when the `-component_decl` argument of the [set_rtl_option](#) Tcl command is set to FALSE. When the `-component_decl` argument of the `set_rtl_option` Tcl command is set to FALSE, the component declaration is not dumped in VHDL, but use library and package statements are dumped. In such case, the component declaration is picked from the package specified in this argument.

`-remove_holes <Y | N>`

(Optional) Specifies whether holes created during the creation of a group hierarchy should be removed.

A hole gets created due to unconnected bits. For example, consider a vector port of five bits. Now if you connect two lower bits and two upper bits of this port and then run the [set_group_options](#) Tcl command, the middle bit remains unconnected resulting in a hole.

If this switch is provided, the `gen_ports` value is taken as [maxport](#), unless specified by you in this command.

By default, the value of this argument is N.

Example

In the following snippet, since the `remove_holes` argument is set to Y, the `{inst1_P1[6:2]}` port is made on the group boundary with the width 5.

```
new component compladd_port -name P1 -direction OUT -lsb 0 -msb
20add_port -name P2 -direction IN -lsb 0 -msb 20save compl
new design topadd_instance -name inst1 -master compladd_instance -name
inst2 -master compladd_instance -name inst3 -master compl
add_connection -instance inst1 -pin P1 -lsb 2 -msb 3-instance inst2 -pin
P2 -lsb 2 -msb 3
add_connection -instance inst1 -pin P1 -lsb 8 -msb10 -instance inst3
-pin P2 -lsb 8 -msb 10
group_design -instances { inst1 } -name grp_D-remove_holes Y -gen_pors
inmodule
```

If you set the `remove_holes` to N, the `{inst1_P1[10:2]}` port is with width 9. In this case, there is a hole from bit 4 to bit 7 inside the port, which means that these bits are not connected to anything.

`-preserve_port_names <Y | N>`

(Optional) When you set this argument to N, the name of ports created on the group boundary is of the format: `<instance-name>_<pin-name>`.

When the user sets this argument to Y, the name of the ports created on the group boundary is of the format: `<pin-name>`

However, if the group boundary already has a port of the same name, the instance name is prefixed before the name of the new port being created on the boundary.

By default, the value of this argument is N.

Example

In the following snippet, the p1 port is made on the group boundary.

```
new component compladd_port -name p1 -direction OUT -lsb 0 -msb 2save
new design dadd_port -name dport -direction OUT -lsb 0 -msb 2add_instance
-name inst1 -master compladd_connection -port dport -instance inst1 -pin
p1 group_design -instances { inst1 } -name grp_D -preserve_port_names Y
```

If you set the argument to N, the inst1_p1 port is created.

```
-preserve_interface_names <Y | N>
```

(Optional) When the user sets this argument to N, the name of interfaces created on the group boundary is of the format: <instance-name>_<interface-name>

When the user sets this argument to Y, the name of interfaces created on the group boundary is of the format: <interface-name>

However, if the group boundary already has an interface of the same name, the instance name is prefixed before the name of the new interface being created on the boundary.

By default, the value of this argument is N.

```
-ordering_style 1
```

(Optional) Controls names of parameters/instance pins during grouping and order of ports created on a hierarchy.

set_hierarchy_separator

Sets a new hierarchy separator used in the unification of hierarchical names for interface instance and registers/memories

Usage

```
set_hierarchy_separator <string>
```

Description

The set_hierarchy_separator command sets a new hierarchy separator used for the unification of hierarchical names for interface instance and registers/memories.

In addition, this command also performs the following functions:

- Clears the already created unique names, if any, and regenerates the unique names.
- Regenerates the memory map view with the new hierarchy separator.

By default, GenSys uses :: as the default hierarchy separator.

Examples

The following example sets @ as the hierarchy separator:

```
set_hierarchy_separator "@"
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case1 directory.

set_ifgenerate_options

Specifies modules in which memory swapping should be allowed

Usage

```
set_ifgenerate_options -preserve_component {<master-names>}
```

Description

The set_ifgenerate_options command specifies the modules (containing generate blocks) in which memory swapping should be allowed.

For details on the memory swapping feature, refer to the *Swapping Memories* section of the *GenSys User Guide*.

Arguments

The set_ifgenerate_options command has the following argument:

```
-preserve_component <master-name>
```

Specifies a space-separated list of modules (containing generate blocks) in which memory swapping should be allowed.

Example

The following example specifies the modules in which memory swapping should be allowed:

```
set_ifgenerate_options -preserve_component {M1 M2 M3}
```

set_interface_port

Adds an interface port to the parent design/component for the specified interface or interface instance

Usage

```
set_interface_port -name <logical-name> -actual_name <actual-name>
  [ -instance <interface-inst-name> ] [ -lsb <lsb> ] [ -msb <msb> ] [
  -direction <direction> ] [ -align <LSB | MSB> ] [ -type <port-type> ] [
  -description <description> ] [ -optional <O | R | RM | RS> ] [
  -registered <TRUE | FALSE> ] [ -default <value> ] [ -master ] [ -slave
  ] [ -system ] [ -lsb_bit <left | right> ] [ -net_name <net-name> ] [
  -net_prefix <net-prefix> ] [ -requiresDriver <true | false> ] [
  -driverType <any | clock | singleshoot> ] [ -logical_lsb <bit> ] [
  -logical_msb <bit> ]
```

Description

The `set_interface_port` command adds the specified interface port *<actual-name>* to the parent design/component using the logical name *<logical-name>* (specified in the parent interface library) for the specified interface instance.

Use the `set_interface_port` command to add logical ports to the interface library and to map logical interface ports to the actual ports of an interface instantiation in a design/component.

Notes

In general, an interface should be defined with the maximum range MSB and LSB for each bus. This presumably, is the defined range for the bus. When you use an interface on a component, you can use it with the same range specified for a given bus as was defined, or you can use it with a subset range specified. However, it is not normally reasonable (with exception below) to expand the range beyond the definition. This is because what is outside the definitions range is undefined. Therefore, the MSB and LSB range specification for an interface, is an important check that interfaces are being used correctly.

There is an exception where you can expand the range, usually used for a data bus or an address bus. You can do this by parameterizing the range. For example:

```
new interface test_if -short_name test_ifset_interface_port -name porta
-actual_name porta -direction INOUT -lsb 0 -msb {[P width]-1}save
test_ifcloseall
new component test_mod -short_name test_modadd_parameter -name width
-type INT -value 16add_interface -name test_if0 -interface_ref
test_ifset_interface_port -name porta -actual_name porta -instance
test_if0save test_mod
```

Arguments

The `set_interface_port` command has the following arguments:

`-name <logical-name>`

Specifies the name of the interface port that is to be mapped.

While mapping a logical port to the parent design/component, you must specify the logical name of an existing logical port in the interface library since you can only map existing logical ports (in the interface library) to the parent design/component.

The specification to set the msb and lsb value of the logical port is given below:

`-name <logical-name> \[<lsb>:<msb>\]`

`-actual_name <actual-name>`

Specifies the actual name by which the specified interface port is to be added to the parent design/component.

`-instance <interface-inst-name>`

(Optional) Specifies the name of the interface instance for which the specified interface port is to be mapped to the parent design/component.

Do not specify the `-instance` argument when you are adding an interface logical port to the interface library.

`-lsb <lsb>`

(Optional) Specifies the LSB of the bit-select/part-select of the specified interface port when you are adding only a selected set of bits of a vector interface port.

See [Specifying Values for Integer Arguments](#) for more details.

If you do not specify the `-lsb` argument and specify the `-msb` argument, the LSB of the bit-select/part-select is assumed to be 0.

Also, the `-lsb` argument should be used for the actual port and not for the logical port of the interface.

`-msb <msb>`

(Optional) Specifies the MSB of the bit-select/part-select of the specified interface port when you are adding only a selected set of bits of a vector interface port.

See [Specifying Values for Integer Arguments](#) for more details.

If you do not specify the `-msb` argument and specify the `-lsb` argument, the MSB of the port is assumed to be 0.

Note:

To add only a particular bit of a vector interface port, specify the bit index number as the value of both the `-lsb` and `-msb` arguments.

Also, the `-msb` argument should be used for the actual port and not for the logical port of the interface.

`-direction <direction>`

(Optional) Specifies the direction of the interface port. Direction can be IN (for input ports), OUT (for output ports), or INOUT (for inout ports).

By default, the port is assumed to be an inout port.

`-align <LSB | MSB>`

(Optional) Specifies the order of scalar-level connectivity for the specified bits of the interface port.

The default value of the `-align` argument is MSB and the interface port bits is connected MSB onwards.

`-type <port-type>`

(Optional) Specifies the port type of the interface logical port being added to the interface library as one of the following:

ADDR	ANALOG	CLK	CONFIG
CONTROL	CORE	CORE_INPUT	CORE_INPUT_FN
CORE_INPUT_T EST	CORE_OEN_FN	CORE_OEN_T EST	CORE_OUTPU T
CORE_OUTPUT _FN	CORE_OUTPUT_T EST	DATA	DFT

EVT	FUNCTION_IN	IO	IO_OEN
IO_PAD	IO_PULLEN	IO_SELECT	OSCCTL
PAD_INPUT	PAD_OEN	PAD_OUTPUT	PVT
PWRDN	RST	SCANIN	SCANOUT
SLEWRATE	TEST_IN	TIEOFF	PRIMARY_INP UT
PRIMARY_OUTP UT			

The PRIMARY_INPUT port type is considered to be the valid port type for the input pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells

The PRIMARY_OUTPUT port type is considered to be the valid port type for the output pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells.

You cannot modify the port type of an interface logical port being mapped to the parent design/component.

`-description <description>`

(Optional) Specifies the description of the interface logical port being added to the parent design/component or to the interface library.

This argument also accepts unicode characters.

`-optional <O | R | RM | RS>`

(Optional) Specifies whether the interface port is an optional port (O), a required port (R), a required port in master (RM), required port in slave (RS) while adding a new interface logical port to the interface library.

Optional logical ports need not be mapped to an actual port during interface instantiations. Unmapped required logical ports result in an error.

You cannot modify the optional type of an interface logical port being mapped to the parent design/component.

`-registered <TRUE | FALSE>`

(Optional) Specifies whether the actual port is registered (that is, driven by flip-flops).

By default, the port is assumed to be registered.

`-default <value>`

(Optional) Specifies the default value of the port being added to the parent design/component when the port does not have any connections defined.

`-master`

(Optional) Adds the port to the interface only if the interface is instantiated as master.

`-slave`

(Optional) Adds the port to the interface only if the interface is instantiated as slave.

`-system`

(Optional) Adds the port to the interface only if the interface is instantiated as system.

Note:

If you do not specify either of the `-master`, `-slave`, or `-system` switch, the port becomes a part of all master, slave, and system interfaces.

`-lsb_bit <left | right>`

(Optional) Specifies the bit endianness of the interface port. By default, the value of this argument is set to `right`, which means that while specifying a port (`P[<left>:<right>]`), the right-most bit is considered as LSB and the left-most bit is considered as MSB. This type of endianness is also known as little bit-endian.

If you set the value of this argument as `left`, GenSys considers the left-most bit of the interface port as LSB and the right-most bit of the port as MSB. This type of endianness is also known as big bit-endian.

Note:

The bit endianness specified by this Tcl command overrides the default bit endianness specified by the [set_lsb_bit](#) Tcl command for the specified interface port.

`-net_name <net-name>`

(Optional) Specifies the net name for the tieoff connection (default/temporary/forced). This net name appears in the generated RTL (Verilog/VHDL).

If multiple connections (on different or same pin) have same net name, GenSys creates a vector net. The size of this vector net is same as the number of connections having the same net name. If unique net name is specified for each tieoff connection, GenSys creates a scalar net.

`-net_prefix <net-prefix>`

(Optional) Specifies the net prefix for the tieoff connection (default/temporary/forced). The net prefix specified by this argument gets prepended to the net name, and this net name then appears in the generated RTL (Verilog/VHDL).

If multiple connections (on same pin) have same net prefix, GenSys creates a vector net. The size of this vector net is same as the number of connections having the same net prefix. If unique net prefix is specified for each tieoff connection, GenSys creates a scalar net.

`-requiresDriver <true | false>`

(Optional) Specifies if the port being added requires a driver when used in a complete design.

`-driverType <any | clock | singleshot>`

(Optional) Specifies the driver type of the port being added.

`-logical_lsb <bit>`

(Optional) Specifies the slice of the logical port.

`-logical_msb <bit>`

(Optional) Specifies the slice of the logical port.

Examples

Example 1 - Add Port

The following example adds port named L01 (using the `set_interface_port` command):

```
set_interface_port -name L01 -direction IN -type DATA -description "First Logical Port" -registered TRUE
```

The following example adds port named A01 to the component for the interface port L02 with [0:31] bits and default value 0:

```
set_interface_port -name L02 -actual_name A01 -direction IN -type ADDR  
-description "Second Logical Port" -lsb 0 -msb 31 -align LSB  
-optional 0 -default 0
```

Note:

To refer to the testcase of the above examples, go to `$SPYGLASS_HOME/ examples/ Genesis/case1` directory.

Example 2 - Mapping of a Single Logical Interface Port to Multiple Physical Ports

In this example, the bits of the logical port `tt` are mapped to five different physical ports, `di1` to `di5`. The 0 (zero) bit is mapped to `di1` and similarly the other bits are mapped according to the value of the corresponding `-actual_name` argument.

```
set_interface_port -instance tinterf1 -name tt -logical_msb 0
-logical_lsb 0 -actual_name di1
set_interface_port -instance tinterf1 -name tt -logical_msb 1
-logical_lsb 1 -actual_name di2
set_interface_port -instance tinterf1 -name tt -logical_msb 2
-logical_lsb 2 -actual_name di3
set_interface_port -instance tinterf1 -name tt -logical_msb 3
-logical_lsb 3 -actual_name di4
set_interface_port -instance tinterf1 -name tt -logical_msb 4
-logical_lsb 4 -actual_name di5
```

set_ipxact_busdef_vlnv

Sets VLNV for a bus definition of the currently active interface

Usage

```
set_ipxact_busdef_vlnv [ -name <name> ] [ -version <version> ] [
-vendor <vendor> ] [ -library <library> ]
```

Description

The `set_ipxact_busdef_vlnv` command sets VLNV for a bus definition of the currently active interface.

This VLNV information can be imported and exported in all formats, such as MRS, Tcl, or IP-XACT.

Arguments

The `set_ipxact_busdef_vlnv` command has the following arguments:

`-name <name>`

(Optional) Specifies the name of the bus definition.

`-version <version>`

(Optional) Specifies the version of the bus definition.

`-vendor <vendor>`

(Optional) Specifies the vendor of the bus definition.

`-library <library>`

(Optional) Specifies the library of the bus definition.

set_ipxact_version

Specifies the IP-XACT version in which the objects should be saved

Usage

```
set_ipxact_version <version-number>
```

Description

Specifies the IP-XACT version in which objects should be saved.

For example, if you specify the version as `ieee_2009`, GenSys saves objects in the IP-XACT version 1685-2009.

By default, GenSys saves objects in the IP-XACT version 1.4.

set_library_path

Specifies the library path.

Usage

```
set_library_path { [ -r ] [ -append ] <dir-name> | <file-name> [
-files {<space-separated-glob-patterns>} <path-list> ] [
-reference <dir-name> ] [ -reset_reference ] }
```

Description

Specifies the library files (XML or Tcl files) with absolute or relative path.

You can specify any of the following:

- A directory name so that all `.xml`, `.tcl`, and `.cmd` files in the specified directory are read in to the library.

A directory name can be an absolute path or a path relative to the current working directory. You can use wildcards that are supported by the Operating System while specifying the directory names.

- A file name so that the specified file is loaded into the library.

You can use wildcards that are supported by the Operating System while specifying the file names.

- A combination of both.
- A space-separated list of glob patterns to specify the type of files to be loaded in the library.

Note:

If the VLNV of the objects in the library files match, GenSys loads the objects selectively. For more details, refer to the [Additional Details](#) section.

You can specify the `set_library_path` command in `.gensys.setup` file to select the specified library, by default, every time GenSys is loaded.

Use the `-reference` and `-reset_reference` arguments to define a directory as the reference library, if you do not intend to do any changes in the IP's present in that directory.

Arguments

The `set_library_path` command has the following arguments:

`-r`

(Optional) Specifies that the library files need to be loaded recursively. Files from all the specified directories and all its sub directories are loaded into the library.

`-append <dir-name> | <file-name>`

(Optional) Specifies that the files from the specified directory or the files whose names are provided with this argument should be appended to the files already existing in the library.

`-files <list-of-glob-patterns> <path-list>`

(Optional) Specifies the type of files to be loaded in the library. You can specify the type of files as a space-separated list of glob patterns. Such files are located in the path, `<path-list>`.

If the `-files` option is not specified, by default GenSys loads `*.xml`, `*.tcl`, and `*.cmd` files from the specified path.

File types can be enclosed in `""` or `{}`.

`-reference <dir-name>`

(Optional) Specifies the directory name, absolute path or path relative to the current working directory. If a valid reference directory is set, any changes performed to the IP's present in this reference directory are not be reflected after the save operation. If you do an [export_all/export_data](#) in the IPXACT format the IP's loaded from the reference directory are not be exported.

By default, the value of this argument is empty.

`-reset_reference`

(Optional) Resets the reference library path. All files are saved after the reference library path is reset.

Additional Details

If the VLNV of objects in multiple library files are same, GenSys loads the objects from these files selectively. The following cases explain how GenSys loads objects with same VLNV, depending on the location and format of library files:

Case 1

If the library files are located in the same directory and have same format, objects from the file with the latest date and time stamp are loaded.

For example, a directory, D1, contains two files, A.xml and B.xml. Both are in MRS format, but B.xml has a latest date and time stamp. For the objects that are in both files and have same VLNV, the objects are loaded from B.xml only.

Case 2

If the library files are located in the same directory but are of different formats, their date and time stamp is checked. If the time stamp of objects from the first library is older than that of the object being read in from the next library, objects defined in both the formats are loaded. Otherwise, only the objects from the library that is specified first are loaded.

For example, a directory, D1, contains two files, A.xml and B.xml. A.xml is in MRS format, but B.xml is in IP-XACT format. Assume that the objects from A.xml are loaded first. If the objects in both files have same VLNV and the date and time stamp of B.xml is less than that of A.xml, only the objects from A.xml are loaded.

Case 3

If the library files are located in different directories, objects from the first directory are loaded.

For example, a directory, D1, contains A.xml and directory, D2, contains B.xml. If the objects in these files have same VLNV and if A.xml is loaded first, only the objects from A.xml are loaded.

Case 4

If the objects in different library files have same VLNV but were defined in both Tcl and non-Tcl formats, the files with Tcl format are ignored. In this case, the objects are not taken from the first directory, as in example of case 3 above, if the files in the first directory are in Tcl format.

Examples

Example 1

```
set_library_path -files {*.abc *.c*} .
```

The above command loads the files matching the pattern *.abc and *.c* from the current working directory. Similar result can be obtained by specifying any of the following commands:

```
set_library_path -files {"*.abc" "*.cdf"} set_library_path -files {*.abc  
*.cdf} set_library_path -files {{*.abc} {*.cdf}} set_library_path -files  
{"*.abc" *.cdf}
```

set_log_dir

Sets the directory for storing log files.

Usage

```
set_log_dir <dir-name>
```

Description

The `set_log_dir` command specifies a directory to store log files. GenSys stores all the log files in the specified directory.

The `set_log_dir` command can be given in `.gensys.setup` file. If not given, the default behavior is applied and a default directory, `log`, gets created in the current directory and all the log files are stored in it.

When the `set_log_dir` command is used, all the log files are moved from the current directory to the specified directory, if exists. In addition, it also removes the log directory, if it is the current directory and was created by GenSys in the current session.

The new log entries are written in the directory specified by the `set_log_dir` command.

set_lsb_bit

Sets the default bit endianness for all the ports in the current session

Usage

```
set_lsb_bit <LEFT | RIGHT>
```

Description

The `set_lsb_bit` command sets the bit endianness for all the ports in the current session.

By default, the bit endianness is set to `RIGHT`, which means that the left-most bit is considered as `MSB` and the right-most bit is considered as `LSB`. This type of endianness is known as little bit-endian (`MSB >= LSB`).

If you set the bit endianness as `LEFT`, the right-most bit is considered as `MSB` and the left-most bit is considered as `LSB`. This type of endianness is also known as big bit-endian (`LSB >= MSB`).

set_messaging_mode

Enables or disables reporting of messages in an incremental mode

Usage

```
set_messaging_mode -incremental <TRUE | FALSE>
```

Description

The `set_messaging_mode` command enables or disables reporting of messages in an incremental mode under the Messages tab of the Command/Log Window.

By default, this argument is set to `FALSE`, and message are reported in a chronological order.

set_msg_severity

Sets the severity of a message

Usage

```
set_msg_severity -id <msg-id> [-sev <DEBUG | INFO | WARNING | ERROR |  
TCL_ERROR>] [-reset]
```

Description

The `set_msg_severity` command changes the severity of a message for a session or resets the severity back to the GenSys default severity.

Arguments

The `set_msg_severity` command has the following arguments:

`-id <msg-id>`

Specifies the message ID that corresponds to the message that requires a change in severity.

`-sev <severity>`

(Optional) Defines the new severity that is to be assigned for the message ID specified in the `-id` argument. You can specify a severity as `DEBUG`, `INFO`, `WARNING`, `ERROR`, or `TCL_ERROR`.

`-reset`

(Optional) Resets the severity to the default setting, as defined in GenSys.

set_netlist_dir

Sets the netlist directory for dumping Verilog/VHDL files

Usage

```
set_netlist_dir <dir-name>
```

Description

The `set_netlist_directory` command specifies a netlist directory for storing Verilog/VHDL files.

When you run the Verilog/VHDL generator, GenSys creates the Verilog/VHDL directory under the netlist directory specified by this command. All Verilog/VHDL files are then dumped in the Verilog/VHDL directory of the specified netlist directory.

Also see [Preference of a Default Directory for Storing Verilog Files](#).

set_netname_option

Assigns a net name to specific types of connections

Usage

```
set_netname_option -netname <pattern> [ -connection_type  
<connection-type> ]
```

Description

The `set_netname_option` command assigns a global net-naming convention for different types of connections.

Corresponding to every connection, there is corresponding wire name in RTL. This wire name is the net name set on the connection. This TCL command helps you to let GenSys know how to name different wires in the RTL.

Priority of Net Names

Net names are assigned priorities in the following order:

1. User-specified Perl subroutine for name control.

For details, refer to the *Creating a Generator for Changing Net Names in Generated RTL* topic in *GenSys Customization Guide*.

2. Net name specified through the -name <net-name> argument of the [add_connection](#) command or net name specified by the [add_netname](#) command.
3. Net name specified through the set_netname_option command.
4. Default net naming convention internally used by GenSys.

Every connection type in GenSys can be viewed as adhoc or tieoff from RTL point of view.

For adhoc connections, the default naming pattern is <driver-instance-name>_<driver-pin-name>. This pattern is generated for connection types, such as interface connections and logical connections as these connections are viewed as adhoc connections from RTL point of view.

For tie-off connections, GenSys does not generate any net in the RTL. Rather, it uses the tie-off value directly.

Arguments

The set_netname_option command has the following arguments:

-netname <pattern>

Specifies a pattern based on which net names should be generated for a particular type of connection.

To create a pattern, use a combination of the following keywords:

Table 2-1

Keyword	Description
#si	Represents a source instance name.
#sp	Represents a source pin name. If you specify the connection type as interface in the -connection_type argument, this keyword represents a source interface name.

Table 2-1

Keyword	Description
#sd	Represents a source port direction.
#xyz	Represents any string. For example, in the following pattern, atrenta is a string: <code>#si_#sp_#sd_atrenta_#di_#dp</code> The underscore (_) in the pattern is not a delimiter. It is just a character that is used in a net name.
#di	Represents a destination instance name.
#dp	Represents a destination pin name. If you specify the connection type as interface in the <code>-connection_type</code> argument, this keyword represents a destination interface name.
#dd	Represents a destination port direction.
#msb	Represents MSB.
#lsb	Represents LSB.
#d	Represents a unique digit starting from 0. Use this keyword if you want a unique digit to be added for conflicting net names. For example, consider the following command: <code>set_netname_option -netname n%d -connection_type all</code> The above command implies that nets in the generated RTL will be named as n0, n1, n2, n3, and so on.

Note:

If you specify a pattern in upper-case, the corresponding name generated is in upper-case. For example, if you specify #SI, the source instance name is written in upper-case.

`-connection_type <connection-type>`

Specifies the type of connection for which a global net-naming convention is used.

You can specify the following values to this argument:

tieoff	adhoc	open	intentional_ope	interface	all
			n		

Examples

Example 1

Consider that you specify the following Tcl commands:

```
new component compladd_port -name P1 -direction IN -lsb 0 -msb 10add_port
-name P2 -direction OUT -lsb 0 -msb 10
new design topadd_port -name P3 -direction INadd_instance -name I1
-master compladd_connection -instance I1 -pin P1 -lsb 5 -msb 5-port
P3add_connection -instance I1 -pin P2 -lsb 5 -msb 9-noconn
set_netname_option -connection_type intentional_open -netname
#si_open_#sd
set_netname_option -connection_type open-netname
#si_#dd_unintentionalOpen_#sd
run GenerateVerilog
```

When you run the above Tcl commands, GenSys assigns net names to the intentional open and open connections based on the pattern specified by the `set_netname_option` Tcl command.

Following is the RTL generated in this case (generated net names in bold):

```
module compl (P1,                P2);input  [10:0]  P1;output [10:0]
P2;endmodule
module top (P3);input          P3;wire    [4:0]
I1_unintentionalOpen_in_0;wire  [4:0]  I1_unintentionalOpen_in_1;wire
[4:0]  I1_unintentionalOpen_out_0;wire  [4:0]  I1_open_out_5;wire
I1_unintentionalOpen_out_5;    compl  I1 (
.P1({I1_unintentionalOpen_in_1,                P3,
I1_unintentionalOpen_in_0}), //I:11
.P2({I1_unintentionalOpen_out_5,
I1_open_out_5,                I1_unintentionalOpen_out_0}) //O:11
);endmodule
```

Example 2

Consider that you specify the following Tcl commands:

```
set_netname_optionnetname
#sp_#si_#SD_NetName_#msb_#lsb_#dd_#dp_XYZ_#d-connection_type adhoc
set_netname_option-netname
#sp_#si_#SD_NetName_#msb_#lsb_#dd-connection_type tieoff
```

```

new component compadd_port -name p1 -direction IN -msb 5 -lsb 0 add_port
-name p3 -direction IN -msb 5 -lsb 0 add_port -name p2 -direction OUT -msb
5 -lsb 0 add_port -name p4 -direction OUT -msb 5 -lsb 0
new design desadd_instance -name comp0 -master compadd_instance -name
comp1 -master compadd_port -name dp1 -direction IN -msb 5 -lsb 0 add_port
-name dp2 -direction OUT -msb 5 -lsb 0 add_connection -instance comp0 -pin
p2 -msb 5 -lsb 0 -instance comp1 -pin p1 -msb 5 -lsb 0 add_connection
-instance comp0 -pin p4 -msb 5 -lsb 0 -instance comp1 -pin p3 -msb 5 -lsb
0 add_connection -instance comp0 -pin p1 -tieoff 1
run GenerateVerilog

```

When you run the above Tcl commands, GenSys assigns net names to the adhoc and tieoff connections based on the pattern specified by the `set_netname_option` Tcl command.

Following is the RTL generated in this case (generated net names in bold):

```

module comp (p1,          p3,          p2,          p4); input
[5:0] p1; input [5:0] p3; output [5:0] p2; output [5:0]
p4; endmodule
module des (dp1,          dp2); input [5:0] dp1; output [5:0]
dp2; wire [5:0] p2_comp0_OUT_NetName_5_0_in_p1_XYZ; wire [5:0]
p4_comp0_OUT_NetName_5_0_in_p3_XYZ; wire [5:0]
p1_comp0_IN_NetName_5_0_in; wire [5:0]
UNINTENTIONAL_OPEN_comp0_p3_0; wire [5:0]
UNINTENTIONAL_OPEN_comp1_p2_0; wire [5:0]
UNINTENTIONAL_OPEN_comp1_p4_0; comp comp0 (
.p1(p1_comp0_IN_NetName_5_0_in), //I:6
.p3(UNINTENTIONAL_OPEN_comp0_p3_0), //I:6
.p2(p2_comp0_OUT_NetName_5_0_in_p1_XYZ), //O:6
.p4(p4_comp0_OUT_NetName_5_0_in_p3_XYZ) //O:6); comp
comp1 ( .p1(p2_comp0_OUT_NetName_5_0_in_p1_XYZ), //I:6
.p3(p4_comp0_OUT_NetName_5_0_in_p3_XYZ), //I:6
.p2(UNINTENTIONAL_OPEN_comp1_p2_0), //O:6
.p4(UNINTENTIONAL_OPEN_comp1_p4_0) //O:6); assign
p1_comp0_IN_NetName_5_0_in = {6'h01}; endmodule

```

set_output_dir

Sets the GenSys output directory

Usage

```
set_output_dir <dir-name>
```

Description

Specifies the directory *<dir-name>* where GenSys outputs are to be stored.

By default, the output directory for GenSys outputs is the current directory (from where GenSys has been invoked.)

You can also set the output directory in the following other ways:

1. Using the `-outdir` command-line option while invoking GenSys
2. Selecting the File > Set Output Directory menu option in GenSys GUI.

Examples

The following example sets the output directory as `output_dir`:

```
set_output_dir ./Work_dir/output_dir
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/examples/Genesis/case6` directory.

set_packaging_options

Sets IP packaging options

Usage

```
set_packaging_options [ -strict_ipxact <TRUE | FALSE> ] [
-file_relative_path <path> ] [ -file_env_var <file-env-variable> ] [
-file_env_value <file-env-value> ] [ -package_pragmas <TRUE | FALSE> ]
```

Description

The `set_packaging_options` command sets IP packaging options.

