

PrimeTime Commands

Version D-2009.12, December 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2009 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

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.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Table of Contents

add_distributed_hosts	1
add_to_collection	3
add_variation	6
all_clocks	7
all_connected	8
all_correlations	10
all_fanin	11
all_fanout	13
all_inputs	16
all_instances	18
all_outputs	20
all_registers	22
all_variations	26
append_to_collection	27
cell_of	29
change_histogram	30
change_selection	33
characterize_context	36
check_block_scope	38
check_level_shifter	41
check_noise	42
check_power	45
check_timing	47
collections_and_querying	55

compare_collections	61
compare_interface_timing	63
complete_net_parasitics	69
connect_net	71
connect_power_domain	73
connect_power_net_info	75
connect_supply_net	76
copy_collection	78
cputime	80
create_activity_waveforms	81
create_cell	82
create_clock	87
create_correlation	90
create_distributed_farm	92
create_eco_astro_constraints	94
create_generated_clock	98
create_ilm	104
create_lcd_operating_condition	111
create_net	113
create_operating_conditions	117
create_power_domain	119
create_power_group	122
create_power_net_info	124
create_power_rail_mapping	126
create_power_switch	129
create_qtm_constraint_arc	132
create_qtm_delay_arc	135
create_qtm_drive_type	137
create_qtm_generated_clock	139
create_qtm_load_type	141
create_qtm_model	143
create_qtm_path_type	145
create_qtm_port	148
create_scenario	150
create_si_context	152
create_supply_net	155
create_supply_port	157
create_variation	159

current_design	162
current_instance	164
current_power_rail	167
current_scenario	169
current_session	172
define_design_mode_group	174
define_qtm_attribute	176
define_scaling_lib_group	178
define_user_attribute	180
derive_clocks	183
disconnect_net	185
drive_of	188
estimate_clock_network_power	189
estimate_eco	192
extract_model	199
filter	204
filter_collection	205
find	208
fix_eco_timing	210
foreach_in_collection	216
get_alternative_lib_cells	218
get_attribute	221
get_cells	223
get_clock_network_objects	226
get_clocks	229
get_correlations	231
get_current_power_domain	233
get_current_power_net	234
get_designs	236
get_distributed_variables	239
get_generated_clocks	242
get_ilm_objects	244
get_lib_cells	246
get_lib_pins	249
get_lib_timing_arcs	252
get_libs	255
get_license	258
get_nets	259

get_noise_violation_sources	263
get_object_name	266
get_path_groups.	267
get_pins	269
get_ports.	273
get_power_domains.	276
get_power_group_objects	278
get_power_switches.	279
get_qtm_ports.	281
get_random_numbers	283
get_selection	284
get_si_bottleneck_nets.	287
get_supply_nets	289
get_supply_ports	291
get_switching_activity.	293
get_timing_arcs	300
get_timing_paths	303
get_variation_attribute	312
get_variations.	314
group_path	316
gui_start	320
gui_stop	321
gui_write_window_image	322
identify_interface_logic.	324
index_collection	328
insert_buffer	329
license_users	338
link	339
link_design	340
list_attributes	344
list_delcalc_resources	347
list_designs.	348
list_key_bindings	350
list_libraries.	351
list_licenses	353
load_of	354
load_upf	355
map_design_mode.	357

max_variation	359
mem	360
merge_models	361
merge_saif	364
min_variation	367
print_message_info	368
query_objects	370
read_aocvm	372
read_db	374
read_ddc	376
read_file	378
read_lib	379
read_milkyway	381
read_parasitics	383
read_saif	390
read_sdc	393
read_sdf	398
read_vcd	402
read_verilog	406
read_vhdl	410
remote_execute	412
remove_annotated_check	414
remove_annotated_clock_network_power	417
remove_annotated_delay	418
remove_annotated_parasitics	420
remove_annotated_power	422
remove_annotated_transition	424
remove_aocvm	426
remove_buffer	428
remove_capacitance	431
remove_case_analysis	432
remove_cell	434
remove_clock	437
remove_clock_gating_check	438
remove_clock_groups	440
remove_clock_latency	442
remove_clock_sense	444
remove_clock_transition	446

remove_clock_uncertainty	447
remove_connection_class	450
remove_context	451
remove_coupling_separation	453
remove_current_session	455
remove_data_check	456
remove_delcalc_resource	458
remove_design	459
remove_design_mode	461
remove_disable_clock_gating_check	463
remove_disable_timing	464
remove_distributed_hosts	465
remove_drive_resistance	466
remove_driving_cell	468
remove_fanout_load	470
remove_from_collection	471
remove_generated_clock	473
remove_host_options	474
remove_ideal_latency	478
remove_ideal_network	480
remove_ideal_transition	482
remove_input_delay	484
remove_input_noise	486
remove_lib	487
remove_license	489
remove_max_area	490
remove_max_capacitance	491
remove_max_fanout	492
remove_max_time_borrow	493
remove_max_transition	494
remove_min_capacitance	495
remove_min_pulse_width	496
remove_multi_scenario_design	498
remove_net	499
remove_noise_immunity_curve	501
remove_noise_lib_pin	502
remove_noise_margin	503
remove_operating_conditions	504