Arguments

The `set_packaging_options` command has the following arguments:

`-strict_ipxact <TRUE | FALSE>`

(Optional) By default, this argument is set to FALSE.

When you set this argument to TRUE then during IP packaging if the tool is not able to convert an RTL/TCL expression to an equivalent XPath expression, the empty dependency expression is exported and an error message is reported .

`-file_relative_path <path>`

(Optional) Specifies the relative path of a file set.

When you specify this argument, file name stored in a file set is with respect to this relative path if the absolute path of the file starts with this value (*<path>*).

`-file_env_var <file-env-variable>`

(Optional) Specifies the name of an environment variable used to specify the path of a file set.

`-file_env_value <file-env-value>`

(Optional) Specifies the value of the environment variable that is specified by the `-file_env_var <variable>` argument of this Tcl command.

When you specify this argument, file name stored in a file set is with respect to the specified environment variable value, that is `$<file-env-var>`, if the absolute file path starts with this environment variable value.

Note:

You must specify the `-file_env_var` and `-file_env_value` arguments of this Tcl command together. If you do not specify either one of these arguments, GenSys ignores the other argument.

`-package_pragmas <TRUE | FALSE>`

(Optional) When this argument is set to TRUE, GenSys infers pragma information for ports defined under synopsys `translate_off/on` and synopsys `synthesis_off/on` pragma.

Note:

This functionality works only for the IP packaging flow, that is, when you specify the `-component` argument with the [load_rtl](#) command.

Examples

Consider the following absolute path of a file:

```
/usr/opt/vhdl/examples/test.vhd
```


Now, the following examples show how the resultant path of a file set is generated based on the usage of various arguments of this Tcl command.

Example 1

```
set_packaging_option -file_env_value "/usr/opt/vhdl"  
-file_relative_path "/usr/opt" -file_env_var "FILE_PATH"
```

In this case, the resultant path is \$FILE_PATH/examples/test.vhd.

Example 2

```
set_packaging_option -file_env_value "/usr/opt/vhdl"  
-file_relative_path "/usr/opt"
```

In this case, the resultant path is vhdl/examples/test.vhd. This path is generated because the -file_env_var argument is not specified with the -file_env_value argument. So the -file_env_value argument is ignored and the path is generated based on the specified relative path.

Example 3

```
set_packaging_option -file_env_value "/usr/opt/vhdl"  
-file_env_var "FILE_PATH"
```

In this case, the resultant path is \$FILE_PATH/examples/test.vhd.

set_parameter

Sets a parameter value for a component, design or component instance

Usage

```
set_parameter  
-component <comp-name | dsgn-name>  
  | -instance <inst-name>  
(-name <param-name>  
-value <param-value> )*  
[ -macro_visibility <macro-condition> ]
```

Description

The `set_parameter` command sets the parameter value for a component, design, or an instance.

Arguments

The `set_parameter` command has the following arguments:

`-component <comp-name | dsgn-name>`

Specifies the name of the component or design whose parameter is being set.

`-instance <inst-name>`

Specifies the name of the component instance where the component parameter is being set.

`-name <param-name>`

Specifies the name of the component parameter whose value is being set.

`-value <param-value>`

Specifies the parameter value or the parameter field value.

For enumerator-type parameters, you can specify one of the valid values only.

For example:

```
set_parameter -instance inst1 -name P1 -value 10
```

`-macro_visibility <macro-condition>`

(Optional) Specifies a macro condition in which parameter override should be visible in the generated Verilog RTL.

Based on the specified macro condition, parameter override is placed under appropriate Verilog conditional pre-processor directives, such as ``ifdef` in generated Verilog RTL.

Refer to [Example 4](#).

Examples

The following example sets the value of parameter, PP1, of component, comp1, to MYVALUE:

```
set_parameter -name PP1 -component comp1 -value MYVALUE
```

set_physical_units

Specifies design masters whose instances should contain the clones

Usage

```
set_physical_units
  -names <list-of-design-master-names>
  [ -append ]
```

Description

The set_physical_units command specifies the names of design masters whose instance should contain clones selected in the Hierarchy Manipulation widget.

Arguments

The set_physical_units command has the following arguments:

-names <list-of-design-master-names>

Specifies a space-separated list of design masters.

The instance hierarchy of these masters is considered by the -to {<hierarchies-containing-clones> argument of the clone_ip command.

-append

(Optional) Appends the design master to the existing list of masters whose instance should be cloned.

If you do not specify this argument, design masters previously specified by this command are overwritten with the new design masters specified by this command.

set_port

Update the fields of the currently active port (either of active design or component).

Usage

```
set_port
  [ -name <port-name> ]
```

```

[ -direction <IN | OUT | INOUT> ]
[ -port_type <port-type> ]
[ -lsb <lsb> ]
[ -msb <msb> ]
[ -align <LSB | MSB> ]
[ -short_name <short-name> ]
[ -clock_rate <clk-rate> ]
[ -power_domain <pd-name> ]
[ -voltage <voltage-value> ]
[ -size <size> ]
[ -default_value <value> ]
[ -control_clock <clk-name> ]
[ -control_reset <rst-name> ]
[ -description <description> ]
[ -open <TRUE | FALSE> ]
[ -optional <O | R | RM | RS> ]
[ -visibility <visibility-condition> ]
[ -vhdl_port_type STD_LOGIC | STD_ULOGIC | BIT |
  SIGNED | UNSIGNED | <complex-type> ]
[ -lsb_bit <left | right> ]
[ -net_name <net-name> ]
[ -net_prefix <net-prefix> ]

```

Description

The `set_port` command update the fields of the currently active port (either of active design or component).

Arguments

The `set_port` command has the following arguments:

`-name <port-name>`

(Optional) Specifies the new name (string) of the active port. If you do not specify this argument, the name of the active port remains unchanged.

`-direction`

(Optional) Specifies the new direction of the port as IN (for input ports), OUT (for output ports), or INOUT (for inout ports).

If you do not specify this argument, the port direction remains same as was specified when that port was created by using the [add_port](#) Tcl command.

`-port_type <port-type>`

(Optional) Specifies the new port type as one of the following:

ADDR	ANALOG	CLK	CONFIG
CONTROL	CORE	CORE_INPUT	CORE_INPUT_FN
CORE_INPUT_TEST	CORE_OEN_FN	CORE_OEN_TEST	CORE_OUTPUT
CORE_OUTPUT_FN	CORE_OUTPUT_TEST	DATA	DFT
EVT	FUNCTION_IN	IO	IO_OEN
IO_PAD	IO_PULLEN	IO_SELECT	OSCCTL
PAD_INPUT	PAD_OEN	PAD_OUTPUT	PVT
PWRDN	RST	SCANIN	SCANOUT
SLEWRATE	TEST_IN	TIEOFF	PRIMARY_INPUT
PRIMARY_OUTPUT			

The PRIMARY_INPUT port type is considered to be the valid port type for the input pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells

The PRIMARY_OUTPUT port type is considered to be the valid port type for the output pin of the SBSR/MBSR cell. This pin will be used to connect to PAD/MUX cells.

If you do not specify this argument, the port type remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-lsb <lsb>`

(Optional) Specifies the new LSB of the port.

Use the -lsb argument to specify the LSB of vector ports.

See [Specifying Values for Integer Arguments](#) for more details.

If you do not specify this argument, the port LSB remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-msb <msb>`

(Optional) Specifies the new MSB of the port.

Use the `-msb` argument to specify the MSB of vector ports.

See [Specifying Values for Integer Arguments](#) for more details.

If you do not specify this argument, the port MSB remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-align <LSB | MSB>`

(Optional) Specifies the new order of scalar-level connectivity for the (vector) port.

If you do not specify this argument, the port alignment remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-short_name <short-name>`

(Optional) Specifies the new short name of the port.

If you do not specify this argument, the port short name remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-clock_rate <clk-rate>`

(Optional) Specifies the new clock rate of the port.

If you do not specify this argument, the clock rate of the port remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-power_domain <pd-name>`

(Optional) Specifies the new power domain of the port.

If you do not specify this argument, the power domain of the port remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-voltage <voltage-value>`

(Optional) Specifies the new voltage value for the power domain of the port.

If you do not specify this argument, the voltage value remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-size <size>`

(Optional) Specifies the new the size of the port.

Use the `-size` argument to specify the size of vector ports when you do not want to specify the port size in terms of LSB/MSB using the `-lsb/-msb` arguments. Then, the port LSB is assumed to be 0 and port MSB is assumed to be (size-1).

If you have specified the LSB/MSB values of the port using the `-lsb/-msb` arguments, the size value is automatically calculated and displayed. You cannot then modify the size value using the `-size` argument.

If you do not specify this argument, the port size remains the same as was specified when that port was created by using the `add_port` Tcl command.

`-default_value <value>`

(Optional) Specifies the new default value of the port.

If there is no connection specified for the port, the port is assumed to be tied to the specified default value.

For a component, the default port value is calculated on the basis of the following precedence:

- If actual port has a default value, it will get the highest precedence.
- If the actual port does not have a default value and no default value is specified while mapping to the interface port through the `set_interface_port` Tcl command, the default value of the logical port is set as the default value of the actual port.
- If the actual port does not have default value but a default value is specified while mapping to the interface port through the `set_interface_port` Tcl command, that default value is set for the actual port.

If you do not specify this argument, the port default value remains the same as was specified when that port was created by using the `add_port` Tcl command.

`-control_clock <clk-name>`

(Optional) Specifies the new name of the control clock for the port.

The specified control clock must be already defined for the design or component.

If you do not specify this argument, the control clock remains the same as was specified when that port was created by using the `add_port` Tcl command.

`-control_reset <rst-name>`

(Optional) Specifies the new name of the control reset for the port.

The specified control reset must be already defined for the design or component.

If you do not specify this argument, the control reset remains the same as was specified when that port was created by using the `add_port` Tcl command.

`-description <description>`

(Optional) Specifies the new description of the port.

This argument also accepts unicode characters.

If you do not specify this argument, the port description remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-open <TRUE | FALSE>`

(Optional) Specifies whether the port can be left unconnected in the RTL being generated.

If you do not specify this argument, the port connection status remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-optional <O | R | RM | RS>`

(Optional) Specifies whether the port is an optional port (O), a required port (R), a required port in master (R), or required port in slave (RS).

If you do not specify this argument, the port property remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-visibility <visibility-condition>`

(Optional) Specifies the comma-separated string of conditions under which the object would be visible to the generators.

If you do not specify this argument, the visibility conditions remain the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-vhdl_port_type`

(Optional) Specifies the new VHDL type of a port.

If you do not specify this argument, the port VHDL type remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-lsb_bit <left | right>`

(Optional) Specifies the new bit endianness of the port.

If you do not specify this argument, the port bit endianness remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

`-net_name <net-name>`

(Optional) Specifies the new net name for the tieoff connection (default/temporary/forced).

If you do not specify this argument, the net name remains the same as was specified when that port was created by using the [add_port](#) Tcl command.


```
-net_prefix <net-prefix>
```

(Optional) Specifies the net prefix for the tieoff connection (default/temporary/forced).

If you do not specify this argument, the net prefix remains the same as was specified when that port was created by using the [add_port](#) Tcl command.

Example

The following Tcl command updates the name of the currently active port to p2:

```
set_port -name p2
```

Rest all the fields of this port would remain the same.

set_report_dir

Sets the report directory.

Usage

```
set_report_dir <dir-name>
```

Description

The set_report_dir command specifies a directory in which all reports generated by GenSys are saved.

By default, all reports are saved in the report directory.

Examples

The following example sets the directory named report_dir (located in ./Work_dir directory) as the report directory:

```
set_report_dir ./Work_dir/report_dir
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case6 directory.

set_rtl_option

Sets the RTL options

Usage

```
set_rtl_option {
  [ -rtl_comment <TRUE | FALSE> ]
  [ -defval_rtl_dump <TRUE | FALSE> ]
  [ -vectorize_net <TRUE | FALSE> ]
  [ -print_conn_comment <TRUE | FALSE> ]
  [ -delta_delay <TRUE | FALSE> ]
  [ -header_template <file-name> ]
  [ -component_decl <TRUE | FALSE> ]
  [ -uniquify_defval_rtl_dump <TRUE | FALSE> ]
  [ -optimized_nets <TRUE | FALSE> ]
  [ -wrapper_creation <TRUE | FALSE> ]
  [ -uniquify_tie_rtl_dump <TRUE | FALSE> ]
  [ -dump_timescale <TRUE | FALSE> ]
  [ -default_tieoff_prefix <default-prefix> ]
  [ -forced_tieoff_prefix <forced-prefix> ]
  [ -temp_tieoff_prefix <temp-prefix> ]
  [ -useSupplyName <TRUE | FALSE> ]
  [ -rtl_supply0_name <supply0-name> ]
  [ -rtl_supply1_name <supply1-name> ]
  [ -black_box_dump < TRUE | FALSE> ]
  [ -preserve_lsb_msb <TRUE | FALSE> ]
  [ -allow_out_port_read <TRUE | FALSE> ]
  [ -archName <architecture-name-format> ]
  [ -dumpVHDLSingleFile <TRUE | FALSE> ]
  [ -vhdl_port_assign <TRUE | FALSE> ]
  [ -config_spec_dump <TRUE | FALSE> ]
  [ -createGatingMux <TRUE | FALSE> ]
  [ -create_stubs <TRUE | FALSE> ]
  [ -preserve_comments <TRUE | FALSE> ]
  [ -preserve_floating_net_name <TRUE | FALSE> ]
  [ -export_binary_value <VERILOG | FALSE> ]
  [ -export_prj_file <TRUE | FALSE> ]
  [ -generate_unique_filelists <TRUE | FALSE> ]
  [ -macros_in_wire_indexes <TRUE | FALSE> ]
}
```

Description

The set_rtl_option command sets various RTL options required for RTL generation.

Arguments

The `set_rtl_option` command has the following arguments:

`rtl_comment` <TRUE | FALSE>

(Optional) Specifies if comments should be included in the generated RTL around instance dumping. When this is set to TRUE (default), GenSys dumps terminal direction, width with instance port map. Also, it groups terminals belonging to the same interface.

By default, the `-rtl_comment` argument is set to TRUE, and comments are included in the generated RTL, as shown in the following example:

```
comp2 I(
//IF1
.P1(p1), //I:32 L1
.P2(p2), //I:32 L2
//end of interface IF1
//IF2
.P3(p3), //I:32 L3
.P4(p4) //I:32 L4
//end of interface IF2
);
```

As shown in the previous example, GenSys generators dump inline comments in the instance dump of RTL. Each terminal in the port map has comments describing the direction, width, and description of the master port. Also, the terminal of interface ports are grouped together and preceded by the interface name as comment.

If set to FALSE, comments are not included in the generated RTL, as shown in the following example:

```
comp2 I(
.P1(p1),
.P2(p2),
.P3(p3),
.P4(p4)
);
```

`-defval_rtl_dump` <TRUE | FALSE>

(Optional) Specifies the behaviour of default value in RTL.

By default, the `-defval_rtl_dump` argument is set to *FALSE* and behaviour of default value in RTL is same as explicit tieoff connection.

When the *TRUE* option is set for the `-defval_rtl_dump` argument, the default value is treated differently than explicit tieoff connection.

Consider the example given below:

```
new component comp
```

```

add_port -name P1 -direction IN -lsb 0 -msb 2 -default_value 3'b101
save comp

new design top
add_instance -name I1 -master comp
save top

```

If the `-defval_rtl_dump` argument is set to `TRUE`, the RTL created for this TCL file will be as given below:

```

module top ();
    wire [2:0] GENESIS_DEFAULT_TIEOFF_101;
    comp I1 (
        .P1(GENESIS_DEFAULT_TIEOFF_101) //I:3
    );
    assign GENESIS_DEFAULT_TIEOFF_101 = {3'h5};
endmodule

```

However, if the `-defval_rtl_dump` argument is set to `FALSE`, the RTL created for this TCL file will be as given below:

```

module top ();

    comp I1 (
        .P1({3'h5}) //I:3
    );
endmodule

-vectorize_net <TRUE | FALSE>

```

(Optional) Specifies whether the port should be vectorized in RTL.

By default, this argument is set to `TRUE` and the port with MSB:LSB as (n:n) is vectorized and is dumped as [n:n] according to MRS. Refer to [Example 1](#).

However, if this argument is set to `FALSE`, the port is dumped as a scalar. Therefore, if you specify a port with MSB:LSB as (n:n), GenSys dumps it as a scalar. Note that n is a whole number.

```
-print_conn_comment <TRUE | FALSE>
```

(Optional) Specifies if the description of connections should be dumped to VHDL. By default, this option is set to `TRUE`. You can set it to `FALSE` to stop dumping of comments in VHDL.

```
-delta_delay <TRUE | FALSE>
```

(Optional) Specifies if the delay information of connections should be dumped to VHDL or Verilog RTL dump. By default, this option is set to `TRUE`. You can set it to `FALSE` to stop dumping of delay information. For more details, refer to [Example 2](#).

`-header_template <file-name>`

(Optional) Specifies the template file that is run before generating every RTL file. The output generated on running the template is used as header for the current RTL file.

The template file, which is written by the user, can make use of name of the active RTL file and the active language to create the header. The name of the active RTL file and the active language are returned by the `getActiveFile` and `getActiveLang` Perl APIs, respectively.

A sample template file is given below:

```
-----
--H
[%-SET file = root.active("activefile")-%]
[%-SET lang = root.active("activelang")-%]

--H Filename : [%file%]
--H Language : [%lang%]
-----
```

`-component_decl <TRUE | FALSE>`

(Optional) Specifies if the component declaration is to be dumped in the VHDL netlist generation. By default, this argument is set to TRUE and component declaration is dumped.

If this argument is set to FALSE, VHDL netlist will not have any component declaration in the architecture of the parent module.

`-uniquify_defval_rtl_dump <TRUE | FALSE>`

(Optional) Behaviour of this argument is similar to the `-defval_rtl_dump` argument, but has one difference. When the `-defval_rtl_dump` argument is set to TRUE, a unique signal name is generated for each default value. In this case, for two or more pins with same default value, same wire is created to connect them.

However, with `-uniquify_defval_rtl_dump`, a unique signal will always be generated for each instance pin even if two or more pins of same or different instances have the same default value. In this case, for two or more pins with same default value, a new wire will be generated for each pin. For more details, refer to [Example 3](#).

`-optimized_nets <TRUE | FALSE>`

(Optional) Specifies whether the wires generated in RTL should be optimized. By default, this argument is set to TRUE and the wires generated in RTL are optimized. For more details, refer to [Example 4](#).

`-wrapper_creation <TRUE | FALSE>`

(Optional) Switches off wrapper module creation. By default, dumping of wrapper modules is allowed during RTL generation. You can set this option to FALSE to stop wrapper creation during RTL generation.

`-uniquify_tie_rtl_dump <TRUE | FALSE>`

(Optional) Uniquifies all the tieoff signals during RTL dump. By default, this argument is set to FALSE and GenSys assigns constant values to the tieoff signals. Set this parameter to TRUE, if you want to uniquify the tieoff signals along with the default tieoff. For more details, refer to [Example 6](#).

`-dump_timescale <TRUE | FALSE>`

(Optional) Dumps the Verilog timescale directive to every Verilog RTL file which is required by the simulators. By default, the value of this argument is set to FALSE, and the timescale directive is not being dumped.

`-default_tieoff_prefix <default-prefix>`

(Optional) Specifies the default tieoff prefix. By default, the value of this option is set to GENESIS_DEFAULT_TIEOFF.

`-forced_tieoff_prefix <forced-prefix>`

(Optional) Specifies the forced tieoff prefix. By default, the value of this option is set to GENESIS_FORCED_TIEOFF.

`-temp_tieoff_prefix <temp-prefix>`

(Optional) Specifies the temporary tieoff prefix. By default, the value of this option is set to GENESIS_TEMP_TIEOFF.

`-useSupplyName <TRUE | FALSE>`

(Optional) If this argument is set to TRUE, the generated Verilog RTL will contain VSS/VCC in place of 1'b0/1'b1 and the generated VHDL RTL will contain VSS/VCC in place of '0'/'1'.

By default, the value of this argument is set to FALSE.

`-rtl_supply0_name <supply0-name>`

(Optional) Specifies the supply0 name, which will be dumped in place of VSS.

`-rtl_supply1_name <supply1-name>`

(Optional) Specifies the supply1 name, which will be dumped in place of VCC.

However, please note the following points:

- The `-rtl_supply0_name` and `-rtl_supply1_name` arguments will not have any effect if the value of the `-useSupplyName` argument is set to `FALSE`.
- If the supply names specified in the `-rtl_supply0_name` and `-rtl_supply1_name` arguments conflict with an existing net in the design, GenSys renames such supply names and flags a warning.
- If you specify same names in the `-rtl_supply0_name` and `-rtl_supply1_name` arguments, GenSys flags an Error message.

`-black_box_dump <TRUE | FALSE>`

(Optional) Specifies whether empty architecture should be dumped for leaf level components in the design.

If the value of this argument is set to `TRUE`, empty architecture is dumped for leaf level components in the design. When you generate Verilog/VHDL after specifying the value of this argument to `TRUE`, the following information would be missing from these generated files:

- Intermediate nets
- Supply names
- Assignment statements

Note:

The VHDL/Verilog dumping of designs will not be effected by using this argument.

By default, the value of this argument is set to `FALSE`.

Refer to [Example 7](#).

`-preserve_lsb_msb <TRUE | FALSE >`

(Optional) Specifies if wire/signal dumped in RTL should have same indexes as of its driver port/terminal. By default, the wire is offset to 0 even if driver port/terminal has offset LSB.

Consider that `I1.P1` \hat{O} `I2.P2` where `I1.P1` is driver with indexes `[10:3]`. If this option is set to `FALSE`(default), wire dumped in RTL will be `"wire [7:0] I1_P1"` while when set to `TRUE`, wire definition will be `"wire [10:3] I1_P1"`.

By default, if some bits of wire are not read/written, it is further optimized in the RTL dump. When this option is set to `TRUE`, wires are not optimized.

Connected wires are optimized to the LSB/MSB of the master port as per the following:

- Optimized LSB of the wire is set to the LSB of the first connected slice of the master port.

- Optimized MSB of the wire is set to the MSB of the last connected slice of the master port.

By default, the value of this argument is set to FALSE.

Refer to [Example 8](#).

Note:

Connections having delta delay are not affected by this switch.

```
-allow_out_port_read <TRUE | FALSE >
```

(Optional) Allows output ports of a design to be read, if this argument is set to TRUE. The output ports being read can act as driver for connections with other ports.

Setting this argument to TRUE is useful in cases in which the usage of Verilog language permits the user to use the output ports as driver for connections.

By default, this argument is set to FALSE.

```
-archName <architecture-name-format>
```

(Optional) Specifies the format of architecture names to be dumped in VHDL during VHDL generation.

For example, consider the following command:

```
set_rtl_option -archName "\$name_structure"
```

The above command would dump architecture name as *<entity-name>_structure*. In this case, \$name is considered as *<entity-name>*.

If you want to dump the architecture name as *my_<entity-name>_structure*, use the following command:

```
set_rtl_option -archName "my\_\$name_structure"
```

```
-dumpVHDLSingleFile <TRUE | FALSE>
```

(Optional) Dumps both VHDL entity and architecture for a module in a single file if this argument is set to TRUE. In this case, the name of the file is *<entity-name>.vhd*.

By default this argument is set to FALSE, which means that VHDL entity and architecture of a module are dumped in separate files. The entity is dumped in *<entity-name>_e.vhd* file and architecture is dumped in *<entity-name>_rtl.vhd* file.

For example, consider an entity *des* that has to be dumped in VHDL. Now, if you set the *-dumpVHDLSingleFile* argument to TRUE, GenSys dumps both entity and architecture in a single file, *des.vhd*. However, if you set this argument to FALSE, GenSys dumps the entity in *des_e.vhd* and architecture in *des_rtl.vhd*.


```
-vhdl_port_assign <TRUE | FALSE >
```

(Optional) Specifies whether the assignment from I/O path should be dumped in the generated VHDL netlist.

By default, this argument to FALSE so that GenSys does not dump the assignment from I/O path in the generated VHDL netlist.

Set this argument to TRUE to have such assignment dumped in the VHDL netlist.

For details, see [Example 9](#).

```
-config_spec_dump <TRUE | FALSE >
```

(Optional) Dumps configuration specification after each component declaration in the generated VHDL.

Following is an example of the specification dumped in the generated VHDL:

```
for I1,I2 : uart use entity WORK.uart(uart1);
```

```

graph TD
    A["for I1,I2 : uart use entity WORK.uart(uart1);"]
    B["Instances"]
    C["Entity"]
    D["Logical Library"]
    E["Architecture"]
    A --> B
    A --> C
    A --> D
    A --> E

```

The above specification indicates binding of the I1 and I2 instances of the uart entity to the uart1 architecture of the uart entity in the WORK library.

By default, this argument is set to FALSE and the configuration specification is not dumped in the generated VHDL.

For details, see [Example 10](#).

```
-createGatingMux <TRUE | FALSE>
```

(Optional) Creates Gating Mux for default value in output/oen path

By default, this argument is set to be TRUE and GenSys creates Gating Mux for default value in output/oen path.

Note:

This is an IO Tcl option.

```
-create_stubs <TRUE | FALSE>
```

(Optional) Specifies if RTL stub models, that is leaf-level modules, should be generated.

If you set this argument to TRUE, the sources.f file points to the GenSys generated RTL.

If you set this argument to FALSE (default), the dirty design/subsystem in sources.f points to GenSys generated RTL and the component (leaf-level module) points to the file-set path. However, if the component does not have any file-set information, sources.f points to GenSys generated RTL.

For details, see [Example 11 \(RTL Stub-Models\)](#).

```
-preserve_comments <TRUE | FALSE>
```

(Optional) By default, this argument is set to TRUE and GenSys saves pragmas and comments in the generated RTL.

Note:

Comments that are added by default by GenSys are always preserved in the generated RTL irrespective of the value of this argument.

```
-preserve_floating_net_name <TRUE | FALSE>
```

(Optional) By default, this argument is set to TRUE; if the input RTL has any hanging wire that is connected to an instance terminal, GenSys preserves the name of the wire in the generated RTL.

GenSys preserves the hanging wire name only when the instance terminal is completely connected to a hanging wire. GenSys does not preserve partial hanging net names.

For further explanation, refer to [Example 12 - preserve_floating_net_name](#).

```
-export_binary_value <VERILOG | FALSE>
```

(Optional) By default, this argument is set to FALSE.

When this argument is set to VERILOG, all tieoff values, default values, and parameter values, which are in Verilog hex('h'), Verilog octal('o'), Verilog decimal('d'), VHDL hex(16#), VHDL decimal(10#), VHDL octal(8#), or VHDL binary(2#) are converted to an equivalent Verilog binary('b') value on generated verilog RTL.

```
-export_prj_file <TRUE | FALSE>
```

(Optional) By default, this argument is set to FALSE.

Set this argument is set to TRUE so that during RTL generation, GenSys generates a project file (.prj file) along with the sources.f and spyglass.f files.

For information on a project file, refer to *Atrenta Console User Guide* of the SpyGlass help set.

```
-generate_unique_filelists <TRUE | FALSE>
```

(Optional) Specifies if file lists should be generated for the masters of sub-systems in the top-level design while generating RTL.

By default, this argument is set to FALSE and GenSys generates file lists only with respect to the top-level design.

Set this argument to TRUE so that GenSys generates file lists for the masters of all the sub-systems in the top-level design.

See [Example 13](#).

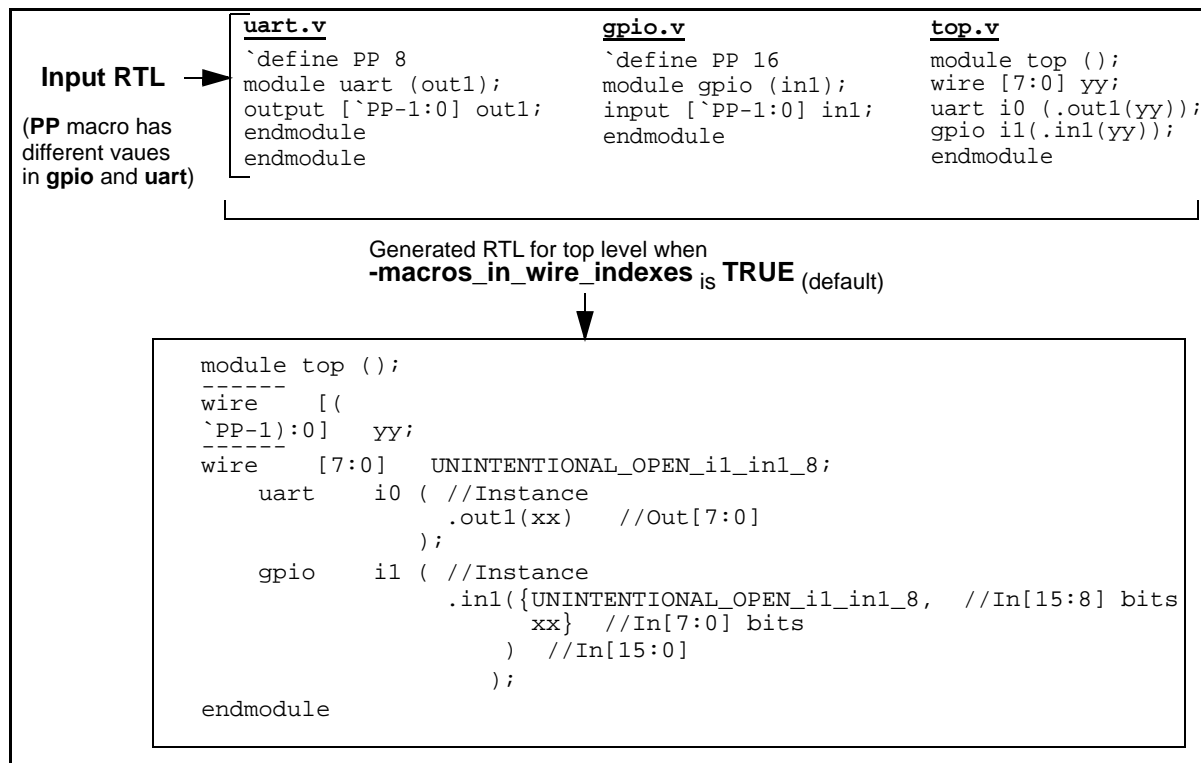
`-macros_in_wire_indexes <TRUE | FALSE>`

(Optional) Set this argument to FALSE to stop dumping parameterized wires in the parent design with ``define` (macros) unless the parent design has the same ``define` (macro) available with the same value as in the driver module.

By default, this argument is set to TRUE.

This argument should be set to FALSE when the input RTL has different values defined for the same macros.

Consider the following example:



In the input RTL of the above example, the PP macro is used in the yy wire (based on the driver side port declaration) assuming macros generally have the same value across the complete RTL. But in this case, this is incorrect because `PP has different values inside uart and gpio.

When this argument is set to FALSE, the generated RTL will have the wire declaration for yy as follows:

```
wire [7:0] yy;
```

This is valid RTL.

Examples

Some of the examples of using the set_rtl_option commands are:

Example 1

This example illustrates the behavior of the vectorize_net argument. Consider the following snippet:

```
new component M1
add_port -name R1 -lsb 0 -msb 0 -direction IN
saveall

new design abc
add_port -name P1 -lsb 0 -msb 0
add_instance -name I1 -master M1
add_connection -instance I1 -pin R1 -port P1
saveall
```

By default, the RTL created for this TCL file will be as given below:

```
module abc (P1);
    inout [0:0] P1;
    M1 I1
    (
        .R1(P1[0]) //I:1
    );
endmodule
```

In the above design, port P1 is treated as a vector. However, if before RTL generation, the value of the -vectorize_net argument is set to FALSE, port P1 will be treated as scalar as given below:

```
module abc (P1);
```

```

inout P1;
M1 I1
(
    .R1(P1) //I:1
);
endmodule

```

Example 2

New connections are added using the following commands:

```

add_connection -instance I1 -pin R1 -port P1 -lsb 8 -msb 10 -deltadelay 2
add_connection -instance I1 -pin R2 -port P1 -lsb 5 -msb 7
add_connection -instance I1 -pin R3 -port P1 -lsb 2 -msb 4 -deltadelay 5

```

For the above given connections, if before RTL generation, the `-delta_delay` option is set to `TRUE`, delay information will be dumped in VHDL RTL as given below:

```

P1_INT <= (P1);
VSS <= '0';
VCC <= '1';
I1 : comp
PORT MAP (
    R1 =>P1_delta, -- I:3
    R2 =>P1_INT(7 downto 5), -- I:3
    R3 =>P1_delta_1 -- I:3
);
P1_delta_1 <= P1_INT(4 downto 2) after 5ns;
P1_delta <= P1_INT(10 downto 8) after 2ns;

```

The delay information will be dumped in Verilog RTL as given below:

```

comp I1 (
    .R1(P1_delta), //I:3
    .R2(P1[7:5]), //I:3
    .R3(P1_delta_1) //I:3
); assign #5 P1_delta_1 = P1[4:2];
assign #2 P1_delta = P1[10:8];

```

Example 3

Unique signal names are generated when `-uniquify_defval_rtl_dump` argument is used for the design given below:

```

set_rtl_option -uniquify_defval_rtl_dump TRUE
new component comp
add_port -name P1 -direction IN -lsb 0 -msb 2 -default_value 3'b101
add_port -name P2 -direction IN -lsb 0 -msb 2 -default_value 3'b101
add_port -name P3 -direction IN -lsb 0 -msb 2 -default_value 3'b111
save comp

```

```

new component compl
add_port -name P1 -direction IN -lsb 0 -msb 2 -default_value 3'b101
save compl

new design top
add_instance -name I1 -master comp
add_instance -name I2 -master compl
run GenerateVerilog

```

In this case, the generated RTL has unique wire names for all pins, as shown below:

```

module top ();
wire    [2:0]    GENESIS_DEFAULT_TIEOFF_I1_P1;
wire    [2:0]    GENESIS_DEFAULT_TIEOFF_I1_P2;
wire    [2:0]    GENESIS_DEFAULT_TIEOFF_I1_P3;
wire    [2:0]    GENESIS_DEFAULT_TIEOFF_I2_P1;

    comp I1 (
        .P1(GENESIS_DEFAULT_TIEOFF_I1_P1),    //I:3
        .P2(GENESIS_DEFAULT_TIEOFF_I1_P2),    //I:3
        .P3(GENESIS_DEFAULT_TIEOFF_I1_P3)      //I:3    );
    compl I2 (
        .P1(GENESIS_DEFAULT_TIEOFF_I2_P1)      //I:3
    );
    assign GENESIS_DEFAULT_TIEOFF_I1_P1 = {3'h5};
    assign GENESIS_DEFAULT_TIEOFF_I1_P2 = {3'h5};
    assign GENESIS_DEFAULT_TIEOFF_I1_P3 = {3'h7};
    assign GENESIS_DEFAULT_TIEOFF_I2_P1 = {3'h5};
endmodule

```

Example 4

Consider the following example in which a connection is created between two component instances:

```

new component compl
add_port -name R1 -direction OUT -msb 5 -lsb 0
save compl

new component comp2
add_port -name R2 -direction IN -msb 5 -lsb 0
save comp2
# design definition and connection
new design TEST_TOP
add_instance -master compl -name inst1
add_instance -master comp2 -name inst2
add_connection -instance inst1 -pin R1 -msb 3 -lsb 2
               -instance inst2 -pin R2 -msb 3 -lsb 2
run GenerateVerilog

```

In the above example, inst1.R1[3:2] is being connected to inst2.R2[3:2]. If the -optimized_nets argument is set to FALSE, the Verilog generated will be as follows:

```
module TEST_TOP ();
wire [5:0] inst1_R1;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_0;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_2;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_0;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_2;
  comp1 inst1 (
    .R1({UNINTENTIONAL_OPEN_inst1_R1_2,
        inst1_R1[3:2],
        UNINTENTIONAL_OPEN_inst1_R1_0})    //O:6
  );
  comp2 inst2 (
    .R2({UNINTENTIONAL_OPEN_inst2_R2_2,
        inst1_R1[3:2],
        UNINTENTIONAL_OPEN_inst2_R2_0})    //I:6
  );
endmodule
```

Here wire inst1_R1 is [5:0] while only two bits of this wire are being used. If the -optimized_nets argument is set to TRUE (default behavior), the Verilog generated will be as follows:

```
module TEST_TOP ();
wire [1:0] inst1_R1;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_0;
wire [1:0] UNINTENTIONAL_OPEN_inst1_R1_2;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_0;
wire [1:0] UNINTENTIONAL_OPEN_inst2_R2_2;
  comp1 inst1 (
    .R1({UNINTENTIONAL_OPEN_inst1_R1_2,
        inst1_R1,
        UNINTENTIONAL_OPEN_inst1_R1_0})    //O:6
  );
  comp2 inst2 (
    .R2({UNINTENTIONAL_OPEN_inst2_R2_2,
        inst1_R1,
        UNINTENTIONAL_OPEN_inst2_R2_0})    //I:6
  );
endmodule
```

Here, the wire inst1_R1 is declared as [1:0] i.e., this wire is optimized.

Example 5

Consider the following example in which a hierarchy is created:

```
new component comp
add_port -name P1 -direction IN -lsb 0 -msb 2
```

```

save comp
new design top
add_port -name R1 -direction IN
add_port -name R2 -direction IN
add_instance -name I1 -master comp
add_connection -instance I1 -pin P1 -lsb 0 -msb 0 -port R1
add_connection -instance I1 -pin P1 -lsb 2 -msb 2 -port R2
add_connection -instance I1 -pin P1 -lsb 1 -msb 1 -tieoff 1
create_partition -name P1
set_partition -instance I1 -hierarchy H1.H2

```

By default, top level RTL created for this design will be as given below:

```

module top (R1,                R2);
input R1;
input R2;
wire UNINTENTIONAL_OPEN_U0_H1_H1_I1_P1_in_1;
    H1 U0_H1 (
        .H1_I1_P1_in(
{R2, UNINTENTIONAL_OPEN_U0_H1_H1_I1_P1_in_1, R1}
)    //I:3
    );endmodule

```

In the RTL given above, the H1_I1_P1_in port has one bit hole.

If the -holes_in_rtl_hierarchy option is set to FALSE, the generated RTL will be as given below:

```

module top (R1, R2);
input R1;
input R2;
    H1 U0_H1 (
        .H1_I1_P1_in({R2, R1})    //I:2
    );
endmodule

```

In this case, the H1_I1_P1_in port is 2 bit wide suppressing the hole.

Example 6

Consider the following example that shows how the tieoff signals are uniquified:

```

new component comp
add_port -name P1 -lsb 0 -msb 1 -direction IN
add_port -name P2 -direction IN
save comp
new design des
add_instance -name I1 -master comp
add_connection -instance I1 -pin P1 -tieoff 0x01
add_connection -instance I1 -pin P2 -tieoff 0x1
run GenerateVerilog

```


By default, the top level RTL generated for design, des, will be as given below:

```
module des ();
  comp    I1 (
    .P1({2'h1}), //I:2
    .P2(1'b1)    //I:1
  );
endmodule
```

When the -uniquify_tie_rtl dump switch is set to TRUE, the top level RTL will be as given below:

```
module des ();
wire [1:0] GENESIS_DEFAULT_TIEOFF_I1_P1;
wire GENESIS_DEFAULT_TIEOFF_I1_P2;
comp I1 (.P1(GENESIS_DEFAULT_TIEOFF_I1_P1), //I:2
.P2(GENESIS_DEFAULT_TIEOFF_I1_P2) //I:1
);assign GENESIS_DEFAULT_TIEOFF_I1_P2 = 1'b1;
assign GENESIS_DEFAULT_TIEOFF_I1_P1 = {2'h1};
endmodule
```

Example 7

Consider the following VHDL produced when the -black_box_dump argument is set to FALSE:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE WORK.ALL;
ARCHITECTURE compl of compl IS
SIGNAL VSS      : STD_LOGIC;
SIGNAL VCC      : STD_LOGIC;
SIGNAL p6_INT   : STD_LOGIC;
SIGNAL p7_INT   : STD_LOGIC;
begin
  p6      <= (p6_INT);
  p7_INT  <= (p7);
  VSS     <= '0';
  VCC     <= '1';
  p1      <= "000";
  p2      <= "000";
  p3      <= "000";
  p4      <= "000";
end compl;
```

However, if the -black_box_dump argument is set to TRUE, following VHDL is produced:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
```

```
USE WORK.ALL;
ARCHITECTURE compl of compl IS
begin
end compl;
```

Consider the following Verilog produced when the `-black_box_dump` argument is set to **FALSE**:

```
module compl (p1,
              p2,
              p3,
              p4,
              p6,
              p7);
  supply0      VSS;
  supply1      VCC;
  inout  [2:0] p1;
  inout  [2:0] p2;
  inout  [2:0] p3;
  inout  [2:0] p4;
  output          p6;
  input           p7;
  assign p1 = {3'h0};
  assign p2 = {3'h0};
  assign p3 = {3'h0};
  assign p4 = {3'h0};
endmodule
```

However, if the `-black_box_dump` argument is set to **TRUE**, following Verilog is produced:

```
module compl (p1,          p2,
              p3,
              p4,
              p6,
              p7);
  inout  [2:0] p1;
  inout  [2:0] p2;
  inout  [2:0] p3;
  inout  [2:0] p4;
  output          p6;
  input           p7;
endmodule
```

Example 8

Consider the following command:

```
set_rtl_option -preserve_lsb_msb TRUE
new component comp
add_port -name pc1 -lsb 2 -msb 9 -direction IN
add_port -name pc2 -lsb 4 -msb 15 -direction OUT
save
```

```

new design des
add_instance -name c1 -master comp
add_instance -name c2 -master comp
add_port -name pd1 -lsb 1 -msb 10 -direction IN
add_port -name pd2 -lsb 2 -msb 15 -direction OUT
add_connection -instance c1 -pin pc1 -lsb 5 -msb 7
               -instance c2 -pin pc2 -lsb 9 -msb 11
add_connection -instance c1 -pin pc2 -lsb 6 -msb 7
               -instance c2 -pin pc1 -lsb 4 -msb 5
add_connection -instance c1 -pin pc1 -lsb 3 -msb 4
               -port pd1 -lsb 5 -msb 6
add_connection -instance c2 -pin pc2 -lsb 6 -msb 7
               -port pd2 -lsb 3 -msb 4
save

```

The Generated RTL Verilog for the above example is as follows:

```

module des (pd1,
            pd2);
input  [10:1]  pd1;
output [15:2]  pd2;
wire   [7:6]   c1_pc2;
wire   [2:2]   UNINTENTIONAL_OPEN_c1_pc1_0;
wire   [9:8]   UNINTENTIONAL_OPEN_c1_pc1_1; wire   [5:4]
UNINTENTIONAL_OPEN_c1_pc2_0;
wire   [15:8]  UNINTENTIONAL_OPEN_c1_pc2_1;
wire   [3:2]   UNINTENTIONAL_OPEN_c2_pc1_0; wire   [9:6]
UNINTENTIONAL_OPEN_c2_pc1_1;
wire   [5:4]   UNINTENTIONAL_OPEN_c2_pc2_0; wire   [8:8]
UNINTENTIONAL_OPEN_c2_pc2_1;
wire   [15:12] UNINTENTIONAL_OPEN_c2_pc2_2;
  comp c1 (
    .pc1({UNINTENTIONAL_OPEN_c1_pc1_1[9:8],
          c2_pc2[11:9],
          pd1[6:5],
          UNINTENTIONAL_OPEN_c1_pc1_0[2:2]}), //I:8
    .pc2({UNINTENTIONAL_OPEN_c1_pc2_1[15:8],
          c1_pc2[7:6],
          UNINTENTIONAL_OPEN_c1_pc2_0[5:4]}), //O:12
  );
  comp c2 (
    .pc1({UNINTENTIONAL_OPEN_c2_pc1_1[9:6],
          c1_pc2[7:6],
          UNINTENTIONAL_OPEN_c2_pc1_0[3:2]}), //I:8
    .pc2({UNINTENTIONAL_OPEN_c2_pc2_2[15:12],
          c2_pc2[11:9],
          UNINTENTIONAL_OPEN_c2_pc2_1[8:8],
          pd2[4:3],
          UNINTENTIONAL_OPEN_c2_pc2_0[5:4]}), //O:12
  );
endmodule

```

The Generated RTL VHDL for the above example is as follows:

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

```

```

USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE WORK.ALL;
ARCHITECTURE des of des IS
--Component Declarations...
COMPONENT comp
  PORT (
    pc1      : IN      STD_LOGIC_VECTOR (9 downto 2);
    pc2      : OUT      STD_LOGIC_VECTOR (15 downto 4)
  );END COMPONENT;
SIGNAL c2_pc2 : STD_LOGIC_VECTOR (11 DOWNT0 9);
SIGNAL c1_pc2 : STD_LOGIC_VECTOR (7 DOWNT0 6);
SIGNAL UNINTENTIONAL_OPEN_c1_pc1_0 : STD_LOGIC_VECTOR
(2 DOWNT0 2);SIGNAL UNINTENTIONAL_OPEN_c1_pc1_1 : STD_LOGIC_VECTOR
(9 DOWNT0 8);SIGNAL UNINTENTIONAL_OPEN_c1_pc2_0 : STD_LOGIC_VECTOR(5
DOWNT0 4);SIGNAL UNINTENTIONAL_OPEN_c1_pc2_1 : STD_LOGIC_VECTOR
(15 DOWNT0 8);SIGNAL UNINTENTIONAL_OPEN_c2_pc1_0 : STD_LOGIC_VECTOR
(3 DOWNT0 2);SIGNAL UNINTENTIONAL_OPEN_c2_pc1_1 : STD_LOGIC_VECTOR
(9 DOWNT0 6);SIGNAL UNINTENTIONAL_OPEN_c2_pc2_0 : STD_LOGIC_VECTOR
(5 DOWNT0 4);SIGNAL UNINTENTIONAL_OPEN_c2_pc2_1 : STD_LOGIC_VECTOR
(8 DOWNT0 8);SIGNAL UNINTENTIONAL_OPEN_c2_pc2_2 : STD_LOGIC_VECTOR
(15 DOWNT0 12);SIGNAL pd1_INT : STD_LOGIC_VECTOR (10 DOWNT0 1);
SIGNAL pd2_INT : STD_LOGIC_VECTOR (15 DOWNT0 2);
begin  pd1_INT  <= (pd1);
      pd2  <= (pd2_INT);
      c1 : comp
      PORT MAP (
        pc1(9 downto 8  =>UNINTENTIONAL_OPEN_c1_pc1_1(9
downto 8),
        pc1(7 downto 5)  =>c2_pc2(11 downto 9),
        pc1(4 downto 3)  =>pd1_INT(6 downto 5),
        pc1(2)  =>UNINTENTIONAL_OPEN_c1_pc1_0(2), -- I:8
        pc2(15 downto 8)  =>UNINTENTIONAL_OPEN_c1_pc2_1(15
downto 8),  pc2(7 downto 6)  =>c1_pc2(7 downto 6),
        pc2(5 downto 4)  =>UNINTENTIONAL_OPEN_c1_pc2_0(5
downto 4) -- O:12
      );  c2 : comp
      PORT MAP (
        pc1(9 downto 6)  =>UNINTENTIONAL_OPEN_c2_pc1_1(9
downto 6),
        pc1(5 downto 4)  =>c1_pc2(7 downto 6),
        pc1(3 downto 2)  =>UNINTENTIONAL_OPEN_c2_pc1_0(3
downto 2), -- I:8
        pc2(15 downto 12)  =>UNINTENTIONAL_OPEN_c2_pc2_2(15
downto 12),
        pc2(11 downto 9)  =>c2_pc2(11 downto 9),
        pc2(8)  =>UNINTENTIONAL_OPEN_c2_pc2_1(8),
        pc2(7 downto 6)  =>pd2_INT(4 downto 3),
        pc2(5 downto 4)  =>UNINTENTIONAL_OPEN_c2_pc2_0(5
downto 4) -- O:12
      );
end des;

```

Example 9

Consider that you execute the following Tcl file:

```
set_rtl_option -vhdl_port_assign FALSE
new component comp
add_port -name P1 -direction IN
new design des
add_instance -name I1 -master comp
add_port -name A1 -direction IN
add_connection -instance I1 -pin P1 -port A1
run GenerateVHDL
```

In the above case, the `-vhdl_port_assign` argument of the `set_rtl_option` Tcl command is set to `FALSE`. Therefore, GenSys dumps the following architecture of the design `des`:

```
COMPONENT comp
  PORT (
    P1      : IN      STD_LOGIC
  );
END COMPONENT;
begin
  I1 : comp
  PORT MAP (
    P1      =>A1 -- I:1
  );
end des;
```

If the `-vhdl_port_assign` argument is set to `TRUE`, GenSys dumps the following architecture of the design `des`:

```
COMPONENT comp
  PORT (      P1      : IN      STD_LOGIC
  );
END COMPONENT;
SIGNAL A1_INT      : STD_LOGIC;
begin  A1_INT      <= (A1);
  I1 : comp
  PORT MAP (  P1      =>A1_INT -- I:1
  );
end des;
```

The difference in both the above architectures is highlighted in blue.