remove_output_delay	505
remove_parasitic_corner	507
remove_path_group	508
remove_port_fanout_number	509
remove_power_groups	510
remove_propagated_clock	511
remove_pulse_clock_max_transition	512
remove_pulse_clock_max_width	514
remove_pulse_clock_min_transition	515
remove_pulse_clock_min_width	516
remove_qtm_attribute	517
remove_rail_voltage	519
remove_resistance	520
remove_scenario	521
remove_setup_hold_pessimism_reduction	522
remove_si_aggressor_exclusion	524
remove_si_delay_analysis	526
remove_si_delay_disable_statistical	529
remove_si_noise_analysis	530
remove_si_noise_disable_statistical	533
remove_steady_state_resistance	534
remove_user_attribute	535
remove_user_sensitization	537
remove_variation	539
remove_wire_load_min_block_size	540
remove_wire_load_model	541
remove_wire_load_selection_group	543
rename_cell	544
rename_design	547
rename_net	549
report_activity_waveforms	552
report_alternative_lib_cells	554
report_analysis_coverage	559
report_annotated_check	564
report_annotated_delay	567
report_annotated_parasitics	570
report_annotated_power	575
report_aocvm	577

report_attribute	581
report_bottleneck	583
report_bus	588
report_case_analysis	590
report_cell	593
report_clock	597
report_clock_gate_savings	601
report_clock_gating_check	608
report_clock_timing	610
report_constraint	623
report_context	634
report_crpr	636
report_delay_calculation	643
report_design	655
report_disable_timing	657
report_distributed_hosts	660
report_driver_model	662
report_etm_arc	665
report_exceptions	669
report_global_slack	674
report_hierarchy	677
report_host_usage	678
report_ideal_network	685
report_lib	689
report_lib_groups	695
report_min_pulse_width	697
report_mode	700
report_multi_scenario_design	704
report_name_mapping	707
report_net	708
report_noise	711
report_noise_calculation	714
report_noise_parameters	718
report_noise_violation_sources	719
report_path_group	722
report_port	724
report_power	727
report_power_analysis_options	740

report_power_calculation	741
report_power_derate	747
report_power_domain.	750
report_power_groups	752
report_power_net_info	754
report_power_network	756
report_power_pin_info	757
report_power_rail_mapping	760
report_power_switch	762
report_pulse_clock_max_transition	763
report_pulse_clock_max_width	766
report_pulse_clock_min_transition.	768
report_pulse_clock_min_width	770
report_qtm_model	772
report_reference.	775
report_scale_parasitics.	777
report_scope_data	778
report_si_aggressor_exclusion.	780
report_si_bottleneck.	783
report_si_delay_analysis	787
report_si_double_switching	793
report_si_noise_analysis	795
report_supply_net.	800
report_switching_activity	802
report_timing.	814
report_timing_derate	838
report_transitive_fanin	842
report_transitive_fanout	844
report_units.	846
report_user_sensitization	848
report_variation.	850
report_vcd_hierarchy	860
report_wire_load.	862
reset_design.	864
reset_mode.	865
reset_noise_parameters.	868
reset_path.	869
reset_power_derate	873

reset_scale_parasitics	875
reset_switching_activity	876
reset_timing_derate	880
reset_variation	883
restore_session	885
save_qtm_model	886
save_session	888
scale_parasitics	890
set_active_clocks	892
set_annotated_check	894
set_annotated_clock_network_power	897
set_annotated_delay	900
set_annotated_power	904
set_annotated_transition	906
set_aocvm_coefficient	908
set_case_analysis	910
set_clock_gating_check	912
set_clock_groups	915
set_clock_latency	919
set_clock_sense	923
set_clock_transition	925
set_clock_uncertainty	927
set_connection_class	931
set_context_margin	933
set_coupling_separation	935
set_current_power_domain	937
set_current_power_net	939
set_data_check	942
set_delcalc_resource	945
set_design_top	946
set_disable_clock_gating_check	947
set_disable_timing	948
set_distributed_parameters	952
set_distributed_variables	954
set_domain_supply_net	956
set_dont_touch	958
set_dont_touch_network	960
set_drive	961

set_drive_resistance.	964
set_driving_cell.	966
set_equal	969
set_false_path	971
set_fanout_load	975
set_host_options	976
set_ideal_latency	980
set_ideal_network.	982
set_ideal_transition.	984
set_input_delay	986
set_input_noise	991
set_input_transition	993
set_isolation	995
set_isolation_control.	998
set_lcd_pulse_width_multipliers.	1000
set_level_shifter_strategy.	1002
set_level_shifter_threshold.	1003
set_lib_rail_connection.	1005
set_library_driver_waveform	1006
set_load	1008
set_max_area.	1012
set_max_capacitance.	1013
set_max_delay	1015
set_max_fanout	1020
set_max_time_borrow	1022
set_max_transition	1024
set_min_capacitance	1026
set_min_delay	1028
set_min_library	1033
set_min_pulse_width	1035
set_mode	1037
set_multi_scenario_license_limit	1040
set_multicycle_path	1042
set_noise_derate	1049
set_noise_immunity_curve.	1051
set_noise_lib_pin	1053
set_noise_margin.	1054
set_noise_parameters	1056