Example 10

Consider an example of the following VHDL generated when the `-config_spec_dump` argument of the `set_rtl_option` is set to `FALSE`:

```
LIBRARY IEEE;
```

```

USE IEEE.std_logic_1164.ALL;
.....
USE WORK.ALL;
ARCHITECTURE top of top IS
--Component Declarations...
COMPONENT uart
GENERIC(
    P1      : INTEGER := 1
);
PORT (
    .....
);
END COMPONENT;
COMPONENT uart_2_0
GENERIC(
    P1      : INTEGER := 2
); END COMPONENT;

```

If you set the `-config_spec_dump` argument of the `set_rtl_option` is set to `TRUE`, the following VHDL is generated (Specifications dumped are highlighted in blue):

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
.....
USE WORK.ALL;
ARCHITECTURE top of top IS
--Component Declarations...
COMPONENT uart
GENERIC(
    P1      : INTEGER := 1
);
PORT (
    .....
);
END COMPONENT;
for I1,I2 : uart use entity WORK.uart(uart);
COMPONENT uart_2_0
GENERIC(
    P1      : INTEGER := 2
);
END COMPONENT;
for I3,I4 : uart_2_0 use entity WORK.uart_2_0(uart_2_0);

```

Example 11 (RTL Stub-Models)

- Example 1

Consider the current working directory as `/stub/case1/`.

Now consider that you specify the following command:

```
set_rtl_option -create_stubs FALSE
load_rtl comp.v des.v
run generateVerilog
```

Where des is a design and comp is instantiated in des.

In this case, the generated sources.f points to:

- /stub/case1/Verilog/des.v (GenSys generated RTL path)
- /stub/case1/comp.v (File-set path)

- **Example 2**

Consider the current working directory as /stub/case1/.

Now consider that you specify the following command:

```
set_rtl_option -create_stubs TRUE
load_rtl comp.v des.v
run GenerateVerilog
```

Where des is a design and comp is instantiated in des.

In this case, the generated sources.f points to:

- /stub/case1/Verilog/des.v (GenSys generated RTL path)
- /stub/case1/Verilog/comp.v (GenSys generated RTL path)

- **Example 3**

Consider the current working directory as /stub/case1/.

Now consider that you specify the following command:

```
new component comp
add_port -name p1
new design d
add_instance -name inst -master comp
add_port -name p2
add_connection -port p2 -instance inst -pin p1
run GenerateVerilog
```

In this case, the generated sources.f points to:

- /stub/case1/Verilog/des.v (GenSys generated RTL path)
- /stub/case1/Verilog/comp.v (GenSys generated RTL path as comp does not have any file-set information).

Example 12 - preserve_floating_net_name

In the following snippet, preserves x, y, z, and w net names by default, instead of generating empty brackets.

```
module top(P1);
input P1;
wire [2:0] x;
wire y;
wire z;
wire [2:0] w;
wire [1:0] v;
leaf I1(x,y,z,w,{v,P1});
endmodule
```

Example 13

Consider that you run the following Tcl commands:

```
new component comp
save
new design sub0
save
new design sub1a
add_instance -master sub0 -name sub0I
save
new design sub1b
save
new design top
add_instance -master sub1a -name sub1aI
add_instance -master sub1b -name sub1bI
add_instance -master comp -name compI
save
```

In this case, if you set the -generate_unique_filelists argument to TRUE, GenSys generates the following:

- sub1a.f
This file contains the file list of masters instantiated inside sub1a. Note that if there were options, such as incdir and sglb required for standalone compilation of sub1a, they are also included in this file.
- Same information as above in sub1b.f
- sources.f and spyglass.f files for the top-level design

However, sub0.f is not generated in this case as sub0 is not instantiated in the top-level design.

set_rtlimport_option

Specifies various options during RTL import.

Usage

```
set_rtlimport_option
  -DEBUG <0 | 1 | 2 | 3 | 4 | 5 | 6>
  [ -load_from_library_browser <TRUE | FALSE> ]
  [ -default_vendor <vendor-name> ]
  [ -default_version <version> ]
  [ -default_library <library-name> ]
  [ -merge_glue <TRUE | FALSE> ]
  [ -preserve_comments <TRUE | FALSE> ]
  [ -preserve_pragmas <TRUE | FALSE> ]
  [ -stop_on_complete_glue_modules <TRUE | FALSE> ]
  [ -stop_on_partial_glue_modules <TRUE | FALSE> ]
  [ -preserve_configurations <TRUE | FALSE> ]
  [ -preserve_if_generate <TRUE | FALSE> ]
  [ -module_configurations_only <TRUE | FALSE> ]
  [ -preserve_file_link_names <TRUE | FALSE> ]
  [ -eval_param_in_sliced_connections <TRUE | FALSE> ]
  [ -set_default_value <TRUE | FALSE> ]
```

Description

The set_rtlimport_option command specifies various options during RTL import.

Arguments

The set_rtlimport_option command has the following arguments:

DEBUG <0 | 1 | 2 | 3 | 4 | 5 | 6>

Specifies the information to be dumped in RTL import log.

DEBUG can take any of the following values:

- 0: Disables generation of rtlimport.log file. This is the default value.

- 1: Specifies that the time and memory stamp of the following events should be dumped in the `rtlimport.log` file.

Import of design	Import of component
Population of instances	Import of ports
Import of parameters	Interfaces inference
Population of connectivity details	Inference of interface connections
Vectorization of scalar connections	Inference of exported interfaces

Along with these details, information about components that are being treated as black boxes is also logged.

Note:

The time and memory stamp is taken when the above mentioned events start taking place.

- 2: Specifies that along with the time and memory stamp of the events listed above, the name of various imported ports and parameters are also logged in the log file.
- 3: This value is reserved for future RTL import related debug messages.
- 4, 5, 6: These values are used for debug messages related to the preserve configuration flow. When any of these values are supplied then a separate debug file named *debug_macros.log* is created in the *log* directory where the all preserve configuration related debug messages are added.

Note:

The log directory set using the [set_log_dir](#) Tcl command.

```
-load_from_library_browser <TRUE | FALSE>
```

(Optional) Specifies whether an RTL module should be imported if it is already present in the Library Browser.

By default, the value of this argument is set to `FALSE`, which means that you should explicitly import RTL modules that are already present in the Library Browser. This indicates that during RTL import, the RTL view is imported even if it is present in the library browser.

When you set this argument to `TRUE`, GenSys automatically picks the RTL modules from the Library Browser and therefore, they need not be imported explicitly. GenSys picks the component/design from library browser rather than importing the RTL view. This argument

is made TRUE when in SoC some IPs are already available in MRS/IPXACT view and part of library browser.

`-default_vendor <vendor-name>`

(Optional) Specifies a default vendor name for design-units being imported.

RTL modules don't have any vendor name associated with it. With this option, vendor name will be added to the component/design imported from RTL.

RTL modules don't have any vendor name associated with it. With this option, vendor name will be added to the component/design imported from RTL.

`-default_version <version>`

(Optional) Specifies a default version number for design-units being imported.

`-default_library <library-name>`

Specifies a default library name for design-units being imported.

`-merge_glue <TRUE | FALSE>`

(Optional) Combines all glue logic of a design into a single module so that only one instance of glue is visible in the schematic.

Set this argument to FALSE to:

- Generate separate glue instances for each process/always block
In this case, multiple glue instances appear in the schematic corresponding to different process/always blocks.
- A glue instance for all complex assign statements
Anything that is not a simple buffer assignment comes as part of a complex assign statement.

By default, this argument is set to TRUE.

The `-merge_glue FALSE` argument has been added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

`-preserve_comments <TRUE | FALSE>`

(Optional) By default, this argument is set to TRUE and GenSys imports pragmas and comments from RTL.

These pragmas and comments are retained during various assembly flows and passed on to the generated RTL.

```
-preserve_pragmas <TRUE | FALSE>
```

(Optional) If this argument is set to TRUE, GenSys imports all the code between the pragmas from RTL.

By default, this argument is set to FALSE, and GenSys ignores all the code between the pragmas.

```
-stop_on_complete_glue_modules <TRUE | FALSE>
```

(Optional) If this argument is set to TRUE, all the modules containing only glue logic are considered as stop modules.

By default this argument is FALSE.

If you set the `-stop_on_partial_glue_modules` argument to TRUE, the `-stop_on_complete_glue_modules` argument gets automatically set to TRUE. Therefore, if you want to black box all the modules having glue logic, set the `-stop_on_complete_glue_modules` argument to TRUE.

```
-stop_on_partial_glue_modules <TRUE | FALSE>
```

(Optional) If this argument is set to TRUE, all the modules containing atleast one glue logic are considered as stop modules.

By default this argument is FALSE.

If you set the `-stop_on_partial_glue_modules` argument to TRUE, the `-stop_on_complete_glue_modules` argument gets automatically set to TRUE. Therefore, if you want to black box all the modules having glue logic, set the `-stop_on_complete_glue_modules` argument to TRUE.

```
-preserve_configurations <TRUE | FALSE>
```

(Optional) If this argument is set to TRUE, GenSys considers the Verilog pre processor directives, such as ``ifdef`, ``ifndef`, ``else`, ``elsif`, and ``endif`.

During RTL import, GenSys imports the objects under these directives and preserves the macro combinations applied on these objects, i.e. instances, ports, and connections.

During Verilog RTL generation, all such objects are placed under the same macro combination in which they were present in the original Verilog RTL.

When you set this argument to TRUE and load RTL in GUI with the `-ui` option, then through the GUI widget, you can selectively enable/disable the global macros that impact the configurability values of the objects defined in the RTL being loaded.

For details, refer to the *Handling Verilog Pre Processor Directives* topic of *GenSys User Guide*.

```
-preserve_if_generate <TRUE | FALSE>
```

(Optional) Enables memory swapping in the generate blocks.

By default, this argument is set to FALSE, and memory swapping in the generate blocks is not allowed.

For details on the memory swapping feature, refer to the *Swapping Memories* topic of *GenSys User Guide*.

The -preserve_if_generate TRUE argument has been added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

```
-module_configurations_only <TRUE | FALSE>
```

(Optional) By default, this argument is set to FALSE.

If an RTL has macro combinations, that is ifdef present, all the RTL views are imported and a merged view is created to capture all the objects and connectivity. The global macro combinations include all the ifdef present inside modules and in the global scope, that is outside module definitions.

When you set this argument to TRUE, global macro combinations are not considered for merging. Only those macro combinations are imported/merged that are available inside the modules imported in GenSys.

This option is applicable only when the -preserve_configurations argument is set to TRUE.

```
-preserve_file_link_names <TRUE | FALSE>
```

(Optional) Preserves soft links of file names when this argument is set to TRUE.

By default, this argument is set to FALSE and GenSys resolves file names to base file names.

Example:

Consider *mytop.v* as the soft link present in the current working directory */aaa/bbb/cwd/*:

```
mytop.v => ../RTL/top.v
```

Now consider that you run the following Tcl commands to load RTL and generate Verilog:

```
set_rtlimport_option -preserve_file_link_names TRUE
load_rtl mytop.v
run GenerateVerilog
```

After specifying the above commands, the generated *sources.f* will have the following entry:

```
/aaa/bbb/cwd/mytop.v (This is the soft link)
```

If you set the -preserve_file_link_names argument of the set_rtlimport_option Tcl command to FALSE (default behavior), the generated *sources.f* will have the following entry:

```
/aaa/bbb/RTL/top.v (This is the base file name)
```

```
-eval_param_in_sliced_connections <TRUE | FALSE>
```

(Optional) By default, if a wire/port in the source RTL is partially connected in a port connection of an instance:

- GenSys creates the additional wire `rtlc_N*` with the same width of the port connection and assigns that wire to the port connection.
- GenSys creates an additional *assign* statement between `rtl_N*`.

The above- Gensys behavior is to preserve parameterization of port connections in the generated RTL.

If you set this argument to TRUE, GenSys does not create additional `rtlc_N*` and *assign* statements. Instead, GenSys evaluates the MSB/LSB expressions of port connections in the generated RTL.

The advantage of setting this argument to TRUE is that additional `rtlc_N*` wires and *assign* statements are not created. However, the disadvantage is that the parametrization of port connections is not preserved.

Example

Consider the following source RTL:

```
module top(in);
  parameter MSB=1;
  input [2:0] in;
  mid inst(.in(in[MSB:0]));
endmodule
```

Based on the value specified to the `-eval_param_in_sliced_connections` argument, different RTL is generated, as shown below:

```
-eval_param_in_sliced_connections  
set to FALSE
```

Generated RTL

```
module top(in);
  parameter MSB=1;
  input [2:0] in;
  wire [MSB:0] rtlc_N0;
  mid inst(.in(rtlc_N0));
  assign rtlc_N0 = in[MSB:0];
endmodule
```

```
-eval_param_in_sliced_connections  
set to TRUE
```

Generated RTL

```
module top(in);
  parameter MSB=1;
  input [2:0] in;
  mid inst(.in(in[1:0]));
endmodule
```

`-set_default_value <TRUE | FALSE>`

(Optional) Specifies whether to store tie-off assignments on output ports as default value while loading RTL as a component (IP-packaging flow for RTL assembly). This option is TRUE by default.

Use this option with the IP-packaging flow, i.e. the `load_rtl` command with the `-component` option.

During IP-packaging flow, any tie-off value assigned to the output port of a component is packaged as default value of the port. When this option is FALSE, GenSys does not set the tie-off value as the default value.

Example

Consider the following source RTL:

```
module sample (input a, output ya);
  assign ya = 1'b0;
endmodule
```

Table 2-2

IPXACT (for port ya) with `-set_default_value` set to TRUE

```
<spirit:port>
  <spirit:name>ya</spirit:name>
  <spirit:wire>
    <spirit:direction>out</spirit:direction>
    <spirit:driver>
      <spirit:defaultValue>0</spirit:defaultValue>
    </spirit:driver>
  </spirit:wire>
</spirit:port>
```

IPXACT (for port ya) with `-set_default_value` set to FALSE

```
<spirit:port>
  <spirit:name>ya</spirit:name>
  <spirit:wire>
    <spirit:direction>out</spirit:direction>
  </spirit:wire>
</spirit:port>
```

set_save_template

Sets a directory structure and file name formats

Usage

```
set_save_template [ mrs | ipxact | tcl ] <template-string>
```

Description

The `set_save_template` command specifies default settings that are used while saving objects of different types, such as MRS, IP-XACT, or Tcl. This command does not specify the saving format, rather it dictates how a specific format is saved.

You can provide multiple specifications of this Tcl command to save different types of objects in different formats and in different directories. For details, see [Example 1](#).

Note:

Subsequent `set_save_template` commands override any previous settings for the same format.

Arguments

The `set_save_template` command has the following arguments:

`<template-string>`

(Optional) Specifies default settings.

You can specify the default settings in the `<template-string>` argument of this Tcl command. Following is the format of specifying this argument:

```
"[<relative-dir>/][%vendor<separator>][%library<separator>]  
[%name<separator>][%version<separator>]%obj.<ext>"
```

Details of various arguments of the above format is as follows:

- `<relative-dir>`,
`[%vendor<separator>],[%library<separator>],[%name<separator>],[%version<separator>]` (Optional): Specifies a relative path (not an absolute path) where the object should be saved. If the specified path does not exist, save operation fails and GenSys reports an error.
- `<separator>`: Specifies the separator as a period (.) or a forward-slash (/).
- `obj.<ext>` (Mandatory): Specifies an extension of the object being saved. The extension can only be xml, mrs, or tcl.

If you do not use this command, all files are saved in the db directory.

Examples

Example 1

Consider that you want to create the following defaults settings:

- All MRS files should be saved as .xml in the mrs_dir directory.
- All IP-XACT files should be saved as .xml in the xml_dir directory.
- All Tcl files should be saved as .tcl in the tcl_dir directory.

To create the above settings, specify the following Tcl commands:

```
set_save_template mrs "mrs_dir/%obj.xml"
set_save_template ipxact "xml_dir/%obj.xml"
set_save_template Tcl "Tcl_dir/%obj.tcl"
```

Example 2

If you want to save all the newly created MRS files as .xml files in the mrs_dir/
<obj-library-name> directory, specify the following command:

```
set_save_template mrs "mrs_dir/%library/%obj.xml" .
```

After executing the above command, if you create a new MRS object, abc, whose library name is mylib, GenSys saves that object as abc.xml in the /mrs_dir/mylib/ directory. However, if you have specified a default save directory, say mydir, by using the [set_default_save_dir](#) command, GenSys saves the new MRS objects under the /mydir/mrs_dir/mylib/ directory.

set_strict_ipxact

Connects exact indices of the same logical ports at both ends while making interface connection

Usage

```
set_strict_ipxact
```

Description

The `set_strict_ipxact` command connects the exact indices of the same logical ports at both ends while making interface connections.

Consider a design, `des`, that has two instances, `I1` and `I2`. Instance `I1` has interface `intf1` and instance `I2` has interface `intf2`. Now consider that an interface connection is to be made between the logical ports, `L1[22:2]` and `L1[23:0]` of interfaces, `I1.intf1` and `I2.intf2`, respectively. If you do not specify the `set_strict_ipxact` command, GenSys connects the logical ports of the two interfaces in the following manner:

For MSB Aligned Connection (Default)	<code>L1[22:2]</code> of <code>intf1</code> gets connected to <code>L1[23:3]</code> of <code>intf2</code>
For LSB Aligned Connection	<code>L1[22:2]</code> of <code>intf1</code> gets connected to <code>L1[20:0]</code> of <code>intf2</code>

However, if you specify the `set_strict_ipxact` command, GenSys connects the exact indices of the logical ports at both the ends, that is, `L1[22:2]` of `intf1` gets connected to `L1[22:2]` of `intf2`.

Note:

When you run this Tcl command, all the connections are LSB-aligned by default. However, if you do not run this Tcl command, the connections are MSB-aligned by default, unless specified otherwise.

Note:

The physical port connections will depend on how the logical and physical ports are mapped.

Example

Consider a design, `des`, that has an interface, `RAM`, which has a logical port, `wdata[38:0]`. An instance, `test0`, instantiated in the design, has an interface, `RAM`, with `wdata[31:2]`, `wdata[32:32]`, and `wdata[36:33]` mapped to three different ports. At design level, there is an interface connection.

Now, when you specify the `set_strict_ipxact` command, the generated RTL will appear as follows:

```
test test0 ( //RAM
    .pric_wdata(P1[31:2]) , //O:30 wdata Write Data
    //End of Interface RAM
    .pric_wdata_ecc(P1[32]) , //O:1 wdata
    .pric_wdata_ecc1(P1[36:33]) //O:4 wdata
);
```

However, if you do not specify the `set_strict_ipxact` command, GenSys will resume its default behavior and will generate following the RTL (MSB aligned):

```
test    test0 (  
    //RAM  
    .pric_wdata(P1[33:4]) ,    //O:30 wdata Write Data  
    //End of Interface RAM  
    .pric_wdata_ecc(P1[34])    ,    //O:1 wdata  
    .pric_wdata_ecc1(P1[38:35])    //O:4 wdata  
);
```

set_template_extension

Sets template file extension.

Usage

```
set_template_extension <extn>[ -append <extn> ]
```

Description

The `set_template_extension` command sets the extension of the template file.

Arguments

The `set_template_extension` command has the following arguments:

`<extn>`

Specifies the extension of the template.

`-append <extn>`

(Optional) Appends a template extension to the existing set of template extensions.

set_template_path

Sets template file path.

Usage

```
set_template_path <dir-path>
```

```
[ -append <dir-path> ]
```

Description

The `set_template_path` command sets a directory path in which the template files are searched, if an absolute path is not specified for them.

Arguments

The `set_template_path` command has the following arguments:

`<dir-path>`

Specifies the directory path.

`-append <dir-path>`

(Optional) Appends the *<dir-path>* directory path to the existing set of directory paths.

set_template_vars

Sets variables to be used in template files

Usage

```
set_template_vars    -var-name <value> *
```

Description

The `set_template_vars` command sets variables to be used in template files used for running the Template generator.

Arguments

The `set_template_vars` command has the following arguments:

`<var-name> <value> *`

Specifies name of a variable and its corresponding value.

Note:

This argument can have multiple occurrences. Therefore, you can specify any number of variables using a single command.

set_ui_options

Enables or disables refreshing of the incremental schematic

Usage

```
set_ui_options -incremental_refresh <true | false>
```

Description

The set_ui_options command enables or disables the refreshing of incremental schematic window.

set_verilog_dir

Sets a directory for Verilog files

Usage

```
set_verilog_dir <dir-name>
```

Description

The set_verilog_dir command specifies a directory that stores Verilog files generated by GenSys Verilog generator.

Preference of a Default Directory for Storing Verilog Files

The following preference is used (from the highest priority) to store Verilog files:

1. Directory specified by the set_verilog_dir command.
2. Directory specified by the [set_netlist_dir](#) command (if set_verilog_dir is not specified).

3. The output/Verilog directory (if none of the above commands is specified).

Note:

In case of partitions, GenSys creates the *<default-verilog-dir>/<partition-name>* sub directories.

set_work_dir

Sets the working directory.

Usage

```
set_work_dir <dir-name>
```

Description

The set_work_dir command specifies a directory *<dir-name>* (with the full path) as the working directory.

By default, the working directory is the current working directory (from where GenSys is invoked).

Examples

The following example sets the current working directory as the working directory:

```
set_work_dir .
```

Note:

To refer to the testcase of the above example, go to \$SPYGLASS_HOME/ examples/ Genesis/case6 directory.

set_xml_dir

Specifies the directory for storing XML files.

Usage

```
set_xml_dir <dir-name>
```

Description

The `set_xml_dir` command specifies a directory *<dir-name>* for storing GenSys XML files.

By default, XML files are saved in the db directory.

You can also set the XML directory in the following other ways:

- Using the `-XML_DIR` command line option while invoking GenSys
- Selecting the File > Set XML Directory menu option in GenSys GUI

Examples

The following example sets `xml_dir` located at `./Work_dir` as the directory for storing GenSys XML files:

```
set_xml_dir ./Work_dir/xml_dir
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case6` directory.

set_xslt

Specifies the XSLT file applied to the IP-XACT files being imported/exported

Usage

```
set_xslt
  -import <xslt-file> | -export <xslt-file>
  [ -version <version> ]
  [ -clear]
```

Description

The `set_xslt` command specifies the XSLT file to be applied to the IP-XACT files that are imported/exported in GenSys.

Arguments

The `set_xslt` command has the following arguments:

`-import <xslt-file>`

Specifies the XSLT file to be applied to the IP-XACT files that are imported in GenSys.

`-export <xslt-file>`

Specifies the XSLT file to be applied to the IP-XACT files that are being exported.

`-version <version>`

(Optional) Specifies the IP-XACT version of the IP-XACT files on which XSLT would be applied.

`-clear`

(Optional) Clears all the previously defined XSLT files.

split_struct_ports

Splits structure ports present in the design

Usage

```
split_struct_ports
    [-master mastername]
    [-all]
    [-process_leaf_components <TRUE|FALSE>]
```

Description

The `split_struct_ports` command is used to split structure ports present in the design.

Note:

The `split_struct_ports` Tcl command generates the "Split_Struct_Ports.rpt" report file in the report directory. This report is overwritten when you run the `split_struct_ports` Tcl command again.

Arguments

The description of the arguments is as follows:

`master`

Specifies the master for splitting struct ports

`all`

Processes all designs/subsystems present in the hierarchy but ignores leaf-level component (IP) for splitting struct ports

`process_leaf_components`

Use this option only with the `-all` argument. If this option is set to `TRUE`, GenSys also processes leaf components/modules present in the hierarchy.

Example

The following command specification splits structure ports present in a particular design:

```
split_struct_ports -master <design name>
```

You can also use `-all` options to apply this transformation in all designs/subsystems.

suppress_message

Suppresses a message to hide it from the message window

Usage

```
suppress_message  
[ <msg-id> ]  
[ -text <string> ]
```

Description

The `suppress_message` command is used to suppress messages in GenSys. After executing this Tcl command, messages are neither shown in the message window nor saved in the log file `gensys.log`.

You can suppress messages by either specifying the message ID or by specifying a string in a message.

Arguments

The `suppress_message` command has the following arguments:

`<msg-ID>`

(Optional) Specifies the message ID for the message you want to suppress. You cannot suppress multiple messages with the same command.

`-text <string>`

(Optional) Specifies a string that exists in the message you want to suppress. This argument suppresses all messages that have the `<string>` string.

syncup_instance

Syncs up all the instances of the specified master

Usage

```
syncup_instance
[ -master_name <name> ]
[ -vendor <vendor_name> ]
[ -version <version_num> ]
[ -library <library_name> ]
```

Description

The `syncup_instance` command syncs up all the instances of the master specified by `-master_name` argument of this command.

Arguments

The `syncup_instance` command has the following arguments:

`-master_name <name>`

(Optional) Specifies the name of the master whose instances should be synced.

If you do not specify this argument, GenSys syncs all instances in the entire hierarchy.

`-vendor <vendor_name>`

(Optional) Specifies the vendor of the master.

`-version <version_num>`

(Optional) Specifies the version number of the master.

`-library <library_name>`

(Optional) Specifies the library name of the master.

templegen

Runs the specified template file and generates data in the specified output file

Usage

```
templegen
  -template <file-name>
  -output <file-name>
  [ -output_path <dir> ]
```

Description

The templegen command runs the specified template file and generates data in the specified output file.

Arguments

The templegen command has the following arguments:

`-template <file-name>`

Specifies a template file

`-output <file-name>`

Specifies an output file in which data should be generated

`-output_path <dir>`

Specifies the directory in which the output file should be generated.

transform_xml

Transforms the given XML file by applying the specified XSLT stylesheet

Usage

```
transform_xml <xml-file>    [ <output-dir> ]
```

Description

The transform_xml command transforms the given XML file by applying the XSLT stylesheet(s) specified by using the [set_xslt](#) Tcl command. GenSys first applies the import XSLT files followed by the export XSLT files.

GenSys generates the output file in the specified output directory, *<output-dir>*. If you do not specify an output directory, GenSys generates the output file in the directory specified by using the [set_output_dir](#) Tcl command.

ungroup_design

Dissolves the hierarchy and brings the instances directly under the top design

Usage

```
ungroup_design
  -name <group-name>
  [ -design <top-design-name> ]
  [ -audit ]
  {
    [ -flatten <TRUE | FALSE> ]
    | [ -level <int value> ]
  }
  {
    [ -prefix <prefix> ] [ -postfix <postfix> ]
    | [ -preserve_hierarchy <TRUE | FALSE> ]
    | [ -hierarchy_separator <TRUE | FALSE> ]
  }
```

Description

The `ungroup_design` command dissolves the hierarchy and brings the instances inside it at an upper level, directly under the top design.

When you specify this Tcl command, GenSys evaluates all the dependant parameters of instances of the design getting ungrouped.

By default, the `ungroup_design` command accepts hierarchical instances:

```
ungroup_design -name {M1::I1}
```

The tool automatically detects whether the specified instances are hierarchical or not.

The default hierarchy separator is the double colon (::). To use a different hierarchy separator symbol, specify the symbol with the `set_hierarchy_separator` command. The following example shows how to use the forward slash (/) as the hierarchy separator.

```
set_hierarchy_separator /  
ungroup_design -name {M1/I1}
```

When components within an instance of a multiply-instantiated module are ungrouped, the `ungroup_design` command uniquifies the module instances by default.

In the following example, the top module has instances, M1 and M2, of the subsystem, mid. The tool ungroups only one of the instances' path.

```
ungroup_design -name {M1::I1}
```

The tool uniquifies mid as mid_0 and ungroups the mid_0 components. The tool uses integers (such as _0,_1,_2,...) to name uniquified instances.

Arguments

The `ungroup_design` command has the following arguments:

`-name <group-name>`

Specifies the name of the group that is to be ungrouped.

`-design <top-design-name>`

(Optional) Specifies the name of the design that contains the group. If the design is not specified, the specified group from the currently active design will be ungrouped.

`-audit`

(Optional) Generates an audit report named, `<group-name>_ungroup_audit.rpt`. When you specify this argument, GenSys does not perform the ungrouping. Rather, it generates this audit report, which contains the following details:

- *Port properties that are not handled during ungroup*

Audit report specifies any default tie-offs and default opens present on the ports on the group. During ungrouping, GenSys does not consider such default properties as connections and does not propagate them to the unconnected terminals on the other side.

- *Connection properties that are not handled during ungroup*

Audit report displays the attributes, datasources, descriptions, glue expressions, or visibility information on the connections crossing the hierarchy. Such properties are not handled during ungroup, and are lost after the ungroup operation.

This report also displays the net/logical connections present inside a group. Currently, GenSys does not handle such connections.

In addition, this report also displays the open/temporary open connections on the port/terminal of the hierarchy being ungrouped. Such connections are not propagated to the terminal on the other side.

Based on the information generated by this report, you can modify your design before performing the ungroup operation.

Note:

Currently, logical connections (nets) and cross-hierarchical connections are not handled.

`-flatten <TRUE | FALSE>`

(Optional) If you set this argument to TRUE, the ungroup operation is performed on a sub-system to resolve all nested hierarchies and all internal sub-systems are ungrouped recursively.

By default, the value is FALSE, which means only one level ungroup operation is performed.

`-level <int-value>`

(Optional) Specifies a level (positive integer value) so that the ungroup operation occurs till the specified level or less number of levels if no more nested sub-systems are present.

The default value of this argument is 1, which means the ungroup operation occurs till one level.

`-prefix <prefix>`

(Optional) Specifies a prefix for the name of the instance being ungrouped.

`-postfix <postfix>`

(Optional) Specifies a postfix for the name of the instance being ungrouped.

`-preserve_hierarchy <TRUE | FALSE>`

(Optional) If this argument is to TRUE, a sub-system instance name followed by an underscore (`_`) is prefixed to the name of the instance being ungrouped.

`-hierarchy_separator <TRUE | FALSE>`

(Optional) If this argument is to TRUE, the hierarchy separator specified by using the [set_hierarchy_separator](#) command is used instead of `_` between a sub-system name and an instance name to create an instance name after ungroup.

Default value is FALSE, which means `_` is used as separator to be used.

uniquify_instance

Modifies the specified instance

Usage

```
uniquify_instance
  -name <instance-name>
  -master <new-master-name>
  [ -vendor <vendor-name> ]
  [ -version <version-num> ]
  [ -library <library-name> ]
  [ -param <name>=<value> ]
```

Description

The `uniquify_instance` command modifies the specified instance of the active design by creating a new master of that instance and updating that master.

In GenSys, there can be scenarios where a design/sub-system/component has multiple instantiations. Certain operations, such as `move_instance`, require that each component should have a unified path with reference to the top design.

This command creates a copy of the master view of a specified instance.

Arguments

The `uniquify_instance` command has the following arguments:

`-name <instance-name>`

Specifies the name of the instance to be modified.

The instance name can be a hierarchical name separated by a hierarchy separator. For example, the `top::i0::i1::i2` name uses the hierarchy separator as `::`. You can change the hierarchy separator by using the [set_hierarchy_separator](#) Tcl command.

`-master <new-master-name>`

Specifies the name of the new master for the instance to be modified.

`-vendor <vendor-name>`

(Optional) Specifies the vendor of the new master.

`-version <version-num>`

(Optional) Specifies the version number of the new master.

`-library <library-name>`

(Optional) Specifies the library name of the new master.

`-param <name>=<value>`

(Optional) Specifies the parameter name-value pair.

Specify the parameter value that should override the parameter value of the instance whose master contains the *generate* statement.

Note:

The `-param` argument is added for a specific customer flow and is not recommended for basic RTL assembly and restructuring flow.

unsuppress_message

Resets a suppressed message to make it appear in the message window

Usage

```
unsuppress_message
    <msg-id>
    [ -text <string> ]
```

Description

The `unsuppress_message` command is used to unsuppress messages in GenSys. After executing this Tcl command, suppressed messages are shown in the message window and saved in the log file `gensys.log`.

You can unsuppress messages by either specifying the message ID or by specifying a string in a message.

Arguments

The `unsuppress_message` command has the following arguments:

`<msg-ID>`

(Optional) Specifies the message ID for the message you want to unsuppress. You cannot suppress multiple messages with the same command.

`-text <string>`

(Optional) Specifies a string that exists in the message you want to unsuppress. This argument suppresses all messages that have the `<string>` string.

update_design_configuration

Updates the design configuration in the root

Usage

```
update_design_configuration
  -name <designConfiguration-name>
  [ -vendor_name <vendor-name> ]
  [ -library_name <library-name> ]
  [ -version_num <version-number> ]
  -design_name <design-name>
  [ -design_vendor <design-vendor> ]
  [ -design_version <design-version> ]
  [ -design_library <design-library> ]
  { -instance_name <design-instance-name>
    -model_view <active-model-view-name> }*
```

Description

The `update_design_configuration` command updates the design configuration in the root.

Arguments

The `update_design_configuration` command has the following arguments:

`-name <designConfiguration-name>`

Specifies the design configuration name.

`-vendor_name <vendor-name>`

(Optional) Specifies the vendor name of the design configuration to be updated.

`-library_name <library-name>`

(Optional) Specifies the library name of the design configuration to be updated.

`-version_num <version-number>`

(Optional) Specifies the version number of the design configuration to be updated.

`-design_name <design-name>`

Specifies the name of the design.

`-design_vendor <design-vendor>`

(Optional) Specifies the vendor name of the design.

`-design_version <design-version>`

(Optional) Specifies the version number of the design.

`-design_library <design-library>`

(Optional) Specifies the library name of the design.

`-instance_name <design-instance-name>`

Specifies the instance name of the design.

`-model_view <active-model-view-name>`

Specifies the active model view name of the design.

update_logicalport_constraints

Updates constraints for the specified logical port

Usage

```
update_logical_port_constraints
  -logical_port_name <logical-port-name>
  [ -on_master_presence <optional|required|illegal> ]
  [ -on_master_width <width> ]
  [ -on_master_direction <in | inout | out> ]
  [ -on_slave_presence <optional|required|illegal> ]
  [ -on_slave_width <width> ]
  [ -on_slave_direction <in | out | inout> ]
  [ -on_system_group {<list-of-group-names> } ]
  [ -on_system_presence {<groups-presence-list> } ]
  [ -on_system_direction {list-of-directions> } ]
  [ -on_system_width {list-of-widths> } ]
```

Description

The `update_logical_port_constraints` command updates constraints for the specified logical port.

Arguments

The `update_logical_port_constraints` command has the following arguments:

`-logical_port_name <logical-port-name>`

Specifies the name of the logical port for which constraints should be updated.

`-on_master_presence <optional|required|illegal>`

(Optional) Specifies the presence value based on which existence of port is controlled.

`-on_master_width <width>`

(Optional) Specifies width of the port when it is present on a master bus interface.

`-on_master_direction <in | inout | out>`

(Optional) Specifies direction of the port when it is present on a master bus interface

`-on_slave_presence <optional|required|illegal>`

(Optional) Specifies the presence value based on which existence of port is controlled.

`-on_slave_width <width>`

(Optional) Specifies width of the port when it is present on a slave bus interface.

`-on_slave_direction <in | out | inout>`

(Optional) Specifies direction of the port when it is present on a slave bus interface

`-on_system_group {<list-of-group-names>}`

(Optional) Specifies a space-separated list of on-system group names.

`-on_system_presence {<groups-presence-list>}`

(Optional) Specifies a space-separated list of presence for corresponding groups.

`-on_system_direction {list-of-directions>}`

(Optional) Specifies a space-separated list of directions for corresponding groups.

`-on_system_width {list-of-widths>}`

(Optional) Specifies a space-separated list of widths for corresponding groups.

update_master

Updates the IP in the current or all designs in the library or updates the master of a single instance or updates the specified design itself

Usage

Usage 1:

```
update_master
<current | all | entity>
-old -name <Old-IP-name>
[ -vendor <Old-IP-vendor-name> ]
[ -version <Old-IP-version-num> ]
[ -library <Old-IP-library-name> ]
-new -name <New-IP-name>
[ -vendor <New-IP-vendor-name> ]
[ -version <New-IP-version-num> ]
[ -library <New-IP-library-name> ]
[ -replace_ports <P1->P2> ]*
[ -replace_parameters <p1->p2> ]*
```

Usage 2:

```
update_master
  -instance <instance_name>
  -name <new_IP_name>
  [-vendor <vendor_name>]
  [-version <version_num>]
  [-library <library_name>]
  [ -replace_ports <P1->P2> ]*
  [ -replace_parameters <p1->p2> ]*
```

Description

The `update_master` command updates the IP in the current or all designs in the library or updates the master of a single instance or updates the specified design itself.

By default, the `update_master` command accepts hierarchical instances:

```
update_master -instance {M1::U1} -name W_SUB1
```

The tool automatically detects whether the specified instances are hierarchical or not.

The default hierarchy separator is the double colon (::). To use a different hierarchy separator symbol, specify the symbol with the `set_hierarchy_separator` command. The following example shows how to use the forward slash (/) as the hierarchy separator.

```
set_hierarchy_separator /
update_master -instance {M1/U1} -name W_SUB1
```

When the master is updated for one or more instances of a multiply-instantiated module, the `update_master` command uniquifies the module instances by default.

In the following example, the top module has instances, M1 and M2, of the subsystem, mid. The master is updated for only one of the instances' path.

```
update_master -instance {M1::U1} -name W_SUB1
```

The tool uniquifies mid as mid_0 and changes the master of the instance, U1, to a new master in mid_0. The tool uses integers (such as _0,_1,_2,...) to name uniquified instances.

Usage 1:

In Usage 1, the `update_master` command updates the IP in the current or all designs in the library or updates the specified design itself.

In this case, you need to specify the names and other VLNV details of the old and new IPs. You also need to specify any of current, all, or entity options to specify the required action for the `update_master` command.

Usage 2:

In Usage 2, the `update_master` command updates the master of a single instance in a design.

In this case, you need to specify the name of the instance for which you need to update the master and the name and other VLVN details of the new IP.

Update master retains the connectivity of the IP instance view by doing a name based matching of old versus new IP ports. It retains the connections for ports having the same name/direction/type between new and old IPs. Use this command only when the old and new IPs match in their interface view, i.e. ports/parameter. GenSys restores the connections of matching port names. This may result into connection loss when the port's name/direction/type do not match.

Arguments

The `update_master` command has the following arguments:

`current`

This option is applicable only for usage 1.

Specifies that an IP needs to be replaced with another IP in the currently active design.

Note:

`current` is the default option.

`all`

This option is applicable only for usage 1.

Specifies that an IP needs to be replaced with another IP in all the designs loaded in the library browser.

`entity`

This option is applicable only for usage 1.

Specifies that the design itself needs to be updated. In this case, only the design is updated and the instances inside the design remain unchanged.

While updating the design, the ports and interfaces on the current design are replaced with the ports and interfaces of the new master. To preserve connections, you can provide mapping between old ports/interfaces/parameters and the new ports/interfaces/parameters using the update Replace IP dialog. For more information on the Replace IP dialog, refer to *GenSys User Guide*.

In addition, the connections between the design ports/interfaces and internal instances and all the exported pins and interfaces are also preserved. The values of parameters are also maintained.

`-old -name <Old-IP-name>`

This option is applicable only for usage 1.

Specifies the name of the old IP that needs to be replaced.

`-vendor <Old-IP-vendor-name>`

(Optional) Specifies the name of the vendor of the old IP.

`-version <Old-IP-version-num>`

(Optional) Specifies the version number of the old IP.

`-library <Old-IP-library-num>`

(Optional) Specifies the library name of the old IP.

`-new`

This option is applicable only for usage 1. This option needs to be followed by the argument, `-name`.

Specifies that the old IP is to be replaced with a new IP.

`-name <New-IP-name>`

This option is applicable for both usage 1 and usage 2.

Specifies the name of the new IP that replaces the old IP.

`-vendor <New-IP-vendor-name>`

(Optional) Specifies the name of the vendor of the new IP.

`-version <New-IP-version-num>`

(Optional) Specifies the version number of the new IP.

`-library <New-IP-library-name>`

(Optional) Specifies the library name of the new IP.

`-instance <instance-name>`

This option is applicable only for usage 2.

Specifies name of the instance for which the master is to be updated.

`-replace_ports`

(Optional) Replaces a port of an IP with the specified port. You can use this argument multiple times to replace multiple ports.

While updating the master, the connectivity is restored using named based port match between old and new master. You can specify a mapping of old versus new port names.

For details, see [Example 4](#).

`-replace_parameters`

(Optional) Replaces a parameter of an IP with the specified parameter. You can use this argument multiple times to replace multiple parameter. For details, see [Example 5](#).

Examples

Example 1

The following command replaces an IP, comp1, with another IP, comp2, in the currently active design.

```
update_master -old -name comp1 -new -name comp2
```

The default option, current, is used to update the IP, comp1, in the current design.

Example 2

The following command replaces the design, des, with another design, des_new.

```
update_master entity -old -name des  
-new -name des_new
```

Example 3

The following command replaces the master of instance comp0 only. It will not change the master of comp1 though both the instances have same master.

```
new design des  
add_instance -name comp0 -master comp  
add_instance -name comp1 -master comp  
update_master -instance comp0 -name comp_new
```

Example 4

Consider the following command containing the `-replace_ports` arguments:

```
update_master -old -name c1 -version 1.0 -library l
  -vendor v -new -name c2 -version 1.1 -library l
  -vendor v -replace_ports P1->P7 -replace_ports P2->P6
```

The above example indicates that:

- The P1 port in the old component c1 is mapped to the P7 port in new master c2.
- The P2 port in the old component c1 is mapped to the P6 port in the new master c2.

Therefore, all connections related to the P1 port will have the new port reference as P7 and all the connections related to the P2 port will have the new port reference as P6.

Example 5

Consider the following command containing the `-replace_parameters` arguments:

```
update_master -old -name c1 -version 1.0 -library l
  -vendor v -new -name c2 -version 1.1 -library l
  -vendor v -replace_parameters p10->p70
  -replace_parameters p20->p60
```

The above example indicates that:

- The p10 parameter in the old component c1 is mapped to the p70 parameter in the new master c2.
- The p20 parameter in the old component c1 is mapped to the p60 parameter in the new master c2.

Therefore, wherever the p10 parameter is used, its name and value is replaced by the name and value of the p70 parameter. Similarly, name and value of the p20 parameter is replaced by the name and value of the p60 parameter.

update_master_interface

Begins or ends the process of editing ports of the master of the specified instance

Usage

```
update_master_interface
```

```
-name <component-name>
[ -vendor <vendor-name> ]
[ -version <version-num> ]
[ -library <library-name> ]
-start | -end
```

Description

The `update_master_interface` command begins or ends the process of adding, updating, or deleting ports of the master of the specified instance.

All the port editing is done by the [add_port](#), [delete_port](#), and [update_port](#) command. For details, see and [Examples](#).

The need to add, update, or delete ports arises in case the RTL boundary of a subsystem is frozen. In such cases, it is important to keep the GenSys-generated ports same as what is present in the RTL boundary.

Arguments

The `update_master_interface` command has the following arguments:

`-name <component-name>`

Specifies the name of the component whose master ports should be updated.

`-vendor <vendor-name>`

(Optional) Specifies the vendor name.

`-version <version-num>`

(Optional) Specifies the version.

`-library <library-name>`

(Optional) Specifies the library name.

`-start`

Begins the process of adding, deleting, or updating ports on the master. The type of editing is specified by the [update_port](#) command.

`-end`

Stops the process of adding, deleting, or updating ports on the master. After specifying this argument, updates specified after the `-start` argument of this command are applied on the master.

Examples

Example 1

Consider the following commands:

```
update_master_interface -name comp1 -start
update_port -name p1 -optimize
update_port -name p2 -rename pr2
update_port -name p3 -split
add_port -name p4
delete_port p5
update_master_interface -name comp1 -end
```

When you run the above commands, GenSys optimizes the p1 port, renames the p2 port, splits the p3 port, adds the p4 port, and deletes the p5 port of the comp1 component.

Example 2

See [Example](#) of the [update_port](#) command.

update_port

Updates the specified port of the master of an instance

Usage

Usage 1:

```
update_port
  -name <port-name>
  -rename <new-port-name>
  [ -msb <msb> ]
  [ -lsb <lsb> ]
```

Usage 2:

```
update_port
  -name <port-name>
  [ -optimize | -split ]
```

Usage 3:

```
update_port
  -optimize -all
```

Description

The `update_port` command updates a port of the master of an instance specified by the [update_master_interface](#) command.

See [Example](#).

Note:

This command supports only those ports that are not parameterized.

Arguments

The `update_port` command has the following arguments:

`-name <port-name>`

Specifies the name of the port to be updated.

This is the port of the master of an instance specified by the [update_master_interface](#) command.

`-rename <new-port-name>`

Specifies a new name for the port.

`-msb <msb>`

(Optional) Specifies the MSB of the port.

`-lsb <lsb>`

(Optional) Specifies the LSB of the port.

`-optimize`

(Optional) Optimizes the specified port by deleting its unconnected bits.

`-all`

(Optional) Optimizes all the ports of the master by deleting their unconnected bits. This argument can only be used with the `-optimize` argument.

`-split`

(Optional) Splits the specified port into multiple ports based on current connectivity.

Example

Consider that you specify Tcl commands, as shown in the following steps:

- **Step 1:** The following commands create a component, instantiate that component in a design, and generate connections:

```
new component comp1
add_port -name p1 -direction INOUT -lsb 0 -msb 10
add_port -name p2 -direction INOUT -lsb 0 -msb 10
add_port -name p3 -direction INOUT -lsb 0 -msb 10
save
new design d
add_instance -name inst1 -master comp1
add_port -name dp1 -direction INOUT -lsb 0 -msb 10
add_connection -instance inst1 -pin p1 -lsb 0 -msb 2
               -port dp1 -lsb 8 -msb 10
add_connection -instance inst1 -pin p2 -lsb 4 -msb 6
               -port dp1 -lsb 0 -msb 2

               -port dp1 -lsb 4 -msb 6
```

- **Step 2:** The following command starts the process of updating ports of the master of comp1:

```
update_master_interface -name comp1 -start
```

- **Step 3:** The following commands updates the ports of the master of comp1:

```
update_port -name p1 -optimize
update_port -name p2 -rename pr2
update_port -name p3 -split
```

- **Step 4:** The following command ends the process of updating ports of the master of comp1:

```
update_master_interface -name comp1 -end
```

- **Step 5:** The following command generates Verilog:

```
run GenerateVerilog
```

After performing the above steps, the following Verilog file is generated containing updated port names of the master of comp1:

```
module comp1 (p1      ,
              pr2     ,
              p3_4_6);
  inout  [2:0]  p1;
  inout  [10:0] pr2;
  inout  [2:0]  p3_4_6;
endmodule

module d (dp1);
  inout  [10:0] dp1;
  wire   [7:0]  UNINTENTIONAL_OPEN_inst1_pr2_3;
  comp1  inst1 (
              .p1(dp1[10:8])      , //IO:3
```

```
.pr2({UNINTENTIONAL_OPEN_inst1_pr2_3,  
      dp1[2:0]}), //IO:11  
      .p3_4_6(dp1[6:4]) //IO:3  
);  
  
endmodule
```

update_row

Updates the specified column values of the current row

Usage

```
update_row  
{ -<col-name> <value> }*
```

Description

The `update_row` command updates the values of the specified column(s) of the current row.

You can specify the `-<col-name>` argument multiple times to update the values of more than one column in a command.

If you update a row of an IP-XACT table, GenSys updates the corresponding node in the XML tree of the IP-XACT Viewer pane.

Examples

The following example updates the size column value to 8.

```
update_row -size 8  
The following example updates the size column value to 8 and the MSB  
column value to 5.  
update_row -size 8 -MSB 5
```

The following example updates the AttributeName column to Atr0, Value column to 1, and Type column to ATTR_INT of the current row:

```
update_row -AttributeName Atr0 -Value 1 -Type ATTR_INT
```

Note:

To refer to the testcase of the above example, go to `$SPYGLASS_HOME/ examples/ Genesis/case0` directory.

update_rule

Updates rule severity; Enables or disables rules

Usage

```
update_rule
  -severity <INFO | WARNING | ERROR | FATAL_ERROR>
  -enable <TRUE | FALSE>
  [-id <Space-Separated-List-Of-Rules>]
  [-category <category>]
  [-sub_category <subCategory>]
  [-all]
```

Description

The update_rule command updates the severity of rules or enables/disables the rules.

Arguments

The update_rule command has the following arguments:

`-severity <INFO | WARNING | ERROR | FATAL_ERROR>`

Defines the new severity of a rule or a list of rules. You can specify the severity as INFO, WARNING, ERROR, or FATAL_ERROR.

`-enable <TRUE | FALSE>`

Set the value to FALSE to not run a rule. By default, all rules are enabled.

`-id <Space-Separated-List-Of-Rules>`

(Optional) Specifies rules that need to be updated.

`-category <category>`

(Optional) Specifies the category for the rules that need to be updated. If you specify both the ID and category arguments, the ID argument takes precedence.

`-sub_category <subCategory>`

(Optional) Updates all rules in the specified subcategory of the category specified through the -category argument. To use this argument, you first need to specify the -category argument. If you specify both the ID and category arguments, the ID argument takes precedence.

-all

(Optional) Updates all rules. This argument takes least precedence.

validate_ipxactxml

Validates the IP-XACT XML files.

Usage

validate_ipxactxml <yes | no>

Description

The validate_ipxactxml command performs validation of the IP-XACT XML files while loading these files in GenSys to ensure that the XML files are adhering to the schema definition provided in the XSD (XML Schema Definition) files.

The validate_ipxactxml command can be specified in the .gensys.setup file or in the Tcl command window in GenSys GUI.

Arguments

The validate_ipxactxml command has the following arguments:

<yes | no>

Specifies whether validation should be performed on the IP-XACT XML files while loading them in GenSys.

By default, validate_ipxactxml command is set to yes, and a validation is always performed on the IP-XACT XML files.

write_formality_script

Creates a formality file for the specified design

Usage

```
write_formality_script
  -top <design-name>
  -output <file-name>
```

Description

The `write_formality_script` command generates a file containing commands of the *Formality* tool.

This file is used to compare changes between a reference and implementation design.

`-top <design-name>`

Specifies the design for which the formality file should be created. This design should be any of the following:

- A design loaded from RTL
- A design saved in the MRS format in a previous GenSys session, but originally loaded from RTL

Before loading the design, set the `-write_mapping_file <Y | N>` argument of the `set_gensys_options` command to Y.

`-output <file-name>`

Specifies the name of the file containing commands of the *Formality* tool.

3

GenSys Tcl APIs

You can query GenSys database from Tcl interface, similar to the way you query from Perl. Querying from Tcl interface is useful for checking parameter values, checking connections, and so on, from within a Tcl script.

For example, you can get a list of instances within a design, use any one of the instances to traverse to the master, and so on.

Most API names start with `get_`, and return either a single object or a list of objects, where appropriate. API calls returning a list end in `s` (for example, `get_design_objects`), and API calls returning a single objects do not (for example, `get_name`). Objects are either strings (for example, `foo` or `37`) or references to objects in the database (for example, `A:attribute:12345678:0`). You do not need to understand these object ids. You simply need to pass them to the other `get_` methods to get meaningful values.

The following example iterates designs, instances, interfaces, and prints names:

```
set root [get_root]
foreach design [get_design_objects $root] {
    foreach inst [get_instances $design] {
        foreach ifinst [get_ifinstances $inst] {
            set intf [get_interface $ifinst]
            set master [get_master $inst]
            puts "design [get_name $design] instance [get_name $inst] master
[get_name $master] interface [get_name $intf]"
        }
    }
}
```

Note:

You can refer to a more complete example under <install-dir>/SPYGLASS_HOME/examples.