set_operating_conditions	1058
set_opposite	1062
set_output_delay	1064
set_parasitic_corner	1069
set_port_fanout_number	1071
set_power_analysis_options	1072
set_power_derate	1077
set_program_options	1081
set_propagated_clock	1084
set_pulse_clock_max_transition	1086
set_pulse_clock_max_width	1088
set_pulse_clock_min_transition	1090
set_pulse_clock_min_width	1092
set_qtm_attribute	1094
set_qtm_global_parameter	1096
set_qtm_port_drive	1098
set_qtm_port_load	1100
set_qtm_technology	1102
set_rail_voltage	1104
set_related_supply_net	1107
set_resistance	1109
set_retention	1111
set_retention_control	1113
set_rtl_to_gate_name	1115
set_scope	1117
set_setup_hold_pessimism_reduction	1119
set_si_aggressor_exclusion	1121
set_si_delay_analysis	1123
set_si_delay_disable_statistical	1126
set_si_noise_analysis	1127
set_si_noise_disable_statistical	1130
set_steady_state_resistance	1131
set_supply_net_probability	1133
set_switching_activity	1135
set_temperature	1141
set_timing_derate	1143
set_units	1149
set_user_attribute	1151

set_user_sensitization	1153
set_variation	1157
set_variation_correlation	1158
set_variation_library	1160
set_variation_quantile	1162
set_voltage	1165
set_wire_load_min_block_size	1167
set_wire_load_mode	1168
set_wire_load_model	1169
set_wire_load_selection_group	1171
size_cell	1173
sizeof_collection	1178
sort_collection	1179
start_gui	1181
start_hosts	1182
start_profile	1187
stop_gui	1188
stop_hosts	1189
stop_profile	1193
sub_variation	1194
swap_cell	1195
transform_exceptions	1199
translate_stamp_model	1203
unset_rtl_to_gate_name	1204
update_noise	1205
update_power	1207
update_scope_data	1210
update_timing	1212
upf_version	1213
variation_correlation	1215
write_activity_waveforms	1217
write_arrival_annotations	1221
write_astro_changes	1223
write_binary_aocvm	1225
write_changes	1227
write_context	1230
write_ilm_netlist	1233
write_ilm_parasitics	1235

write_ilm_script.	1237
write_ilm_sdf.	1239
write_interface_timing.	1241
write_parasitics.	1244
write_physical_annotations	1246
write_profile	1249
write_saif	1250
write_script.	1252
write_sdc	1254
write_sdf.	1258
write_sdf_constraints	1264
write_spice_deck	1268

add_distributed_hosts

Add one or more distributed hosts for distributed jobs.

SYNTAX

```
int add_distributed_hosts
[-32bit]
-farm lsf | grd | generic | now
[-setup_path setup_path]
[-num_of_hosts count]
[-options string]
[-submission_script submission_script]
[hostname]
```

ARGUMENTS

- `[-32bit]`
Specifies that the architecture of the selected distributed hosts is 32-bit. If not specified, the default value is native architecture. For example, 32bit on 32-bit systems and 64bit on 64-bit systems.
- `-farm lsf | grd | generic | now`
Specifies the type of the farm from which the distributed hosts are to be acquired.
- `[-setup_path path]`
Specifies the path to the submission scripts and executables for load sharing facility (LSF), Global Resource Director (GRD), and generic farm types. The path should contain the following scripts and executables:
- | | |
|----------------------------|---|
| <code>-farm grd</code> | setup_path should contain "qsub", "qdel" |
| <code>-farm lsf</code> | setup_path should contain "bsub", "bkill" |
| <code>-farm generic</code> | setup_path should contain "gsub", "gdel" |
- When you specify the `-setup_path` option, the submission to the compute resource takes the form "setup_path/submission script".
The `-setup_path` option is not valid with the `-farm now` option.
- `[-options string]`
Used only in conjunction with `-farm lsf | grd | generic`. It is a list of arguments to be passed off to the submission_script used for job submission to a LSF, GRD, or generic queue.
- `[-num_of_hosts count]`
Specifies the number of hosts to use in an LSF, GRD, generic, or now compute environment. The number must be at least 1. If you do not specify this option, the number of hosts is 1.
- `-submission_script`
Specifies the script to call when submitting jobs to LSF, GRD, or generic farm types. When you specify both the `-setup_path` and `-submission_script`

options, the submission to the compute resource takes the form "setup_path/submission script".

The *-submission_script* option is not valid with the *-farm now* option.

hostname

Specifies a single UNIX host. This option is used only in conjunction with the *-farm now* option.

DESCRIPTION

The options for the **add_distributed_hosts** command depend on whether you have an on-site load balancing capability and whether or not you want to use it.

If on-site load balanced facilities are available, you should specify them using the *-farm lsf*, *-farm grd*, or *-farm generic* options, depending on the type of resource available.

If no on-site load balancing capabilities are available or there are other compute resources available that are not under load balancing control, you should specify them using the *-farm now hostname* options.

All combinations of resources are concurrently supported by this command; however, each separate type must be specified independently by separately invoking the **add_distributed_hosts** command.