Object-to-Methods Tree

The Objects-to-Methods tree is described below where the top item is an object type and its leaves are the methods applicable to the object type with the method return type annotated as L for list, S for string, O for Object, and I for Boolean or Integer:

root

- |->L [get_design_objects](#)
 - |-> L [get_components](#)
 - |-> L [get_interfacedefs](#)
 - |-> O [get_tableschema](#)
 - |-> O [get_columnschema](#)
 - |->O [get_active_object](#)

design

- |-> L [get_instances](#)
 - |-> L [get_interfaces](#)
 - |-> L [get_ports](#)
 - |-> L [get_tables](#)
 - |-> L [get_aconnects](#)
 - |-> L [get_tconnects](#)
 - |-> L [get_iconnects](#)
 - |-> L [get_reset](#)
 - |-> L [get_align](#)
 - |-> L [get_partitions](#)
 - |-> L [get_parameters](#)
 - |-> L [get_attributes](#)
 - |-> S [get_datasource](#)
 - |-> S [get_name](#)
 - |-> S [get_description](#)
 - |-> S [get_xmlfile](#)
 - |-> S [get_shortname](#)
 - |-> O [get_vnlv](#)

|-> [do_elaborate](#)

|-> [do_unelaborate](#)

component

- |-> L [get_interfaces](#)
- |-> L [get_instances](#)
- |-> L [get_ports](#)
- |-> L [get_tables](#)
- |-> L [get_reset](#)
- |-> L [get_align](#)
- |-> L [get_reset](#)
- |-> L [get_memories](#)
- |-> L [get_parameters](#)
- |-> L [get_attributes](#)
- |-> S [get_datasource](#)
- |-> S [get_name](#)
- |-> S [get_xmlfile](#)
- |-> S [get_type](#)
- |-> S [get_isinternal](#)
- |-> S [get_description](#)
- |-> O [get_vnlv](#)
- |-> S [get_partition](#)

interfacedef

- |-> L [get_ports](#)
- |-> L [get_tables](#)
- |-> L [get_attributes](#)
- |-> S [get_datasource](#)
- |-> O [get_vnlv](#)
- |-> S [get_xmlfile](#)
- |-> S [get_name](#)
- |-> S [get_description](#)

|-> S [get_shortname](#)

interface

- |-> L [get_ports](#)
 - |-> L [get_attributes](#)
 - |-> O [get_interfacedef](#)
 - |-> S [get_datasource](#)
 - |-> S [get_name](#)
 - |-> S [get_type](#)
 - |-> I [get_ismirror](#)
 - |-> S [get_description](#)
 - |-> S [get_voltage](#)
 - |-> S [get_shortname](#)
 - |-> S [get_powerdomain](#)

parameter

- |-> L [get_children](#)
 - |-> S [get_name](#)
 - |-> S [get_value](#)
 - |-> S [get_type](#)

instance

- |-> L [get_tables](#)
 - |-> L [get_terminals](#)
 - |-> L [get_ifinstances](#)
 - |-> L [get_aconnects](#)
 - |-> L [get_tconnects](#)
 - |-> L [get_iconnects](#)
 - |-> L [get_attributes](#)
 - |-> S [get_datasource](#)
 - |-> S [get_name](#)
 - |-> S [get_description](#)
 - |-> S [get_voltage](#)

```
|-> S |-> O get_master
|-> S get_powerdomain
|-> L get_parameters
|-> S get_partition
|-> do_elaborate
|-> do_unelaborate
```

terminal

- |-> S get_name
|-> O get_port

port

- |-> L get_attributes
|-> S get_datasource
|-> S get_name
|-> S get_lname
|-> S get_shortcode|
-> S get_type
|-> S get_dir
|-> S get_default
|-> S get_lsb
|-> S get_msb
|-> S get_width
|-> I get_is_scalar
|-> S get_is_open
|-> S get_voltage
|-> S get_clockrate
|-> S get_powerdomain
|-> S get_optional
|-> S get_align
|-> S get_description
|-> O get_hdl_type

- |-> O [get_hdl_type_lsb](#)
- |-> O [get_hdl_type_msb](#)
- |-> O [get_hdl_type_language](#)
- |-> O [get_hdl_type_library](#)
- |-> O [get_hdl_type_package](#)
- |-> S [get_interfaceGroup](#)

lport

- |-> S [get_name](#)
- |-> S [get_lname](#)
- |-> S [get_actdir](#)
- |-> S [get_type](#)
- |-> S [get_dir](#)
- |-> S [get_default](#)
- |-> S [get_lsb](#)
- |-> S [get_msb](#)
- |-> S [get_width](#)
- |-> S [get_description](#)
- |-> I [get_reset](#)
- |-> L [get_attributes](#)
- |-> I [get_optional](#)
- |-> S [get_datasource](#)

aconnect

- |-> O [get_first](#)
- |-> O [get_second](#)
- |-> L [get_attributes](#)
- |-> O [get_datasource](#)
- |-> S [get_align](#)
- |-> S [get_plane](#)
- |-> I [get_is_elab_generated](#)
- |-> S [get_backref](#)

|-> S [get_command](#)
|-> S [get_morebackref](#)
|-> S [get_deltadelay](#)

tconnect

- |-> O [get_first](#)
|-> O [get_value](#)
|-> L [get_attributes](#)
|-> O [get_datasource](#)
|-> O [get_istemp](#)
|-> S [get_backref](#)
|-> S [get_command](#)
|-> I [get_is_elab_generated](#)
|-> S [get_morebackref](#)

iconnect

- |-> O [get_first](#)
|-> O [get_second](#)
|-> S [get_align](#)
|-> S [get_plane](#)
|-> S [get_backref](#)
|-> S [get_command](#)
|-> S [get_morebackref](#)
|-> S [get_deltadelay](#)

aconnect_1

- |-> S [get_lsb](#)
|-> S [get_msb](#)
|-> O [get_instance](#)
|-> O [get_terminals](#)
|-> O [get_parent](#)
|-> O [get_port](#)

aconnect_2

- |-> S |-> S [get_msb](#)
|-> O [get_instance](#)
|-> O [get_terminals](#)
|-> O [get_parent](#)
|-> O [get_port](#)

iconnect_1

- |-> O [get_instance](#)
|-> O [get_ifinstance](#)
|-> O [get_parent](#)
|-> L [get_tieoffs](#)

iconnect_2

- |-> O [get_instance](#)
|-> O [get_ifinstance](#)
|-> O [get_parent](#)
|-> L [get_splices](#)
|-> L [get_tieoffs](#)

ifinstance

- |-> O [get_interface](#)
|-> O [get_instance](#)
|-> L [get_terminals](#)
|-> L [get_iconnects](#)
|-> L [get_attributes](#)
|-> O [get_datasource](#)
|-> I [get_is_exported](#)
|-> S [get_is_open](#)

addressif

- |-> S [get_start](#)
|-> S [get_end](#)
|-> S [get_size](#)

|-> S [get_description](#)

|-> S [get_datasource](#)

|-> L [get_attributes](#)

registerobject

- |-> S [get_name](#)
 - |-> S [get_type](#)
 - |-> O [get_reset](#)
 - |-> L [get_attributes](#)
 - |-> O [get_datasource](#)
 - |-> O [get_reset](#)

register

- |-> L [get_category](#)
 - |-> L [get_attributes](#)
 - |-> S [get_datasource](#)
 - |-> S [get_name](#)
 - |-> S [get_width](#)
 - |-> S [get_width](#)
 - |-> S [get_attributes](#)
 - |-> S [get_offset](#)
 - |-> S [get_actdir](#)
 - |-> S [get_reset](#)
 - |-> S [get_resetmask](#)
 - |-> S [get_rowdata](#)
 - |-> O [get_description](#)

bitfield

- |-> S [get_name](#)
 - |-> S [get_functiontype](#)
 - |-> S [get_lsb](#)
 - |-> S [get_msb](#)
 - |-> O [get_reset](#)

- |-> S [get_resetmask](#)
- |-> S [get_aconnects](#)
- |-> S [get_actdir](#)
- |-> S [get_description](#)
- |-> S [get_aconnects](#)
- |-> S [get_rowdata](#)
- |-> I [get_second](#)
- |-> I [get_xmlfile](#)
- |-> I [get_width](#)
- |-> L [get_attributes](#)
- |-> S [get_datasource](#)
- |-> L [get_category](#)

bitenum

- |-> L [get_attributes](#)
 - |-> S [get_datasource](#)
 - |-> S [get_minvalue](#)
 - |-> S [get_maxvalue](#)
 - |-> S [get_actdir](#)
 - |-> S [get_name](#)
 - |-> I [get_value](#)
 - |-> S [get_description](#)

registergroup

- |-> O [get_reset](#)
 - |-> L [get_attributes](#)
 - |-> S [get_datasource](#)
 - |-> S [get_name](#)
 - |-> S [get_attributes](#)
 - |-> S [get_offset](#)
 - |-> S [get_attributes](#)
 - |-> S [get_shortcode](#)

|-> S [get_description](#)

memory

- |-> S [get_name](#)
 - |-> S [get_shortcode](#)
 - |-> I [get_maxdatawidth](#)
 - |-> S [get_actdir](#)
 - |-> S [get_offset](#)
 - |-> S [get_endian](#)
 - |-> S [get_size](#)
 - |-> S [get_attributes](#)
 - |-> S [get_description](#)
 - |-> S |-> L [get_attributes](#)
 - |-> S [get_datasource](#)

attribute

- |-> S [get_name](#)
 - |-> S [get_type](#)
 - |-> S [get_value](#)
 - |-> S [get_help](#)
 - |-> S [get_category](#)

partition

- |-> S [get_name](#)
 - |-> S [get_path](#)
 - |-> S [get_language](#)
 - |-> L [get_rtlfiles](#)

rtlfiles

- |-> S [get_name](#)

table

- |-> L [get_columndata](#)
 - |-> L [get_rowdata](#)
 - |-> L [get_colnames](#)

- |-> O [get_ptable](#)
- |-> S [get_name](#)
- |-> I [get_tid](#)
- |-> S [get_ptablecolname](#)
- |-> I [get_ptablerownum](#)
- |-> I [get_istab](#)
- |-> I [get_nrows](#)
- |-> I [get_ncols](#)
- |-> O [get_subtable](#)

`datasource`

- |-> S [get_person](#)
- |-> S [get_method](#)
- |-> S [get_reason](#)
- |-> S [get_datetime](#)
- |-> S [get_type](#)
- |-> I [get_frozen](#)
- |-> S [get_creationdate](#)

`vnlv`

- |-> S [get_version](#)
- |-> S [get_name](#)
- |-> S [get_library](#)
- |-> S [get_vendor](#)

`tableschema`

- |-> L [get_columnschemas](#)
- |-> L [get_hiddencolumns](#)
- |-> S [get_name](#)
- |-> S [get_hiername](#)
- |-> I [get_isevalcolpresent](#)
- |-> I [get_isevalallowed](#)

```
|-> I get\_isvalidcolpresent
|-> I get\_isvalidateallowed
|-> S get\_key
|-> I get\_ishidden
|-> I get\_isreadonly
|-> I get\_isbasetable
|-> I get\_isextensiontable
|-> I get\_isflattable
|-> I get\_istab
|-> S get\_rowtype
|-> I get\_line
|-> S get\_file
|-> S get\_help
|-> S get\_version
```

columnschema

- |-> L [get_dependcolumns](#)
|-> L [get_keycolumns](#)
|-> L [get_cellenums](#)
|-> O [get_tableschema](#)
|-> O [get_autofillschema](#)
|-> S [get_name](#)
|-> S [get_hiername](#)
|-> S [get_help](#)
|-> I [get_ishidden](#)
|-> I [get_isreadonly](#)
|-> I [get_isextension](#)
|-> I [get_iseval](#)
|-> I [get_iskey](#)
|-> I [get_isvalidate](#)
|-> S [get_celltype](#)


```
|-> S get\_file
|-> I get\_line

autofillschema
• |-> S get\_default

rtlcomponent
• |-> O get\_comcomponent
  |-> L get\_rtlinstances
  |-> L get\_rtlports
  |-> S get\_name

rtlinstance
• |-> O get\_masterrtlcomponent
  |-> O get\_parentrtlcomponent
  |-> S get\_name
```

Tcl Relation API Functions: Global API Functions

The Tcl relation API functions return a lists of objects or a single object, where applicable. This section provides a list of Relation APIs.

GenSys provides the following global API functions:

- [get_hierarchyseparator](#)
- [get_isMultipleInstantiations](#)
- [get_root](#)
- [get_report_dir](#)
- [get_work_dir](#)
- [get_xml_dir](#)
- [get_output_dir](#)

get_hierarchyseparator

Returns the current hierarchy separator string

Declaration

`get_hierarchyseparator`

Returns

Returns the current hierarchy separator string

Example

This example shows how to return the hierarchy separator set by using the `set_hierarchy_separator` Tcl method. If no separator is set, the default (:) is returned.

```
set_hierarchy_separator .  
....
```

Now, if you want to retrieve the already set hierarchy separator in a Tcl script, specify the following:

```
set var [get_hierarchyseparator]  
puts $var
```

The hierarchy separator string is contained in `$var`.

`get_isMultipleInstantiations`

Instantiation of a design

Declaration

`get_isMultipleInstantiations`

Returns

0 for single instantiation of a design/component

1 for multiple instantiations of a design/component

Example

```
set get_isMultipleInstantiations [$component]
```

```
set get_isMultipleInstantiations [$design]
```

get_root

Returns the comRoot object

Declaration

```
get_root
```

Returns

The comRoot object

Example

```
set root [get_root]
```

get_report_dir

Returns the name of the directory where reports are saved

Declaration

```
get_report_dir
```

Returns

Name of the directory where reports are saved

Example

```
puts[get_report_dir]
```

get_work_dir

Returns name of current working directory

Declaration

```
get_work_dir
```

Returns

Name of the current working directory

Example

```
puts[get_work_dir]
```

get_xml_dir

Returns directory name where XML files are saved

Declaration

```
get_xml_dir
```

Returns

Name of the directory where XML files are saved

Example

```
puts[get_xml_dir]
```

get_output_dir

Returns name of directory where output files are saved

Declaration

```
get_output_dir
```

Returns

Name of the directory where output files are saved

Example

```
puts[get_output_dir]
```

Tcl Relation API Functions: GenSys Built-In Tcl APIs

The Tcl APIs in this section are categorized into the following:

- [Autoconnect Applications](#)
- [Object Retrieval Applications](#)

Autoconnect Applications

Independent generators to achieve certain connectivity objectives. The APIs pertain to:

- Connect-by-name applications: The API available are as follows:

connect_by_name	connect_by_name_hier	connect_by_prefix
connect_by_prefix_hier	connect_by_postfix	connect_by_postfix_hier
connect_by_prefix_postfix	connect_by_prefix_postfix_hier	

- Tie-off applications: The API available are as follows:

tie_high_unconn	tie_high_unconn_hier
tie_low_unconn	tie_low_unconn_hier

- Export applications: The API available are as follows:

export_unconn_hier	export_unconn_hier_inputs
export_unconn_hier_matching_ports	export_unconn_hier_outputs

Object Retrieval Applications

Wrappers over current TCL APIs to get information about the currently working design hierarchy. The APIs pertain to:

- Get object-by-name applications: The API available are as follows:

get_design_by_name	get_instance_by_name	get_port_by_name
get_terminal_by_name		

- Connected/unconnected terminal/port applications: The API available are as follows:

get_all_unconnected_ports	get_all_unconnected_ports_hier	get_all_unconnected_terminals
get_all_unconnected_terminals_hier	get_all_connected_ports	get_all_connected_ports_hier
get_all_connected_terminals	get_all_connected_terminals_hier	

- Get tie-off terminal/port applications: The API available are as follows:

get_all_tied_ports	get_all_tied_ports_hier
get_all_tied_terminals	get_all_tied_terminals_hier

- Get port/terminal connections applications: The API available are as follows:

get_port_connections	get_terminal_connections
get_port_tieoff_connections	get_terminal_tieoff_connections

connect_by_name

Connects instance terminals and/or design ports with same name for specified design

Declaration

```
connect_by_name <design-name> [<port-name-list>]
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- port-name-list (Optional): If given, only port names present in the list are considered for connection. If no port names are given, the API considers all possible ports.

Examples

Example 1

This example shows the impact of using `connect_by_name` on a design. Since no specific ports are specified, all possible ports in the design are considered. The following table shows the original and generated RTLs after using `connect_by_name`.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(), .p2(), .p3(), .q1()); Mid2 mid2 (.p1(), .p2(), .p4(), .q1()); endmodule </pre>	<pre> connect_by_name top </pre>	<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(p1), .p2(p2), .p3(p3), .q1(q1)); Mid2 mid2 (.p1(p1), .p2(p2), .p4(p4), .q1()); endmodule </pre>

Example 2

This example shows the impact of using `connect_by_name` and specifying port names. The following table shows the original RTL and the RTL generated after using `connect_by_name`.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(), .p2(), .p3(), .q1()); Mid2 mid2 (.p1(), .p2(), .p4(), .q1()); endmodule </pre>	<pre> connect_by_name top {p1 p4} </pre>	<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(p1), .p2(), .p3(), .q1()); Mid2 mid2 (.p1(p1), .p2(), .p4(p4), .q1()); endmodule </pre>

connect_by_name_hier

Connects instance terminals and/or design ports with same name for complete hierarchy starting from specified design

Declaration

```
connect_by_name_hier <design-name> [<port-name-list>]
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- port-name-list (Optional): If given, only port names present in the list are considered for connection. If no port names are given, the API considers all possible ports.

Example

```
connect_by_name_hier top  
connect_by_name_hier top {p1 p4}
```

connect_by_prefix

Connects instance terminals and/or design ports with prefix for specified design

Declaration

```
connect_by_prefix <design-name> <prefix>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- prefix: Prefix string that needs to be present in port names to be considered for connection

Example

This example shows the impact of using `connect_by_prefix` with the `p1_` prefix. The following table shows the original RTL and the RTL generated after using `connect_by_prefix`

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(), .p2(), .p3(), .q1()); Mid2 mid2 (.p1(), .p2(), .p4(), .q1()); endmodule </pre>	<pre> connect_by_prefix top p1_ </pre>	<pre> module top (p1_1, p2_1, p3_1, p4_1, q1_1, q2_1); input [7:0] p1_1; input [7:0] p2_1; input [7:0] p3_1; input [7:0] p4_1; output [7:0] q1_1; output [7:0] q2_1; Mid1 mid1 (.p1_2(p1_1), .p2_2(), .p3_2(), .q1_2()); Mid2 mid2 (.p1_3(p1_1), .p2_3(), .p4_3(), .q1_3()); endmodule </pre>

connect_by_prefix_hier

Connects instance terminals and/or design ports with prefix for complete hierarchy starting from specified design

Declaration

```
connect_by_prefix_hier <design-name> <prefix>
```

Arguments

- `design-name`: Name of the design for which the rule needs to be run
- `prefix`: Prefix string that needs to be present in port names to be considered for connection

Example

```
connect_by_prefix_hier top a1_
```

connect_by_postfix

Connects instance terminals and/or design ports with suffix for specified design

Declaration

```
connect_by_postfix <design-name> <suffix>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- suffix: Suffix string that needs to be present in port names to be considered for connection

Example

This example shows the impact of using connect_by_postfix with the _p1 suffix. The following table shows the original RTL and the RTL generated after using connect_by_postfix.

Original RTL	Tcl Commands Run	Generated RTL
<pre>module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(), .p2(), .p3(), .q1()); Mid2 mid2 (.p1(), .p2(), .p4(), .q1()); endmodule</pre>	<pre>connect_by_postfix top _p1</pre>	<pre>module top (a_p1, a_p2, a_p3, a_p4, a_q1, a_q2); input [7:0] a_p1; input [7:0] a_p2; input [7:0] a_p3; input [7:0] a_p4; output [7:0] a_q1; output [7:0] a_q2; Mid1 mid1 (.b_p1(a_p1), .b_p2(), .b_p3(), .b_q1()); Mid2 mid2 (.c_p1(a_p1), .c_p2(), .c_p4(), .c_q1()); endmodule</pre>

connect_by_postfix_hier

Connects instance terminals and/or design ports with mentioned suffix for complete hierarchy starting from the specified design

Declaration

```
connect_by_postfix_hier <design-name> <postfix>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- postfix: Suffix string that needs to be present in port names to be considered for connection

Example

```
connect_by_postfix_hier top _a1
```

connect_by_prefix_postfix

Connects instance terminals and/or design ports with prefix and suffix for the specified design

Declaration

```
connect_by_prefix_postfix <design-name> <prefix> <suffix>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- prefix: Prefix string that needs to be present in port names to be considered for connection
- suffix: Suffix string that needs to be present in port names to be considered for connection

Example

This example shows the use of `connect_by_prefix_postfix` with the `port_` prefix and `_p1` suffix. The following table shows the original RTL and the RTL generated.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (p1, p2, p3, p4, q1, q2); input [7:0] p1; input [7:0] p2; input [7:0] p3; input [7:0] p4; output [7:0] q1; output [7:0] q2; Mid1 mid1 (.p1(), .p2(),.p3(),.q1()); Mid2 mid2 (.p1(), .p2(),.p4(),.q1()); endmodule </pre>	<pre> connect_by_prefix_postfix top port_ _p1 </pre>	<pre> module top (port_a_p1, port_a_p2, port_a_p3, port_a_p4, port_a_q1, port_a_q2); input [7:0] port_a_p1; input [7:0] port_a_p2; input [7:0] port_a_p3; input [7:0] port_a_p4; output [7:0] port_a_q1; output [7:0] port_a_q2; Mid1 mid1 (.port_b_p1(port_a_p1), .port_b_p2(),.port_b_p3(), .port_b_q1()); Mid2 mid2 (.port_c_p1(port_a_p1), .port_c_p2(),.port_c_p4(), .port_c_q1()); endmodule </pre>

connect_by_prefix_postfix_hier

Connects instance terminals and/or design ports with prefix and suffix for the complete hierarchy starting from specified design

Declaration

```
connect_by_prefix_postfix_hier <design-name> <prefix> <suffix>
```

Arguments

- `design-name`: Name of the design for which the rule needs to be run
- `prefix`: Prefix string that needs to be present in port names to be considered for connection

- **suffix:** Suffix string that needs to be present in port names to be considered for connection

Example

```
connect_by_prefix_postfix_hier top al_ _clk
```

tie_high_unconn

Ties high all unconnected instance input terminals or design output ports of specified design

Declaration

```
tie_high_unconn <design-name> [<port-name-list>]
```

Arguments

- **design-name:** Name of the design for which the rule needs to be run
- **port-name-list (Optional):** If given, only port names present in the list are considered for tie-off. If no port names are specified, the API considers all possible ports.

Example

This example shows the use of tie_high_unconn. The following table shows the original RTL and the RTL generated.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (p1, q1); input [7:0] p1; output [7:0] q1; wire [3:0] mid2_q2; wire [7:0] mid2_q3; wire [3:0] UNINTENTIONAL_OPEN_mid 2_q2_4; wire [3:0] UNINTENTIONAL_OPEN_mid 1_p2_4; mid mid1 (.p1(), .p2({UNINTENTIONAL_OPE N_mid1_p2_4, mid2_q2}), .p3(mid2_q3), .p4(), .q1(), .q2(), .q3(), .q4()); mid mid2 (.p1(), .p2(), .p3(), .p4(), .q1(), .q2({UNINTENTIONAL_OPE N_mid2_q2_4, mid2_q2}), .q3(mid2_q3), .q4()); endmodule </pre>	<pre> tie_high_unconn top </pre>	<pre> module top (p1, q1); input [7:0] p1; output [7:0] q1; wire [3:0] mid2_q2; wire [7:0] mid2_q3; wire [3:0] UNINTENTIONAL_OPEN_mid2_q2_4; mid mid1 (.p1({8'hFF}), .p2({4'hFF, mid2_q2}), .p3(mid2_q3), .p4({8'hFF}), .q1(), .q2(), .q3(), .q4()); mid mid2 (.p1({8'hFF}), .p2({8'hFF}), .p3({8'hFF}), .p4({8'hFF}), .q1(), .q2({UNINTENTIONAL_OPEN_mid2_ q2_4, mid2_q2}), .q3(mid2_q3), .q4()); endmodule </pre>

tie_high_unconn_hier

Ties high all unconnected instance input terminals or design output ports of complete design hierarchy starting from specified design

Declaration

```
tie_high_unconn_hier <design-name> [<port-name-list>]
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- port-name-list (Optional): If given, only port names present in the list are considered for tie-off. If no port names are specified, the API considers all possible ports.

Example

```
tie_high_unconn_hier top  
tie_high_unconn_hier top {p1 p4}
```

tie_low_unconn

Ties high all unconnected instance input terminals or design output ports of specified design

Declaration

```
tie_low_unconn <design-name> [<port-name-list>]
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- port-name-list (Optional): If given, only port names present in the list are considered for tie-off. If no port names are specified, the API considers all possible ports.

Example

```
tie_low_unconn top  
tie_low_unconn top {p1 p4}
```

tie_low_unconn_hier

Ties high all unconnected instance input terminals or design output ports of complete design hierarchy starting from specified design

Declaration

```
tie_low_unconn_hier <design-name> [<port-name-list>]
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- port-name-list (Optional): If given, only port names present in the list are considered for tie-off. If no port names are specified, the API considers all possible ports.

Example

```
tie_low_unconn_hier top  
tie_low_unconn_hier top {p1 p4}
```

export_unconn_hier

Bottom-up exports all unconnected terminals or ports present in the entire design hierarchy to the top-level design

Declaration

```
export_unconn_hier
```

Arguments

None

Example

This example shows the use of `export_unconn_hier`. The following table shows the original RTL and the RTL generated.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (a1, b3, a4, b1, a3, b4); input [7:0] a1; input [7:0] b3; input [7:0] a4; output [7:0] b1; output [7:0] a3; output [7:0] b4; mid mid1 (.a1(), .b2(), .b1(), .a2()); assign b4 = a4; endmodule module mid (a1, b2, b1, a2); input [7:0] a1; input [7:0] b2; output [7:0] b1; output [7:0] a2; low low1 (.a1(), .a2(), .a3(), .a4(), .b1(), .b2(), .b3(), .b4()); endmodule module low (a1, a2, a3, a4, b1, b2, b3, b4); input [7:0] a1; input [7:0] a2; input [7:0] a3; input [7:0] a4; output [7:0] b1; output [7:0] b2; output [7:0] b3; output [7:0] b4; endmodule </pre>	<pre> export_unconn_hier </pre>	<pre> module top (a1, b3, a4, b1, a3, b4); input [7:0] a1; input [7:0] b3; input [7:0] a4; output [7:0] b1; output [7:0] a3; output [7:0] b4; input [7:0] a2_0; input [7:0] a3_0; output [7:0] b2_0; output [7:0] b3_0; output [7:0] b4_0; mid mid1 (.a1(a1), .b2(b2), .b1(b1), .a2(a2), .a2_0(a2_0), .a3(a3_0), .a4(a4), .b2_0(b2_0), .b3(b3_0), .b4(b4_0)); assign b4 = a4; endmodule module mid (a1, b2, b1, a2, a2_0, a3, a4, b2_0, b3, b4); input [7:0] a1; input [7:0] b2; output [7:0] b1; output [7:0] a2; input [7:0] a2_0; input [7:0] a3; input [7:0] a4; output [7:0] b2_0; output [7:0] b3; output [7:0] b4; low low1 (.a1(a1), .a2(a2_0), .a3(a3), .a4(a4), .b1(b1), .b2(b2_0), .b3(b3), .b4(b4)); endmodule ... </pre>

export_unconn_hier_inputs

Bottom-up exports all unconnected input terminals or ports present in the entire design hierarchy to the top-level design

Declaration

```
export_unconn_hier_inputs
```

Arguments

None

Example

```
export_unconn_hier_inputs
```

export_unconn_hier_outputs

Bottom-up exports all unconnected output terminals or ports present in the entire design hierarchy to the top-level design

Declaration

```
export_unconn_hier_outputs
```

Arguments

None

Example

```
export_unconn_hier_outputs
```

export_unconn_hier_matching_ports

Bottom-up exports all unconnected terminals or ports present in the entire design hierarchy to the top-level design for which matching port exists at the top-level design

Declaration

```
export_unconn_hier_matching_ports
```

Arguments

None

Example

This example shows the use of `export_unconn_hier_matching_ports`. The following table shows the original RTL and the RTL generated.

Original RTL	Tcl Commands Run	Generated RTL
<pre> module top (a1, b3, a4, b1, a3, b4); input [7:0] a1; input [7:0] b3; input [7:0] a4; output [7:0] b1; output [7:0] a3; output [7:0] b4; mid mid1 (.a1(), .b2(), .b1(), .a2()); assign b4 = a4; endmodule module mid (a1, b2, b1, a2); input [7:0] a1; input [7:0] b2; output [7:0] b1; output [7:0] a2; low low1 (.a1(), .a2(), .a3(), .a4(), .b1(), .b2(), .b3(), .b4()); endmodule module low (a1, a2 , a3, a4, b1, b2, b3, b4); input [7:0] a1; input [7:0] a2; input [7:0] a3; input [7:0] a4; output [7:0] b1; output [7:0] b2; output [7:0] b3; output [7:0] b4; endmodule </pre>	<pre> export_unconn_hier_matchi ng_ports </pre>	<pre> module top (a1, b3, a4, b1, a3, b4); input [7:0] a1; input [7:0] b3; input [7:0] a4; output [7:0] b1; output [7:0] a3; output [7:0] b4; output [7:0] b4_0; mid mid1 (.a1(a1), .b2(), .b1(b1), .a2(), .a4(a4), .b4(b4_0)); assign b4 = a4; endmodule module mid (a1, b2, b1, a2, a4, b4); input [7:0] a1; input [7:0] b2; output [7:0] b1; output [7:0] a2; input [7:0] a4; output [7:0] b4; low low1 (.a1(a1), .a2(), .a3(), .a4(a4), .b1(b1), .b2(), .b3(), .b4(b4)); endmodule module low (a1, a2, a3, a4, b1, b2, b3, b4); input [7:0] a1; input [7:0] a2; input [7:0] a3; input [7:0] a4; output [7:0] b1; output [7:0] b2; output [7:0] b3; output [7:0] b4; endmodule </pre>

get_design_by_name

Returns design object with specified name from current hierarchy

Declaration

```
get_design_by_name <design-name>
```

Arguments

- design-name: Name of the design object which needs to be returned

Example

```
get_design_by_name top
```

get_instance_by_name

Returns instance object with specified hierarchical name present in the design

Declaration

```
get_instance_by_name <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_instance_by_name top mid::low
```

get_port_by_name

Returns port object with specified name present in the design

Declaration

```
get_port_by_name <design-name> <port-name>
```

Arguments

- design-name: Name of the design for which port object needs to be returned
- port-name: Name of the port which needs to be returned

Example

```
get_port_by_name mid a1
```

get_terminal_by_name

Returns terminal object with specified name of the specified instance present in the design

Declaration

```
get_terminal_by_name <design-name> <instance-hier-name> <terminal-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.
- terminal-name: Name of the terminal which needs to be returned

Example

```
get_terminal_by_name top mid:low p1
```

get_all_unconnected_ports

Returns list of all unconnected ports in the design

Declaration

```
get_all_unconnected_ports <design-name>
```

Arguments

- design-name: Name of the design for which unconnected ports list needs to be returned

Example

```
get_all_unconnected_ports mid
```

Note:

Tie-off ports are considered as connected.

get_all_unconnected_ports_hier

Returns list of all unconnected ports present in the complete hierarchy starting from specified design

Declaration

```
get_all_unconnected_ports_hier <design-name>
```

Arguments

- design-name: Name of the design for which unconnected ports list needs to be returned

Example

```
get_all_unconnected_ports_hier mid
```

Note:

Tie-off ports are considered as connected.

get_all_unconnected_terminals

Returns list of all unconnected terminals in the specified instance in the design

Declaration

```
get_all_unconnected_terminals <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which unconnected terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_unconnected_terminals top mid::low
```

Note:

Tie-off terminals are considered as connected.

get_all_unconnected_terminals_hier

Returns list of all unconnected terminals in the complete hierarchy starting from the specified instance present in the design

Declaration

```
get_all_unconnected_terminals_hier <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the design for which the rule needs to be run
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which unconnected terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_unconnected_terminals_hier top mid::low
```

Note:

Tie-off terminals are considered as connected.

get_all_connected_ports

Returns list of all connected ports in the specified design

Declaration

```
get_all_connected_ports <design-name>
```

Arguments

- design-name: Name of the design for which the connected ports list needs to be returned

Example

```
get_all_connected_ports mid
```

Note:

Tie-off ports are considered as connected.

get_all_connected_ports_hier

Returns list of all connected ports present in the complete hierarchy starting from the specified design

Declaration

```
get_all_connected_ports_hier <design-name>
```

Arguments

- design-name: Name of the design for which the connected ports list needs to be returned

Example

```
get_all_connected_ports_hier mid
```

Note:

Tie-off ports are considered as connected.

get_all_connected_terminals

Returns list of all connected terminals in the specified instance in the design

Declaration

```
get_all_connected_terminals <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name

- **instance-hier-name:** Hierarchical name of the instance starting from specified design name for which connected terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_connected_terminals top mid::low
```

Note:

Tie-off terminals are considered as connected.

get_all_connected_terminals_hier

Returns list of all connected terminals in the complete hierarchy starting from the specified instance present in the design

Declaration

```
get_all_connected_terminals_hier <design-name> <instance-hier-name>
```

Arguments

- **design-name:** Name of the starting design for instance hierarchical name
- **instance-hier-name:** Hierarchical name of the instance starting from specified design name for which connected terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_connected_terminals_hier top mid::low
```

Note:

Tie-off terminals are considered as connected.

get_all_tied_ports

Returns list of all tied ports in the specified design

Declaration

```
get_all_tied_ports <design-name>
```

Arguments

- design-name: Name of the design for which tied ports list needs to be run

Example

```
get_all_tied_ports mid
```

get_all_tied_ports_hier

Returns list of all tied ports present in the complete hierarchy starting from specified design

Declaration

```
get_all_tied_ports_hier <design-name>
```

Arguments

- design-name: Name of the design for which tied ports list needs to be run

Example

```
get_all_tied_ports_hier mid
```

get_all_tied_terminals

Returns list of all tied terminals of specified instance in the design

Declaration

```
get_all_tied_terminals <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which tied terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_tied_terminals top mid::low
```

get_all_tied_terminals_hier

Returns list of all tied terminals in the complete hierarchy starting from the specified instance present in the design

Declaration

```
get_all_tied_terminals_hier <design-name> <instance-hier-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which tied terminal list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.

Example

```
get_all_tied_terminals_hier top mid::low
```

get_port_connections

Returns list of adhoc connections present on the specified port of the design

Declaration

```
get_port_connections <design-name> <port-name>
```

Arguments

- design-name: Name of the design for which connection list of the specified port needs to be returned
- port-name: Name of the port for which connections list needs to be returned

Example

```
get_port_connections mid a1
```

Note:

Tie-off connections are not included in this list.

get_terminal_connections

Returns list of adhoc connections present on the terminal of the specified instance present in the design

Declaration

```
get_terminal_connections <design-name> <instance-hier-name>  
<terminal-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which terminal connection list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.
- terminal-name: Name of the terminal of the instance for which connection list needs to be returned

Example

```
get_terminal_connections top low::mid p1
```

Note:

Tie-off connections are not included in this list.

get_port_tieoff_connections

Returns list of tie-off connections present on the specified port in the design

Declaration

```
get_port_tieoff_connections <design-name> <port-name>
```

Arguments

- design-name: Name of the design for which connection list of the specified port needs to be returned
- port-name: Name of the port for which connections list needs to be returned

Example

```
get_port_tieoff_connections mid a1
```

get_terminal_tieoff_connections

Returns list of tie-off connections present on the terminal of the specified instance present in the specified design

Declaration

```
get_terminal_tieoff_connections <design-name> <instance-hier-name>  
<terminal-name>
```

Arguments

- design-name: Name of the starting design for instance hierarchical name
- instance-hier-name: Hierarchical name of the instance starting from specified design name for which terminal connection list needs to be returned. Hierarchical instance name should not include starting parent design name. For example, in design Top->Mid->Low, the hierarchical name of Low is Mid::Low.
- terminal-name: Name of the terminal of the instance for which connection list needs to be returned

Example

```
get_terminal_tieoff_connections top mid::low p1
```

GenSys Tcl Methods

GenSys Tcl Methods are used to access the objects in the GenSys database.

do_elaborate

Elaborates the instance parameters or design connections

Applies to

instance, design

Arguments

None

Description

Elaborates the design connections for a design or elaborates the instance parameters for an instance. After elaboration, instance parameter values would be available.

Returns

None

Example

```
do_elaborate
```

do_unelaborate

Unelaborates the instance parameters or design connections

Applies to

instance, design

Arguments

None

Description

Unelaborates the design connections for a design or unelaborates the instance parameters for an instance. After un-elaboration, master parameter values would be available on an instance.

Returns

None

Example

```
do_unelaborate
```

get_actdir

Returns the actual logical port direction

Applies to

lport

Arguments

None

Description

Returns the actual logical port direction. If the port direction is not set then this method returns undef.

Returns

Actual logical port direction

Example

```
foreach lport [get_lports $interface] {  
    puts [get_actdir $lport]
```

get_active_object

Returns the active object

Applies to

root

Arguments

Type name of the object. Type name can have any of the values such as design, component, register, bitfield, instance, interfacedef, typetree, interface, bitenum, port, reggroup, connection, table, subtable, partitiontable, partitiontablehierarchy, activefile, activelang, or rtlcomponent.

Returns

Active object of the given type

Example

```
set root [get_root]
set design [get_active_object $root -type design]
```

get_aconnects

Returns the list of adhoc connections under the design/component instance/scalar terminal/scalar port

Applies to

Design, instance, scalar terminal, scalar port, vector terminal, vector port

Arguments

None

Returns

Returns the list of adhoc connections under the design/component instance/scalar terminal/scalar port vector terminal/ vector port

Example

```
foreach conn [get_aconnects $instance] {
    set first [get_first $conn]
    set first [get_instance $first]
    set second [get_second $conn]
    set sinst [get_instance $second]
```

get_align

Returns the align value of the connection, port

Applies to

aconnect, iconnect, port

Arguments

None

Returns

Align value of the connection and the port

Example

```
foreach port [get_ports $component]
  puts [get_align $port]
```

get_attributes

Returns the list of attributes or the named attribute under the object

Applies to

Most objects

Arguments (optional)

-name <attribute-name> (e.g. -name foo)

Returns

List of attributes for the object or the named attribute

Example

```
foreach att [get_attributes $component] {
  puts [get_name $att]
  puts [get_value $att]
  ...OR...
set att [get_attributes $component -name "foo"]
```

```
puts [get_name $att]
puts [get_value $att]
```

get_autofillschema

Returns the autofill schema for the a columnschema

Applies to

Columnschema

Arguments

None

Returns

The autofill schema for the columnschema

Example

```
foreach cs [get_columnschema $tableschema] {
    set as [get_autofillschema $cs]
```

get_backref

Returns the backref string for the connection

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

Backref string for the connection

Example

```
foreach conn [get_iconnects $design] {
```

```
puts [get_backref $conn]
```

get_category

Returns the attribute category

Applies to

attribute

Arguments

None

Returns

Attribute category

Example

```
foreach att [get_attributes $component] {  
    puts [get_category $att]
```

get_cell

Returns the cell of the specified row of a table

Applies to

table

Arguments

None

Example

```
for {set j 0} {$j < 2} {incr j} {  
    puts [get_data [get_cell $table "Name" $j] ]  
    puts " - "  
    puts [get_data [get_cell $table "LSB" $j] ]  
    puts " - "  
    puts [get_data [get_cell $table "MSB" $j] ]  
    puts " - "  
}
```

get_cellenums

Returns a list of enum values for column schema when the column type is ENUM_TYPE

Applies to

columnschema

Arguments

None

Example

```
foreach cs [get_columnschema $tableschemal] {  
    foreach ce [get_cellenums $cs] {  
        puts $ce  
    }  
}
```

get_celltype

Returns cell type of column schema

Applies to

columnschema

Arguments

None

Returns

Cell type of column schema (specified as INT_TYPE, BOOL_TYPE, STRING_TYPE, ENUM_TYPE, or TABLE_TYPE.)

Example

```
foreach cs [get_columnschema $tableschemal] {  
    puts [get_celltype $cs]  
}
```

get_children

Returns the list of child TypeTrees for the parameter

Applies to

parameter

Arguments

None

Returns

List of child TypeTrees of the parameter

Example

```
foreach child [get_chilren $parameter] {  
    puts [get_name $child]  
    puts [get_value $child]  
}
```

get_clockrate

Returns the clockrate field of the port

Applies to

port

Arguments

None

Return type

Returns the clockrate field data of the port

Example

```
foreach port [get_ports $design] {  
    puts [get_clockrate $port]  
}
```

get_colnames

Returns list of column names in a table

Applies to

table

Arguments

None

Return type

Returns list of column names in a table

Example

```
foreach table [get_tables $design] {  
    foreach colname [get_colnames $table] {  
        puts $colname  
    }  
}
```

get_columndata

Returns list of cells in the given column name

Applies to

table

Arguments

-column <column-name>

(e.g. -column foo)

Return type

Returns list of cells in the given column name

Example

```
foreach cell [get_columndata $table -column "X"] {  
    puts [get_data $cell]  
}
```

get_columnschema

Returns the column schema for the table schema

Applies to

root, tableschema

Arguments

(For root) full hierarchical name of the column;

(For tableschema) column name and the table pointer

Returns

(For root) Returns the columnschema pointer, if found; Otherwise, returns NULL. (Here, the column schema is searched by the full hierarchical column name)

(For tableschema) Returns the columnschema pointer; Otherwise, returns NULL

Example

```
set cs [get_columnschema $tableschema -name "C1"]
```

get_columnschemas

Returns a list of column schemas for the table schema

Applies to

tableschema

Arguments

None

Example

```
foreach cs [get_columnschemas $tableschema] {
```

get_comcomponent

Returns the COM component/design pointer for the RTL component

Applies to

rtlcomponent

Arguments

None

Returns

Returns the COM component/design pointer for the RTL component. In case of partitioning, if RTL component does not have any corresponding COM component, its parent COM component is returned.

Example

```
puts[get_output_dir]
```

get_command

Returns the Tcl command executed to create the connection

Applies to

connect, lconnect, iconnect, tconnect

Arguments

None

Returns

The Tcl command, which was executed to create this connection

Example

```
foreach conn [get_iconnects $design] {  
    puts [get_command $conn]
```

get_components

Returns a list of all currently open component objects, or the named component object if a name is specified

Applied to

Root object

Note:

If the root object is not provided, it will figure out the root on its own

Arguments (optional)

`-name <name-of-object>`

(For example: `-name component1`)

Returns

A list of component objects, or a single component object. Returns an empty string if no components are open or a component object of the specified name is not open.

Example

With root being provided:

```
foreach comp [get_components $root] {  
...OR...  
set comp [get_components $root -name "abc"]
```

Without root being provided:

```
foreach comp [get_components ] {  
...OR...  
set comp [get_components -name "abc"]
```

get_controlclock

Returns the control clock for the port

Applies to

port

Arguments

None

Example

```
foreach port [get_ports $component] {  
    puts [get_controlclock $port]
```

get_controlreset

Returns the control reset port pointer of the port

Applies to

port

Arguments

None

Example

```
foreach port [get_ports $component] {  
    puts [get_controlreset $port]
```

get_creationdate

Returns the creation date from the data source of the object

Applies to

Most objects

Arguments

None

Returns

Returns the creation date from the data source of the object

Example

```
foreach port [get_ports $component] {  
    puts [get_controlreset $port]
```

get_data

Returns string data in the cell

Applies to

cell

Arguments

None

Returns

Returns string data in the cell

Example

```
foreach cell [get_columndata $table -column "X"] {  
    puts [get_data $cell]
```

get_datasource

Returns the data source of the object

Applies to

Most objects

Arguments

None

Returns

Returns the data source of the object

Example

```
set ds [get_datasource $obj]  
puts [get_person $ds]  
puts [get_method $ds]  
puts [get_datetime $ds]
```

get_datetime

Returns the change date and time for the data source

Applies to

datasource

Arguments

None

Returns

Returns the change date and time for the data source

Example

```
puts [get_person $ds]  
puts [get_method $ds]  
puts [get_datetime $ds]
```

get_default

Returns the default value of the port/logical port/interface port or autofill schema

Applies to

port, lport

Arguments

None

Returns

Default value of the port/logical port/interface port

Example

```
foreach port [get_ports $design] {  
    puts [get_default $port]
```

get_deltadelay

Returns the deltdelay value for a connection

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

String value that is the deltadelay value for a connection

Example

```
foreach conn [get_aconnects $design] {  
    puts [get_deltadelay $conn]
```

get_dependcolumns

Returns a list of dependent columns names on the column schema

Applies to

columnschema

Arguments

None

Example

```
foreach cs [get_columnschema $tableschema] {  
    foreach dc [get_dependcolumns $cs]
```

get_description

Returns the description of the specified object

Applies to

design, component, interfacedef, interface, instance, interface, port, bitenum

Arguments

None

Returns

Description for the specified object

Example

```
foreach port [get_ports $component] {  
    puts [get_description $port]
```

get_designs

Returns a list of all currently open design objects or the named design object

Applies to

Root object

Note:

If Root object is not provided, it identifies the root on its own.

Arguments

-name <name-of object>

(For example: -name design1)

Returns

A list of design objects or a single design object. Returns an empty string if no designs are open or a design object of the specified name is not open.

Example

With root being provided:

```
foreach des [get_designs $root] {  
...OR...  
set des [get_designs $root -name "abc"]
```

Without root being provided:

```
foreach des [get_designs] {  
...OR...  
set des [get_designs -name "abc"]
```

get_design_objects

Returns a list of all currently open design objects, or the named design object if a name is specified

Applied to

Root object

Arguments (optional)

-name <name-of object>

(eg. -name design1)

Returns

A list of design objects, or a single design object. Returns an empty string if no designs are open or a design object of the specified name is not open.

Example

```
foreach design [get_design_objects $root] {  
...OR...  
set design [get_design_objects $root -name "foo"]
```

get_dir

Returns the direction of the port/logical port/interface port

Applies to

port, lport

Arguments

None

Returns

One of the following values:

IN

OUT

INOUT

Example

```
foreach ports [get_ports $design] {  
    puts [get_dir $port]
```

get_end

Returns the end address of the Address Interface

Applies to

addressif

Arguments

None

Returns

End Address of the Address Interface

Example

```
foreach addif [get_addressifs $design] {  
    put [get_end $addif]
```

get_endian

Returns the endian type for the memory

Applies to

memory

Arguments

None

Returns

BIG for Big Endian type memory or SMALL for Little Endian type memory

Example

```
foreach memory [get_memories $component] {  
    put [get_endian $memory]
```

get_enumchoices

Returns list of items in the given cell of type ENUM

Applies to

cell

Arguments

None

Returns

Returns list of items in the given cell of type ENUM

Example

```
foreach cell [get_columndata $table -column "X"] {  
    foreach ec [get_enumchoices $cell] {  
        puts $ec
```

get_expr

Returns the live-formula expression string of the cell if the cell contains live-formula, else returns ""

Applies to

cell

Arguments

None

Returns

Live-formula expression string in the cell

Example

```
foreach cell [get_columndata $table -column "X"] {  
    puts [get_expr $cell]
```

get_file

Returns the filename in which the object is defined

Applies to

tableschema, columnschema

Arguments

None

Example

```
puts [get_file $tableschema]
```

get_first

Returns the driver end of a connection

Applies to

aconnect, iconnect, tconnect

Arguments

None

Returns

Driver end of a connection

Note:

The [get_second](#) method returns the sink end of the connection.

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]  
    set first [get_instance $first]  
    set second [get_second $conn]  
    set sinst [get_instance $second]
```

get_frozen

Returns the Data Frozen value for data source

Applies to

datasource

Arguments

None

Returns

Data frozen value for data source

Example

```
set ds [get_datasource $obj]  
puts [get_frozen $ds]
```

get_functiontype

Returns the function type of the bitfield

Applies to

bitfield

Arguments

None

Returns

One of the following values:

ADDRES S	COMMAN D	CONSTAN T	CONTRO L	COUNTE R
DATA	MEM	RESERVE D	STATUS	

Example

```
foreach bitf [get_bitfields $register]
  puts [get_functiontype $bitf]
```

get_hdl_type

Returns the HDL type of the port

Applies to

port

Arguments

None

Returns

HDL type of the port

Example

```
foreach port [get_ports $component]
  puts [get_hdl_type_lsb $port]
```

get_hdl_type_lsb

Returns the LSB of the HDL type

Applies to

port

Arguments

None

Returns

LSB of the HDL type

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_lsb $port]
```

get_hdl_type_msb

Returns the MSB of the HDL type

Applies to

port

Arguments

None

Returns

MSB of the HDL type

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_msb $port]
```

get_hdl_type_language

Returns the language of the HDL type

Applies to

port

Arguments

None

Returns

Language of the HDL type

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_language $port]
```

get_hdl_type_library

Returns the library of the HDL type

Applies to

port

Arguments

None

Returns

Library of the HDL type

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_library $port]
```

get_hdl_type_package

Returns the package of the HDL type

Applies to

port

Arguments

None

Returns

Package of the HDL type

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_package $port] {
```

get_help

Returns the help message for the object

Applies to

attribute, tableschema, columnschema

Arguments

None

Returns

Help message associated with the object

Example

```
foreach port [get_ports $component]
    puts [get_hdl_type_package $port] {
```

get_hiddencolumns

Returns a list of hidden columns for table schema

Applies to

tableschema

Arguments

None

Example

```
foreach hcols [get_hiddencolumns $tableschema] {  
    puts [get_help $att]
```

get_hiername

Returns the hierarchical name of the object

Applies to

tableschema, columnschema

Arguments

None

Example

```
puts [get_hiername $columnschema]
```

get_hiernames

Returns the list of hierarchical path names (from the top design) for the instance object (the path names will be delimited by "::")

Applies to

instance

Arguments

None

Returns

List of hierarchical path names

Example

```
foreach avsc_inst_name [get_hiernames $ainst] { ... }
```

get_iconnects

Returns the list of interface connections for the component instance

Applies to

instance

Arguments

None

Returns

List of interface connections for the component instance. Use the “first” and “second” methods to get the 2 ends of each connection in the list

Example

```
foreach conn [get_iconnects $component] {
```

get_ifinstance

Returns the interface instance for the interface connection

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

Interface Instance of the interface connection

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]  
    set finst [get_instance $first]  
    set fifinst [get_ifinstance $first]
```

get_ifinstances

Returns the list of interface instances under the component instance

Applies to

instance

Arguments

None

Returns

List of interface instances for the component instance

Example

```
foreach ifi [get_ifinstances $instance] {  
    set interface [get_interface $ifi]  
    set interfacedef [get_interfacedef $interface]  
    puts [get_name interfacedef]
```

get_interface

Returns the interface master for the interface instance

Applies to

ifinstance

Arguments

None

Returns

Interface master of the interface instance

Example

```
foreach ifi [get_ifinstances $instance] {  
    set interface [get_interface $ifi]  
    set interfacedef [get_interfacedef $interface]  
    puts [get_name interfacedef]
```

get_interfaceGroup

Returns interface group for the port

Applies to

port

Arguments

None

Returns

Interface group for the port

Example

```
foreach port [get_ports $component] {  
    puts [get_interfaceGroup $port]
```

get_instance

Returns the instance for the connection

Applies to

aconnect_1, aconnect_2, iconnect_1

Arguments

None

Returns

Component instance of the connection

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]
```

get_instances

Returns the list of instances or the named instance under the design

Applies to

design, component

Arguments (optional)

-name <instance-name>

(For example: -name abc)

Note:

If a design is not provided, it would apply on the active design being present.

Returns

List of instances for the design/component or the named instance of the design/component

Example

With design being provided:

```
foreach inst [get_instances $design] {  
...OR...  
set inst [get_instances $design -name "abc"]
```

Without design being provided (This would work on the active design):

```
foreach inst [get_instances] {  
  ...OR...  
  set inst [get_instances -name "abc"]  
}
```

get_intdata

Returns int data in the cell

Applies to

cell

Arguments

None

Returns

Returns int data in the cell

Example

```
foreach cell [get_columndata $table -column "X"] {  
  puts [get_intdata $cell]  
}
```

get_interfacedef

Returns the parent interface definition for the interface

Applies to

interface

Arguments

None

Returns

Parent interface definition for the specified interface

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]  
    set ifil [get_ifinstance $first]  
    set intf1 [get_interface $ifil]  
    set ifdef1 [get_interfacedef $intf1]
```

get_interfacedefs

Returns a list of all currently open interfacedef objects, or the named interfacedef object if a name is specified

Applied to

Root object

Arguments (optional)

-name <name-of object>

(e.g. -name AXI)

Returns

A list of interfacedef objects, or a single interfacedef object. Returns an empty string if no interfacedefs are open or a interfacedef object of the specified name is not open.

Example

```
foreach ifdef [get_interfacedefs $root] {  
    ...OR...  
    set ifdef [get_interfacedefs $root -name "foo"]
```

get_interfaces

Returns the list of interfaces or the named interface under the object

Applies to

design, component

Arguments (optional)

-name <interface-name>

(e.g. `-name foo`)

Returns

List of interfaces for the design/component or the named interface of the design/component

Example

```
foreach intf [get_interfaces $component] {  
    ...OR...  
    set intf [get_interfaces $component -name "foo"]  
}
```

get_isautofill

Checks whether the table schema contains Autofill

Applies to

tableschema

Arguments

integer value

Returns

1 if the table schema contains Autofill. Otherwise, returns 0.

Example

```
puts [get_isautofill $tableschema]
```

get_isbasetable

Returns 1 if table schema is a base table

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is a base table. Otherwise, returns 0

Example

```
puts [get_isbasetable $tableschema]
```

get_isdesign

Returns 1 if the component is a design

Applies to

component

Arguments

None

Returns

1 if the component is a design. Otherwise, returns 0.

Example

```
puts [get_isdesign $component]
```

get_is_elab_generated

Returns 1 if the connection is generated during elaboration

Applies to

aconnect, tconnect

Arguments

None

Returns

1 if the adhoc connection is generated during elaboration. Otherwise, returns 0.

Example

```
foreach conn [get_aconnects $instance]
  puts [get_is_elab_generated $conn]
```

get_iseval

Returns 1 if the column schema is evaluable

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is specified as an evaluable column. Otherwise, returns 0.

Example

```
foreach cs [get_columnschema $tableschema] {
  puts [get_iseval $cs]
```

get_isevalallowed

Recursively checks if the table schema has any evaluable columns

Applies to

tableschema

Arguments

None

Returns

1 if evaluable columns are present in the table schema hierarchy. Otherwise, returns 0.