EXAMPLES

In the following example, two hosts are added for a distributed command. The first host, platinum1, is specified to have 2 hosts and the second, platinum2, is specified to have 4 hosts. The processes on platinum1 is 64-bit where as the processes on platinum2 is 32-bit. Notice that the actual number of processors in platinum1 and platinum2 is irrelevant, but, for optimal performance, the number or processes allocated should match the processors in each host. For example, 2 processors in platinum1 and 4 processors in platinum2.

```
pt_shell> add_distributed_hosts -farm now -num_of_hosts 2 platinum1
1
pt_shell> add_distributed_hosts -farm now -num_of_hosts 4 platinum2 -32bit
1
```

To use this command where on-site load balancing capabilities (LSF/GRD/GENERIC) are in place, use the *-farm lsf*, *-farm grd*, or *-farm generic* options. In the example below, the distributed hosts list is populated by defining four 32-bit hosts from an LSF farm, and two 64-bit hosts from a GRD farm.

```
pt_shell> add_distributed_hosts -farm lsf -setup_path "/bin/lsf" \
-options { -R "tmp>300" -q} -num_of_hosts slaves 4 -32bit
1
pt_shell> add_distributed_hosts -farm grd -setup_path "/bin/grd" \
-options {-cwd -V -r y -P bnormal -l} -num_of_hosts slaves 2
```

add_distributed_hosts

add_to_collection

Adds objects to a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection add_to_collection  
base_collection  
object_spec  
[-unique]
```

Data Types

<i>base_collection</i>	collection
<i>object_spec</i>	list

ARGUMENTS

base_collection

Specifies the base collection to which objects are to be added. This collection is copied to the result collection, and objects matching *object_spec* are added to the result collection. The *base_collection* option can be an empty collection (empty string), subject to some constraints, explained in the DESCRIPTION.

object_spec

Specifies a list of named objects or collections to add.
If the base collection is heterogeneous, collections can be added to it.
If the base collection is homogeneous, the object class of each element in this list must be the same as in the base collection. If it is not the same class, it is ignored. From heterogeneous collections in the *object_spec*, only objects of the same class of the base collection are added. If the name matches an existing collection, the collection is used; otherwise, the objects are searched for in the database using the object class of the base collection.
The *object_spec* has some special rules when the base collection is empty, as explained in the DESCRIPTION.

-unique

Indicates that duplicate objects are to be removed from the resulting collection. By default, duplicate objects are not removed.

DESCRIPTION

The **add_to_collection** command allows you to add elements to a collection. The result is a new collection representing the objects in the *object_spec* added to the objects in the base collection.

Elements that exist in both the base collection and the *object_spec*, are duplicated in the resulting collection. Duplicates are not removed unless you use the **-unique** option. If the *object_spec* is empty, the result is a copy of the base collection.

If the base collection is homogeneous, the command searches in the database for any elements of the *object_spec* that are not collections, using the object class of the base collection. If the base collection is heterogeneous, all implicit elements of the *object_spec* are ignored.

When the *base_collection* argument is the empty collection, some special rules apply to the *object_spec*. If the *object_spec* is non-empty, there must be at least one homogeneous collection somewhere in the *object_spec* list (its position in the list does not matter). The first homogeneous collection in the *object_spec* list becomes the base collection and sets the object class for the function. The examples show the different errors and warnings that can be generated.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example from PrimeTime (using the **get_ports** command) gets all ports beginning with 'mode', then adds the "CLOCK" port.

```
pt_shell> set xports [get_ports mode*]
{"mode[0]", "mode[1]", "mode[2]"}
pt_shell> add_to_collection $xports [get_ports CLOCK]
{"mode[0]", "mode[1]", "mode[2]", "CLOCK"}
```

The following example from PrimeTime adds the cell u1 to a collection containing the SCANOUT port.

```
pt_shell> set sp [get_ports SCANOUT]
{"SCANOUT"}
pt_shell> set het [get_cells u1]
{"u1"}
pt_shell> query_objects -verbose [add_to_collection $sp $het]
{"port:SCANOUT", "cell:u1"}
```

The following examples show how **add_to_collection** behaves when the base collection is empty. Adding two empty collections yields the empty collection. Adding an implicit list of only strings or heterogeneous collections to the empty collection generates an error message, because no homogeneous collections are present in the *object_spec* list. Finally, as long as one homogeneous collection is present in the *object_spec* list, the command succeeds, even though a warning message is generated. The example uses the variable settings from the previous example.

```
pt_shell> sizeof_collection [add_to_collection "" ""]
0
```

```
pt_shell> set A [add_to_collection "" [list a $het c]]
Error: At least one homogeneous collection required for argument 'object_spec'
to add_to_collection when the 'collection' argument is empty (SEL-014)
```

```
pt_shell> add_to_collection "" [list a $het $sp]
Warning: Ignored all implicit elements in argument 'object_spec'
to add_to_collection because the class of the base collection
```

```
could not be determined (SEL-015)
{"SCANOUT", "u1", "SCANOUT"}
```

SEE ALSO

```
collections(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
```

add_variation

Sums two or more variations. Returns a collection that corresponds to this sum variation.

SYNTAX

```
collection add_variation  
variation_list
```

Data Types

```
variation_list    collection
```

ARGUMENTS

```
variation_list  
    Lists the variations to be added up.
```

DESCRIPTION

Enables the addition of two or more variations. The sum is a variation. This command creates this sum variation and returns it as a collection.

EXAMPLES

The following example adds two variations, \$v1 and \$v2, (assuming they have already been created) into collection \$vsum, which is a new variation collection.

```
pt_shell> set col [add_to_collection $v1 $v2]  
_sel128
```

```
pt_shell> set vsum [add_variation $col]  
_sel129
```

SEE ALSO

```
create_distribution(2)  
sub_variation(2)  
max_variation(2)  
min_variation(2)
```

all_clocks

Creates a collection of all clocks in the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection all_clocks
```

ARGUMENTS

None.

DESCRIPTION

The **all_clocks** command creates a collection of all clocks in the current design. If you do not define any clocks, the empty collection (empty string) is returned.

If you want only certain clocks, use **get_clocks** to create a collection of clocks matching a specific pattern and optionally pass in filter criteria.

EXAMPLES

The following example applies the **set_propagated_clock** command to all clocks in the design.

```
pt_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

```
collections(2)  
create_clock(2)  
derive_clocks(2)  
get_clocks(2)  
set_propagated_clock(2)
```

all_connected

Creates a collection of objects connected to a net, pin, or port object. You can assign this collection to a variable or pass it into another command.

SYNTAX

```
collection all_connected  
object_spec  
[-leaf]
```

Data Types

object_spec list

ARGUMENTS

object_spec
Specifies the object whose connections are returned. This is a collection of one element which is a net, pin, or port collection, or the name of a net, pin, or port.

-leaf

global or leaf pins. When specified, this gives the leaf pins of a hierarchical net. For non-hierarchical nets, there is no difference in output.

DESCRIPTION

The **all_connected** command creates a collection of objects connected to a specified net, pin, or port. The *object_spec* option is either a collection of exactly one net, pin, or port object, or a name of an object. If it is a name, PrimeTime searches for a net, pin, or port, in that order. If the *object_spec* refers to a net, the *-leaf* option can be used to get the global leaf pins of the net.

The command returns a collection. The collection can contain nets, ports, pins, or a combination of ports and pins. In the latter case, the ports will be first, followed by the pins.

When issued from the command prompt, **all_connected** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example shows all objects connected to net "CLOCK":

```
pt_shell> query_objects -verbose [all_connected [get_nets CLOCK]]  
{"port:CLOCK", "pin:U1/CP", "pin:U2/CP", "pin:U3/CP", "pin:U4/CP"}
```

SEE ALSO

```
collections (2)  
get_nets (2)  
get_pins (2)  
query_objects (2)  
collection_result_display_limit(3)
```

all_correlations

Creates a collection of all correlations in the current design. You can assign these correlations to a variable or pass them into another command.

SYNTAX

```
collection all_correlations
```

ARGUMENTS

None.

DESCRIPTION

The **all_correlations** command creates a collection of all correlations in the current design. If you do not define any correlations, the empty collection (empty string) is returned.

If you want only specific correlations, use the **get_correlations** command to create a collection of correlations matching a specific pattern and optionally pass in filter criteria.

EXAMPLES

The following example creates a list of all correlations and assigns the list to `corr_list`.

```
pt_shell> set corr_list [all_correlations]
```

SEE ALSO

```
collections(2)  
create_correlation(2)  
get_correlations(2)
```

all_fanin

Creates a collection of pins/ports or cells in the fanin of specified sinks.

SYNTAX

```
collection all_fanin  
-to sink_list  
[-flat] [-only_cells]  
[-startpoints_only]  
[-levels level_count]  
[-pin_levels pin_count]  
[-trace_arcs arc_types]  
[-step_into_hierarchy]
```

Data Types

<i>sink_list</i>	list
<i>level_count</i>	int

ARGUMENTS

-to *sink_list*

Specifies a list of sink pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. The timing fanin of each sink in *sink_list* becomes part of the resulting collection. If a net is specified, the effect is the same as listing all driver pins on the net. This argument is required.

-startpoints_only

When this option is specified, only the timing start points are included in the result.

-only_cells

The result includes only cells in the timing fanin of the *sink_list* and not pins or ports.

-flat

There are two major modes in which **all_fanin** functions: hierarchical (the default) and flat. When in hierarchical mode, only objects within the same hierarchical level as the current sink are included in the result. In flat mode, the only non-leaf objects in the result are hierarchical sink pins.

-levels *cell_count*

The traversal stops when reaching a depth of search of *cell_count* hops, where the counting is performed over the layers of cells of same distance from the sink.

-pin_levels *pin_count*

The traversal stops when reaching a depth of search of *pin_count* hops, where the counting is performed over the layers of pins of same distance from the sink.

`-trace_arcs arc_types`

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (the default), which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); **enabled**, which permits the tracing of all enabled arcs and disregards case analysis values; and **all**, which permits the tracing of all combinational arcs regardless of either case analysis or arc disabling. (Note that the **enabled** option corresponds to the default behavior of this command in releases of PrimeTime prior to 2006.12.)

`-step_into_hierarchy`

This option may only be used in hierarchical mode and only has effect with either **-levels** or **-pin_levels**. Without the switch, a hierarchical block at the same level of hierarchy as the current sink is considered to be a cell; the input pins are considered a single level away from the related output pins, regardless of what is inside the block. With the switch enabled, the counting is performed as though the design were flat, and although pins inside the hierarchy are not returned, they determine the depth of the related output pins.

DESCRIPTION

The **all_fanin** command creates a collection of objects in the timing fanin of specified sink pins/ports or nets in the design. A pin is considered to be in the timing fanin of a sink if there is a timing path through combinational logic from the pin to that sink (please also see the **-trace_arcs** option). The fanin stops at the clock pins of registers (sequential cells).

If a current instance in the design is not the top level of hierarchy, only objects within the current instance are returned.

EXAMPLES

This example shows the timing fanin of a port in the design. The fanin includes a register, reg1.