Example

```
puts [get_isevalallowed $tableschema]
```

get_isevalcolpresent

Returns 1 if the table schema has evaluable columns

Applies to

tableschema

Arguments

None

Returns

1 if the table schema has evaluable columns. Otherwise, returns 0.

Example

```
puts [get_isevalcolpresent $tableschema]
```

get_isextension

Checks whether the column schema is an extension

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is an extension. Otherwise, returns 0.

Example

```
foreach cs [get_columnschema $tableschema] {
```

```
puts [get_isextension $cs]
```

get_isextensiontable

Returns 1 if the table schema is an extension table

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is an extension table. Otherwise, returns 0.

Example

```
puts [get_isextensiontable $tableschema]
```

get_isflattable

Returns 1 if the table schema is a flat table

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is a flat table. Otherwise, returns 0.

Example

```
puts [get_flattable $tableschema]
```

get_is_exported

Returns 1 if the interface instance is the instance of an exported interface

Applies to

ifinstance

Arguments

None

Returns

1 if the interface instance is an instance of an exported instance. Otherwise returns 0.

Example

```
foreach conn [get_iconnects $instance] {  
    set first [get_first $conn]  
    set ifil [get_ifinstance $first]  
    puts [get_is_exported $ifinstance]
```

get_ishidden

Returns 1 if the object is hidden

Applies to

tableschema, columnschema

Arguments

None

Returns

1 if the table schema is hidden. Otherwise, returns 0.

Example

```
puts [get_ishidden $tableschema]
```

get_isinternal

Returns 1 if the object is internally generated

Applies to

Component, instance

Arguments

None

Returns

1 if the component or instance is internally generated. Otherwise, returns 0.

Example

```
foreach inst [get_instances $design] {  
    puts [isinternal $inst]
```

get_iskey

Returns 1 if the column schema is key column

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is a key column. Otherwise, returns 0.

Example

```
foreach cs [get_columnschema $tableschema] {  
    puts [get_iskey $cs]
```

get_ismirror

Returns 1 if the Interface is a mirror image of the Interface definition

Applies to

interface

Arguments

None

Returns

Returns a string (YES/NO/MONITOR) according to the mirror type of the interface

Example

```
foreach conn [get_iconnects $instance] {  
    set first [get_first $conn]  
    set ifil [get_ifinstance $first]  
    set intf1 [get_interface $ifil]  
    puts [get_ismirror $intf1]
```

get_is_multi_dimension

Return TRUE if it is a multi-dimensional port else FALSE

Applies to

Terminal, Port

Arguments

None

Returns

Return TRUE if it is a multi-dimensional port else FALSE

Example

```
get_is_multi_dimension $port  
get_is_multi_dimension $term
```

get_isopen

Returns 1 if the logical connection is an open connection

Applies to

lconnect

Arguments

None

Returns

Returns 1 if the logical connection is an open connection. Otherwise, returns 0.

Example

get_is_open

Returns the open field of the port

Applies to

port, ifinstance

Arguments

None

Returns

Returns the open field data of the port.

Example

```
foreach port [get_ports $component] {  
    puts [get_is_open $port]
```

get_isreadonly

Returns 1 if the object is read-only

Applies to

tableschema, columnschema

Arguments

None

Returns

Returns 1 if the table schema is read-only. Otherwise, returns 0.

Example

```
foreach cs [get_columnschemas $tableschema] {  
    puts [get_isreadonly $cs]
```

get_is_scalar

Returns 1 if the port is a scalar port

Applies to

port

Arguments

None

Returns

1 if the port is a scalar port. Otherwise returns 0.

Example

```
foreach port [get_ports $component] {  
    puts [get_is_scalar $port]
```

get_istab

Returns 1 if the table schema is a tab

Applies to

tableschema, table

Arguments

None

Returns

Returns 1 if the table schema has been specified as tab. Otherwise, returns 0.

Example

```
puts [get_istab $tableschema]
```

get_istable

Returns 1 if the cell is a subtable of another table

Applies to

cell

Arguments

None

Returns

Returns 1 if the cell is a subtable of another table. Otherwise, returns 0.

Example

```
puts [get_istable $cell]
```

get_istemp

Returns 1 if the logical or tieoff connection is a temporary connection

Applies to

lconnect, tconnect

Arguments

None

Returns

Returns 1 if the logical or tieoff connection is a temporary connection. Otherwise, returns 0.

Example

```
foreach conn [get_tconnects $instance] {  
    puts [get_istemp $conn]
```

get_is_term_or_port

Checks if a connection end is terminal or port

Applies to

Connection end

Arguments

None

Returns

Return TRUE if the connection end is terminal else FALSE if its port

Example

```
set first [get_first $conn]  
set second [get_second $conn]  
set first_term [get_is_term_or_port $first]  
set second_term [get_is_term_or_port $second]
```

get_isvalidate

Returns 1 if Validate Perl function is specified for column schema

Applies to

columnschema

Arguments

None

Returns

1 if validate Perl function has been specified for column schema. Otherwise, returns 0.

Example

```
foreach cs [get_columnschemas $tableschema] {  
    puts [get_isvalidate $cs]
```

get_isvalidcolpresent

Returns 1 if table schema has validation columns

Applies to

tableschema

Arguments

None

Returns

1 if the table schema has validation columns. Otherwise, returns 0.

Example

```
puts [get_isvalidcolpresent $tableschema]
```

get_isvalidateallowed

Recursively checks whether validation columns are present in the table schema

Applies to

tableschema

Arguments

None

Returns

1 if validation columns are present in the table schema. Otherwise, returns 0.

Example

```
puts [get_isvalidateallowed $tableschema]
```

get_key

Returns the name of the key column for the table schema

Applies to

tableschema

Arguments

None

Example

```
puts [get_isvalidateallowed $tableschema]
```

get_keycolumns

Returns the key columns for dynamic columns associated with the column schema

Applies to

columnschema

Arguments

None

Example

```
foreach cs [get_columnschemas $tableschema] {  
    foreach kc [get_keycolumns $cs] {  
        puts $kc  
    }  
}
```

get_language

Returns the language of the generated RTL files

Applies to

partition

Arguments

None

Returns

Language of the RTL files generated for each unique partition set, such as Verilog, VHDL, SV, or SystemC

Example

```
foreach part [get_partitions $design] {  
    puts [get_language $part]  
}
```

get_library

Returns the library name of the versioned item

Applies to

vnlv

Arguments

None

Returns

Library name of the versioned item.

Example

```
set vlnv [get_vnlv $component]
puts [get_library $vlnv]
```

get_lname

Returns the logical name for the port/logical port/interface port

Applies to

port, lport, bitenum

Arguments

None

Returns

Logical name of the port/logical port/interface port

Example

```
set vlnv [get_vnlv $component]
puts [get_library $vlnv]
```

get_line

Returns the line number at which the object is defined

Applies to

tableschema, columnschema

Arguments

None

Example

```
foreach cs [get_columnschema $tableschema] {  
    puts [get_line $cs]
```

get_longlongintdata

Returns a string corresponding to integer-equivalent value stored in a table cell

Applies to

cell

Arguments

None

Example

```
foreach cell [get_columndata $table -column "X"] {  
    puts [get_longlongintdata $cell]
```

get_lports

Returns the list of logical ports of the interface definition or the interface

Applies to

interfacedef, interface

Arguments (optional)

-name <logical-port-name>

(e.g. -name foo)

Returns

List of logical ports of the interface definition or the interface

Example

```
foreach lport [get_lports $interface] {  
    puts [get_lname $lport]  
    ...OR...  
    set lport [get_lports $interface -name "foo"]  
}
```

get_lsb

Returns the LSB value of the port/connection

Applies to

port, lport, aconnect_1, aconnect_2, splice

Arguments

None

Returns

LSB value of the port or connection

Example

```
foreach lport [get_lports $interface] {  
    puts [get_lsb $lport]  
}
```

get_macro_visibility

Returns the macro visibility of the port, instance, parameter, adhoc/tieoff connection, and file

Applies to

port, instance, parameter, aconnect(adhoc connection), tconnect (tie-off connection), file

Arguments

None

Returns

Macro visibility of the port, instance, parameter, adhoc/tieoff connection, and file. If port is under nested ifdefs, macro_visibility is returned as X:Y:Z. You can parse this output.

Example

```
foreach port [get_ports $design] {  
    puts [get_macro_visibility $port]
```

get_master

Returns the master component for the component instance

Applies to

instance

Arguments

None

Returns

Master component of the component instance

Example

```
foreach inst [get_instances $design] {  
    puts [get_master $inst]
```

get_masterrtlcomponent

Returns master RTL component for the RTL instance

Applies to

rtlinstance

Arguments

None

Returns

Master RTL component for the RTL instance

Example

get_maxdatawidth

Returns the maximum data width of memory

Applies to

memory

Arguments

None

Returns

Maximum data width of memory

Example

```
foreach memory [get_memories $component] {  
    puts [get_maxdatawidth $memory]
```

get_maxvalue

Returns maximum value of bit-enum

Applies to

bitenum

Arguments

None

Returns

Maximum value of bit-enum

Example

```
foreach bite [get_bitenums $bitfield] {  
    puts [get_maxvalue $bite]
```

get_memories

Returns a list of memories under the component or a named memory

Applies to

component

Arguments (optional)

-name <memory-name>

(e.g. -name foo)

Returns

List of memories under the component or the named memory

Example

```
foreach memory [get_memories $component] {  
    ...OR...  
set memory [get_memories $component -name "foo"]
```

get_method

Returns the change method for data source

Applies to

datasource

Arguments

None

Returns

Change method for data source

Maximum value of valuerange

Example

```
set ds [get_datasource $obj]
puts [get_person $ds]
puts [get_method $ds]
puts [get_datetime $ds]
```

get_memory

Returns the Memory object

Applies to

address

Arguments

None

Returns

Memory object

Example

get_minvalue

Returns minimum value of bit-enum

Applies to

bitenum

Arguments

None

Returns

Minimum value of bit-enum

Example

```
foreach bite [get_bitenums $bitfield] {  
    puts [get_minvalue $bite]
```

get_morebackref

Returns more backref information

Applies to

connect, lconnect, iconnect, tconnect

Arguments

None

Returns

More backref information, if any, which is added during elaboration phase

Example

```
foreach conn [get_aconnects $instance] {  
    puts [get_morebackref $conn]
```

get_msb

Returns the MSB value of the port/connection

Applies to

port, lport, aconnect_1, aconnect_2, splice

Arguments

None

Returns

MSB value of the port or connection

Example

```
foreach port [get_ports $design] {  
    puts [get_msb $port]
```

get_name

Returns the object name

Applies to

Most objects

Arguments

None

Returns

Name of the object

Example

```
puts [get_name $object]
```

get_ncols

Returns number of columns in the table

Applies to

table

Arguments

None

Returns

Number of columns in the table

Example

```
foreach table [get_tables $design] {  
    puts [get_ncols $table]
```

get_nrows

Returns number of rows in the table

Applies to

table

Arguments

None

Returns

Number of rows in the table

Example

```
foreach table [get_tables $design] {  
    puts [get_rows $table]
```

get_numrows

Returns the number of rows of the table

Description

Returns the number of rows of the table after evaluation. If the table schema contains an EVAL ROW function, the output of the function is evaluated and number of rows of the table is returned.

If the table schema does not contain an EVAL ROW function, -1 is returned.

Applies to

tableschema

Arguments

table pointer (for which the number of rows is to be evaluated)

Example

```
puts[get_output_dir]
```

get_offset

Returns the offset value for the object

Applies to

register, memory, group

Arguments

None

Returns

Offset value of the register/register group/memory

Example

```
foreach reg [get_registers $component] {  
    puts [get_offset $reg]
```

get_optional

Returns the Optional field data for the port, logical port

Applies to

lport, port

Arguments

None

Returns

Optional field data

Example

```
foreach port [get_ports $component] {  
    puts [get_optional $port]
```

get_other_end

Returns list of adhoc connection ends present on other side

Applies to

scalar terminal, scalar port, vector terminal, vector port

Arguments

None

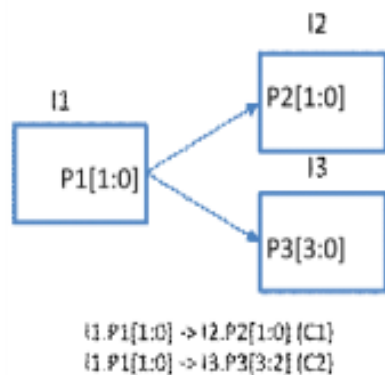
Returns

Return the list of adhoc connection end on the other side of port or terminal

Example

A design has two connections:

- I1.P1[1:0] -> I2.P2[1:0] (C1)
- I1.P1[1:0] -> I3.P3[3:2] (C2)



This is illustrated in the following figure:

In this design:

- \$obj1 represents connection end corresponding to I2.P2[1:0] of connection C1
- \$obj2 represents connection end corresponding to I3.P3[3:2] of connection C2
- \$obj3 represents connection end for I2.P2[1]
- \$obj4 represents connection end for I3.P3[3]

The behavior of the get_other_end API is as follows:

- On vector terminal object I1.P1[1:0], it returns the list of connection ends {\$obj1,\$obj2}
- On scalar bit terminal object I1.P1[1], it returns list of connection ends {\$obj3, \$obj4}

get_parameters

Returns list of parameters or the named parameter

Applies to

design, component, instance

Arguments (optional)

-name <parameter-name>

(e.g. -name foo)

Returns

List of all parameters or the named parameter under design/component/instance.

Note:

For instances, first use the `do_elaborate` method to ensure you will pick up instance-specific parameters

Example

```
do_elaborate
foreach param [get_parameters $instance] {
    puts [get_name $param]
    ...OR...
do_elaborate
set param [get_parameters $instance -name "foo"]
```

get_parent

Returns the parent interface for the interface connection

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

Parent instance of the interface connection

Example

```
foreach conn [get_iconnects $instance] {
    set first [get_first $conn]
    puts [get_parent $first]
```

get_parentrtlcomponent

Returns parent RTL component for the RTL instance

Applies to

rtlinstance

Arguments

None

Returns

Parent RTL component for the RTL instance.

Example

get_person

Returns the change person name for the data source

Applies to

datasource

Arguments

None

Returns

Change person name for data source

Example

```
set ds [get_datasource $obj]
puts [get_person $ds]
puts [get_method $ds]
puts [get_datetime $ds]
```

get_partition

Returns partition name for the component/component instance

Applies to

component, instance

Arguments

None

Returns

Partition name for component/component instance

Example

```
foreach inst [get_instances $design] {  
    puts [get_partition $inst]
```

get_partitions

Returns list of partition objects

Applies to

design

Arguments

None

Returns

List of partition objects

Example

```
foreach parts [get_partitions $design] {  
    puts [get_name $part]
```

get_path

Returns the path of generated RTL files

Applies to

partition

Arguments

None

Returns

Path of RTL files generated for the given partition object

Example

```
foreach parts [get_partitions $design] {  
    puts [get_path $part]
```

get_plane

Returns the plane name for the connection

Applies to

iconnect, aconnect

Arguments

None

Returns

Plane name for the connection

Example

```
foreach conn [get_aconnects $instance] {  
    puts [get_plane $conn]
```

get_port

Returns the port for the terminal

Applies to

terminal

Arguments

None

Returns

Port for the specified terminal

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]  
    set term [get_terminal $first]  
    set_port [get_port $term]
```

get_port_connected

Returns the list of the connected ports or terminals

Applies to

scalar terminal, scalar port, bussed ports

Arguments

Instance name and scalar terminal name, scalar port name, bussed port name

Returns

A list of connected terminals/ports for the given scalar terminal or port.

You can use the APIs, such as [get_name](#), [get_lsb](#), and [get_msb](#) to get more information about the terminals/ports returned.

Example

Consider the following API:

The following API returns the list of terminals/ports (pointers) connected to scalar port P1 of design.

```
get_port_connected P1
```

The above API returns the list of terminals/ports (pointers) connected to scalar port P1(2) of vector port P1 of design.

```
get_port_connected P1(2)
```

The above API returns the list of terminals/ports (pointers) connected to scalar terminal P1 of the instance I1.

```
get_port_connected I1.P1
```

The above API returns the list of terminals/ports (pointers) connected to the scalar terminal P1(1) or vector terminal P1 of instance I1.

```
get_port_connected I1.P1(1)
```

get_portname

Returns the port name of the splice port

Applies to

splice

Arguments

None

Returns

Port name of the splice port

Example

```
foreach conn [get_iconnects $instance] {  
    set first [get_first $conn]
```

```
foreach splice [get_splices $first] {  
    puts [get_portname $splice]
```

get_ports

Returns the list of ports or the named port under the design/component

Applies to

design, component

Arguments (optional)

-name <port-name>

(For example: -name abc).

Note:

If a design is not provided, it applies on the active design being present.

Returns

List of ports for the object or the named port of the object

Example

With design being provided:

```
foreach port [get_ports $design] {  
    puts [get_name $port]  
    ...OR...  
    set port [get_ports $design -name "abc"]
```

Without design being provided (This would work on the active design):

```
foreach port [get_ports ] {  
    puts [get_name $port]  
    ...OR...  
    set port [get_ports -name "abc"]
```

get_powerdomain

Returns the power domain of the port, interface, or instance

Applies to

port, interface, instance

Arguments

None

Returns

Power domain of the port

Example

```
foreach port [get_ports $design] {  
    puts [get_powerdomain $port]
```

get_ptable

Returns parent table pointer

Applies to

table, cell

Arguments

None

Returns

Pointer to the parent table

Example

```
set parenttable [get_ptable $stable]  
puts [get_name $parenttable]
```

get_ptablecolname

Returns the parent column name for the table cell or sub-table

Applies to

table, cell

Arguments

None

Returns

Parent column name for the table cell or sub-table

Example

```
puts [get_ptablecolname $subtable]
```

get_ptablerownum

Returns the parent row number for the table cell or sub-table

Applies to

table, cell

Arguments

None

Returns

Parent row number for the table cell or sub-table

Example

```
puts [get_ptablerownum $cell]
```

get_reason

Returns the change reason for data source

Applies to

datasource

Arguments

None

Returns

Change reason for the data source

Example

```
set ds [get_datasource $obj]
puts [get_person $ds]
puts [get_method $ds]
puts [get_datetime $ds]
puts [get_reason $ds]
```

get_reset

Returns the reset value of the bitfield

Applies to

Register, bitfield

Arguments

None

Returns

Reset value of the register, bitfield

Example

```
foreach bitf [get_bitfields $register] {
    puts [get_reset $bitf]
```

get_resetmask

Returns the reset mask of the object

Applies to

bitfield

Arguments

None

Returns

Reset mask of the register or bitfield

Example

```
foreach bitf [get_bitfields $register] {  
    puts [get_resetmask $bitf]
```

get_rowdata

Returns list of cell in the given row number

Applies to

table

Arguments

Row number

Returns

Returns list of cell in the given row number

Example

```
foreach reg [get_registers $component] {  
    puts [get_reserved $reg]
```

get_rowtype

Returns type of rows for the table schema

Applies to

tableschema

Arguments

None

Returns

Returns type of rows for the table schema (specified as SINGLE or MULTIPLE.) Returns Eval where the schema specifies ROW_TYPE :: perl function() for the table

Example

```
puts [get_rowtype $tableschema]
```

get_rtlfiles

Returns the list of RTL files for the partition

Applies to

partition

Arguments

None

Returns

List of RTL files for the partition object

Example

```
foreach $file [get_rtlfiles $partition] {  
    puts $file
```

get_rtlinstances

Returns the list of RTL instances for the RTL component

Applies to

rtlcomponent

Arguments

None

Returns

List of RTL instances for the RTL component

Example

get_rtlports

Returns list of RTL ports for the RTL component

Applies to

rtlcomponent

Arguments

None

Returns

List of RTL ports for the RTL component

Example

get_second

Returns the sink end of a connection

Applies to

aconnect, iconnect

Arguments

None

Returns

Sink end of a connection

Note:

The [get_first](#) method returns the driver end of the connection.

Returns

Nothing

Example

```
foreach conn [get_aconnects $instance] {  
    set first [get_first $conn]  
    set first [get_instance $first]  
    set second [get_second $conn]  
    set sinst [get_instance $second]
```

get_scalarports

Returns the scalar ports of a vector port

Applies to

port

Arguments

None

Returns

Scalar ports of a vector port

get_scalarterminals

Returns the scalar terminals of a vector terminal

Applies to

terminal

Arguments

None

Returns

Scalar terminals of a vector terminal

get_selfinstances

Returns the list of unique instantiations of a given component or design object

Applies to

Applies to component or design

Arguments

None

Returns

List of unique instances for the design/component

Example

```
foreach ainst [get_selfinstances $avsc_master] { ... }
```

get_shortcode

Returns the short name of the object

Applies to

Most objects

Arguments

None

Returns

Short name of the register block or the register or the port

Example

```
puts [get_shortcode $object]
```

get_size

Returns the size of memory

Applies to

memory

Arguments

None

Returns

Size of the memory

Example

```
foreach memory [get_memories $component] {  
    puts [get_size $memory]
```

get_splices

Returns a list of splice ports

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

List of splice ports of iconnect_1 and iconnect_2

Example

```
foreach conn [get_iconnects $instance] {  
    foreach splice [get_splices $conn]
```

get_start

Returns the start address of the Address Interface

Applies to

addressif

Arguments

None

Returns

Start Address

Example

```
foreach aif [get_addressifs $design] {  
    puts [get_start $aif]
```

get_subtable

Returns subtable pointer

Applies to

table

Arguments

Column name *colname*, Row number *rownumber*

Returns

Subtable pointer

Example

get_table

Returns pointer to the subtable or NULL

Applies to

cell

Arguments

None

Returns

Pointer to the subtable if the cell is a subtable. Otherwise, returns NULL.

Example

```
foreach cell [get_rowdata $table -row 3] {  
    set subtable [get_table $cell]
```

get_tables

Returns the list of tables or a named table under the object

Applies to

design, component, interfacedef, instance

Arguments (optional)

-name <table-name>

(e.g. -name foo)

Returns

List of tables under the object or the named table

Example

```
foreach table [get_tables $design] {  
    ...OR...  
    set table [get_tables $design -name "foo"]
```

get_tableschema

Returns table schema for the root, column schema, and table schema

Applies to

root, columnschema, table

Arguments

None (for columnschema and table schema), full hierarchical table name (for root)

Returns

(For root) Returns tableschema pointer; otherwise returns NULL.

(For columnschema) Returns the corresponding table schema if the columnschema TYPE is specified as TABLE_TYPE; otherwise, returns NULL.

(For table) Returns tableschema pointer.

Example

```
set root [get_root]  
set tschema [get_tableschema $root]
```

get_tconnects

Returns the list of adhoc to tieoff connections on a design/instance

Applies to

design, instance

Arguments

None

Returns

List of tieoff connections on a design or instance. Use the “first” method to get the terminal to which the tieoff is attached for each object in the list.

Example

```
foreach tconn [get_tconnects $instance] {  
    puts [get_value $tconn]
```

get_terminals

Returns the list or named terminal of the component instance or interface instance

Applies to

instance, ifinstance

Arguments (Optional)

-name <terminal-name>

Returns

List of terminals of the component instance or interface instance or the named terminal of the instance/ifinstance

Example

```
foreach inst [get_instances $design] {  
    foreach term [get_terminals $inst] {  
        ...OR...  
        set term [get_terminals $inst -name "p1"]
```

get_tid

Returns the table ID

Applies to

table

Arguments

None

Returns

ID of the table

Example

```
foreach table [get_tables $design] {  
    puts [get_tid $table]
```

get_tieoffs

Returns a list of splice tieoff values

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

List of splice tieoff values of iconnect_1 and iconnect_2

Example

```
foreach conn [get_iconnects $instance] {  
    set first [get_first $conn]  
    foreach tieoff [get_tieoffs $first] {  
        puts $tieoff
```

get_type

Returns the object type

Applies to

component, address, port, lport, interface, attribute, datasource, parameter, reggroup, cell

Arguments

None

Returns

The object type as one of the following:

Object	Return Type values
component	CPU, BUSLOGIC, MEMORY, PERIPHERAL, SUBSYSTEM, OTHERS
address	MEMORY_BLOCK, REGISTER_BLOCK
port, lport	CLK, RST, EVT, DATA, ADDR, CONTROL, TIEOFF, IO_INPUT, IO_OUTPUT, IO_PAD, IO_SELECT, IO_OEN, IO_PULLEN, FUNCTION_IN, TEST_IN, SCANIN, SCANOUT, DFT, CORE
interface	MASTER, SLAVE
attribute	ATTR_STRING, ATTR_INT, ATTR_BOOL, ATTR_ENUM, ATTR_HEX
datasource	STRING
parameter	INT, STRING, BOOL, ENUM, ARRAY, RECORD
reggroup	REGISTER, GROUP
cell	TABLE, INT, BOOL, STRING, ENUM, PERL

Example

```
puts [get_type $object]
```

get_validate

Returns the number of invalid cells

Applies to

table

Description

Validates that table and returns the number of invalid cells

Arguments

None

Example

```
set tname {design.Instances}  
set table [get_table $design -name $tname]  
puts [get_validate $table]
```

get_value

Returns the value of the object

Applies to

parameter, attribute, bitenum, tconnect

Arguments

None

Returns

ValueTree of the parameter or the value of the attribute or bitenum, or the tieoff value for the adhoc to tieoff connection

Example

```
do_evaluate  
foreach param [get_parameters $instance] {  
    puts [get_value $param]}
```

get_vendor

Returns the vendor name of the versioned item

Applies to

vnlv

Arguments

None

Returns

Vendor name of the versioned item

Example

```
set vlnv [get_vnlv $design]
puts [get_vendor $vlnv]
```

get_version

Returns the version

Applies to

vnlv, tableschema

Arguments

None

Returns

Version of the versioned item

Example

```
set vlnv [get_vnlv $design]
puts [get_version $vlnv]
```

get_vnlv

Returns the vnlv of the object

Applies to

component, design, interfacedef

Arguments

None

Returns

Vnlv of the object

Example

```
set vlnv [get_vnlv $design]
puts [get_vendor $vlnv]
```

get_voltage

Returns the voltage of the port, interface, or instance

Applies to

Port, interface, instance

Arguments

None

Returns

Returns the voltage of port, interface, or instance

Example

```
foreach port [get_ports $component]
    puts [get_voltage $port]
```

get_width

Returns the width of the object

Applies to

port, lport, register, blocksize

Arguments

None

Returns

Width of the port/logical port/register

Example

```
foreach port [get_ports $component]
    puts [get_width $port]
```

get_xmlfile

Returns the XML file for the object

Applies to

design, component, interfacedef

Arguments

None

Returns

Name of the XML file

Example

```
puts [get_xmlfile $component]
```