```
pt_shell> query_objects [all_fanin -to out_1]
{"out_1", "reg1/Q", "reg1/CP"}
```

This example shows the flat mode of **all_fanin**. The sink is an input pin of a hierarchical cell, H1, which is connected to an output pin of another hierarchical cell, H2. H2 contains additional hierarchy and eventually, a leaf cell with 2 inputs, each of which has a top level register in its fanin.

```
pt_shell> query_objects [all_fanin -to H1/a -flat]
{"H1/a", "H2/U1/n1/Z", "H2/U1/n1/A", "H2/U1/n1/B",
 "reg1/Q", "reg2/Q", "reg1/CP", "reg2/CP"}
```

SEE ALSO

`all_fanout(2)`
`report_transitive_fanin(2)`
`current_instance(2)`

`all_fanin`

all_fanout

Creates a collection of pins/ports or cells in the fanout of the specified sources.

SYNTAX

```
collection all_fanout  
-from source_list  
-clock_tree [-flat]  
[-only_cells] [-endpoints_only]  
[-levels level_count]  
[-pin_levels pin_count]  
[-trace_arcs arc_types]  
[-step_into_hierarchy]
```

Data Types

source_list list *level_count* int

ARGUMENTS

-from *source_list*

Specifies a list of source pins, ports, or nets in the design. Each object is a named pin, port, or net, or a collection of pins, ports, or nets. The timing fanout of each source in *source_list* becomes part of the resulting collection. If a net is specified, the effect is the same as listing all load pins on the net. This option is exclusive with the **-clock_tree** option.

-clock_tree

Indicates that all clock source pins and/or ports in the design are to be used as the list of sources. Clock sources are specified using **create_clock**. If there are no clocks, or if the clocks have no sources, the result is the empty collection. This option is exclusive with the **-from** option.

-endpoints_only

When this option is specified, only the timing endpoints are included in the result.

-only_cells

The result includes only cells in the timing fanout of the *source_list* and not pins or ports.

-flat

There are two major modes in which the **all_fanout** command functions: hierarchical (default) and flat. When in hierarchical mode, only objects within the same hierarchical level as the current source are included in the result. In flat mode, the only non-leaf objects in the result are hierarchical source pins.

-levels *cell_count*

The traversal stops when reaching a depth of search of *cell_count* hops, where the counting is performed over the layers of cells of same distance from the

source.

-pin_levels pin_count

The traversal stops when reaching a depth of search of *pin_count* hops, where the counting is performed over the layers of pins of same distance from the source.

-trace_arcs arc_types

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (default), **enabled**, and **all**. The **timing** value permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis). The **enabled** value permits the tracing of all enabled arcs and disregards case analysis values. The **all** value permits the tracing of all combinational arcs regardless of either case analysis or arc disabling. (Note that the **enabled** option corresponds to the default behavior of this command in releases of PrimeTime prior to version Z-2006.12.)

-step_into_hierarchy

This option may only be used in hierarchical mode and only has effect with either the **-levels** or **-pin_levels** option. Without the switch, a hierarchical block at the same level of hierarchy as the current sink is considered to be a cell; the output pins are considered a single level away from the related input pins, regardless of what is inside the block. With the switch enabled, the counting is performed as though the design were flat, and although pins inside the hierarchy are not returned, they determine the depth of the related input pins.

DESCRIPTION

The **all_fanout** command creates a collection of objects in the timing fanout of specified source pins/ports or nets in the design. A pin is considered to be in the timing fanout of a source if there is a timing path through combinational logic from that source to the pin (please also see the **-trace_arcs** option). The fanout stops at the inputs to registers (sequential cells). The sources are specified using either **-clock_tree** or **-from source_list**.

If a current instance in the design is not the top level of the hierarchy, only objects within the current instance are returned.

EXAMPLES

This example shows the timing fanout of a port in the design. The fanout includes a register, reg3.

```
pt_shell> query_objects [all_fanout -from in1]
{"in1", "reg3/D"}
```

This example shows the difference between the hierarchical and flat modes of **all_fanout**. The source is an output pin of a hierarchical cell, H3/z1, which is connected to an input pin of another hierarchical cell, H4/a. H4 contains a leaf cell U1 with input A and output Z. The first command is hierarchical mode, and shows that hierarchical pins are included in the result. The second command is in leaf

mode, and leaf pins from the lower level are included.

```
pt_shell> query_objects [all_fanout -from H3/z1]  
{ "H3/z1", "H4/a", "H4/z", "reg2/D" }
```

```
pt_shell> query_objects [all_fanout -from H3/z1 -flat]  
{ "H3/z1", "H4/U1/A", "H4/U1/Z", "reg2/D" }
```

SEE ALSO

```
all_fanin(2)  
report_transitive_fanin(2)  
current_instance(2)
```

all_inputs

Creates a collection of all input ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_inputs
[-level_sensitive]
[-edge_triggered]
[-clock clock_name]
```

Data Types

clock_name list

ARGUMENTS

-level_sensitive

Only considers ports with level-sensitive input delay. This is specified by the **set_input_delay 2 -clock CLK -level_sensitive IN1** command.

-edge_triggered

Only considers ports with edge-triggered input delay. This is specified by **set_input_delay 2 -clock CLK IN2**.

-clock *clock_name*

Only considers ports with input delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

DESCRIPTION

The **all_inputs** command creates a collection of all input or inout ports in the current design. You can limit the contents of the collection by specifying the type of input delay that must be on a port.

If you want only certain ports, use the **get_ports** command to create a collection of ports matching a specific pattern and optionally passing filter criteria.

When issued from the command prompt, the **all_inputs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example specifies a driving cell for all input ports.

```
all_inputs
```



```
pt_shell> set_driving_cell -lib_cell FFD3 -pin Q [all_inputs]
```

SEE ALSO

```
collections(2)  
get_ports(2)  
report_port(2)  
query_objects(2)  
set_driving_cell(2)  
set_input_delay(2)  
collection_result_display_limit(3)
```

all_instances

Creates a collection of all instances of a specific design or library cell in the current design, relative to the current instance. You can assign the resulting collection of cells to a variable or pass it into another command.

SYNTAX

```
collection all_instances  
[-hierarchy]  
object_spec
```

ARGUMENTS

-hierarchy
Searches for instances in all levels of instance hierarchy below the current instance. By default, only instances from the current level of hierarchy are considered.

object_spec
Specifies the target design or library cell. This can be a design collection, lib_cell collection, or name.

DESCRIPTION

The **all_instances** command creates a collection of cells that are instances of a design or library cell. The search for instances is made relative to the current instance within the current design. By default, **all_instances** considers only instances at the current level of the hierarchy. If you use the **-hierarchy** option, the search continues throughout the hierarchy.

The *object_spec* can be a simple name. In this case, any instance of a design or lib_cell with that name will match. Alternatively, the *object_spec* can be a collection of exactly one design or one library cell. Any other collection results in an error message. Using the collection can help focus the search for instances of specific designs or lib_cells, especially after **swap_cell** has been used.

When issued from the command prompt, **all_instances** behaves as though **query_objects** has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example uses **all_instances** to get the instances of the design 'low' in the current level of hierarchy.

```
pt_shell> all_instances low  
{"U1", "U3"}
```

```
all_instances  
18
```

The following example uses **-hierarchy** to display instances of a design across multiple levels of hierarchy.

```
pt_shell> query_objects [all_instances low -hierarchy]
{"U1", "U2/U1", "U3"}
```

In the following example, there are two instances of design "inter". One of them is swapped for a new design. The difference between using **all_instances** with a name and a collection of one design is shown.

```
pt_shell> swap_cell i1 inter_2.db:inter
1
pt_shell> all_instances inter
{"i1", "i2"}
pt_shell> all_instances [get_designs inter_2.db:inter]
{"i1"}
```

SEE ALSO

```
collections(2)
current_design(2)
current_instance(2)
get_designs(2)
get_lib_cells(2)
get_cells(2)
list_designs(2)
query_objects(2)
collection_result_display_limit(3)
```

all_outputs

Creates a collection of all output ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_outputs [-level_sensitive]
[-edge_triggered]
[-clock clock_name]
```

Data Types

clock_name list

ARGUMENTS

-level_sensitive

Only considers ports with level-sensitive output delay. This is specified by **set_output_delay 2 -clock CLK -level_sensitive IN1**.

-edge_triggered

Only considers ports with edge-triggered output delay. This is specified by **set_output_delay 2 -clock CLK IN2**.

-clock *clock_name*

Only considers ports with output delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

DESCRIPTION

The **all_outputs** command creates a collection of all output or inout ports in the current design. You can limit the contents of the collection by specifying the type of output delay that must be on a port.

If you want only certain ports, use **get_ports** to create a collection of ports matching a specific pattern and optionally passing filter criteria.

When issued from the command prompt, **all_outputs** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example specifies a pin capacitance for all output ports.

```
pt_shell> set_load 4.56 [all_outputs]
```

The following example shows the names all of the output ports with output delay relative to PHI1.

```
pt_shell> all_outputs -clock PHI1
```

SEE ALSO

```
collections (2)  
get_ports (2)  
report_port (2)  
query_objects (2)  
set_output_delay (2)  
collection_result_display_limit(3)
```

all_registers

Creates a collection of register cells or pins. You can assign the resulting collection to a variable or pass it into another command.

SYNTAX

```
collection all_registers
[-clock clock_name]
[-rise_clock rise_clock_name]
[-fall_clock fall_clock_name]
[-cells] [-data_pins] [-clock_pins] [-slave_clock_pins] [-async_pins] [-output_pins]
[-level_sensitive] [-edge_triggered] [-master_slave] [-no_hierarchy]
```

Data Types

<i>clock_name</i>	list
<i>rise_clock_name</i>	list
<i>fall_clock_name</i>	list

ARGUMENTS

- clock *clock_name*
Considers all registers clocked by *clock_name*. This is either the name of a clock, or a collection containing a clock. For example, all registers whose clock pins are in the fan out of the specified clock.
- rise_clock *rise_clock_name*
Considers all registers clocked by *rise_clock_name* and having open edge effectively the rising clock edge. This is either the name of a clock, or a collection containing a clock. For example, rising edge triggered flip flops without any inversion on the clock path or falling edge triggered flip flops with inversion on the clock path.
- fall_clock *fall_clock_name*
Considers all registers clocked by *fall_clock_name* and having open edge effectively falling the clock edge. This is either the name of a clock, or a collection containing a clock. For example, rising edge triggered flip flops with inversion on the clock path or falling edge triggered flip flops without any inversion on the clock path.
- cells
Creates a collection of cells (default). The cells are registers and are further limited by other command options.
- data_pins
Creates a collection of register data pins. The collection can be limited by other command options.
- clock_pins
Creates a collection of register clock pins. The collection can be limited by other command options.

-slave_clock_pins
Creates a collection of register slave clock pins (the slave clock pins of master-slave registers). Specify slave clock pins as **clocked_on_also** in the library.

-async_pins
Creates a collection of async preset or clear pins.

-output_pins
Creates a collection of register output pins.

-level_sensitive
Limits search to level-sensitive latches.

-edge_triggered
Limits search to edge-triggered flip-flops.

-master_slave
Only considers master or slave register cells.

-no_hierarchy
Only searches the current instance; does not descend the hierarchy.

DESCRIPTION

The **all_registers** command creates a collection of pins or cells related to registers.

When issued from the command prompt, **all_registers** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

This example performs a trace from a port to all register clock pins which are clocked by CLK.

```
pt_shell> report_timing -from [get_ports {CLK}] \
          -to [all_registers -clock CLK -clock_pins]
*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : counter
Version: 1997.01-development
Date   : Thu Jul 25 14:23:48 1996
*****
```

Startpoint: CLK (input port clocked by CLK)
 Endpoint: ffa/CP (internal pin)
 Path Group: (none)
 Path Type: max

Point	Incr	Path

input external delay	0.00	0.00 r
CLK (in)	0.00	0.00 r
ffa/CP (FD2)	0.00	0.00 r
data arrival time		0.00

(Path is unconstrained)		

The following example performs a trace from all registers clock pins, which are triggered by clock rising edge.

```
pt_shell> report_timing -from [all_registers -rise_clock \
                  CLK -clock_pins]
```

```
*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : li_FD3x
Version: 2002.03-SI1
Date   : Tue Nov 13 11:07:29 2001
*****
```

Startpoint: ff3 (falling edge-triggered flip-flop clocked by clk')
 Endpoint: ff4 (falling edge-triggered flip-flop clocked by clk')
 Path Group: clk
 Path Type: max

Point	Incr	Path

clock clk' (fall edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
ff3/CP (FD3x)	0.00	0.00 f
ff3/Q (FD3x)	1.44	1.44 f
iv2/Z (IVA)	0.31	1.75 r
ff4/D (FD3x)	0.00	1.75 r
data arrival time		1.75
clock clk' (fall edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
ff4/CP (FD3x)		10.00 f
library setup time	-0.90	9.10
data required time		9.10

data required time		9.10
data arrival time		-1.75

slack (MET)

7.35

SEE ALSO

`collections(2)`

`get_cells(2)`

`get_pins(2)`

`collection_result_display_limit(3)`

all_variations

Creates a collection of all variations in the current design. You can assign these variations to a variable or pass them into another command.

SYNTAX

```
collection all_variations
```

ARGUMENTS

None.

DESCRIPTION

The **all_variations** command creates a collection of all variations in the current design. If you do not define any variations, the empty collection (empty string) is returned.

If you want only certain variations, use **get_variations** to create a collection of variations matching a specific pattern and optionally pass in filter criteria.

EXAMPLES

The following example creates a list of all variations and assigns the list to `corr_list`.

```
pt_shell> set corr_list [all_variations]
```

SEE ALSO

```
collections(2)  
create_variation(2)  
get_variations(2)
```

append_to_collection

Add object(s) to a collection. Modifies variable.

SYNTAX

```
collection add_to_collection  
var_name  
object_spec  
[-unique]  
collection var_name  
list      object_spec
```

ARGUMENTS

var_name
Specifies a variable name. The objects matching *object_spec* are added into the collection referenced by this variable.

object_spec
Specifies a list of named objects or collections to add.

-unique
Indicates that duplicate objects are to be removed from the resulting collection. By default, duplicate objects are not removed.

DESCRIPTION

The **append_to_collection** command allows you to add elements to a collection. This command treats the variable name given by *var_name* as a collection, and appends all of the elements in *object_spec* to that collection. If the variable does not exist, it is created as a collection with elements from *object_spec* as its value. If the variable does exist, and it does not contain a collection, it is an error.

The result of the command is the collection which was initially referenced by *var_name*, or the collection created if the variable did not exist.

The **append_to_collection** command provides the same semantics as a common use of **add_to_collection** but with significant improvement in performance. An example of replacing **add_to_collection** with **append_to_collection** is given below.

```
set var_name [add_to_collection $var_name $objs]
```

Using **add_to_collection** this becomes:

```
append_to_collection var_name $objs
```

The **append_to_collection** command can be much more efficient than **add_to_collection** if you are building up a collection in a loop. The arguments of the command have the same restrictions as the **add_to_collection** command. Please see the **add_to_collection**

man page for more information about those restrictions.

EXAMPLES

The following example from PrimeTime shows how a collection can be built up with **append_to_collection**:

```
pt_shell> set xports
Error: can't read "xports": no such variable
      Use error_info for more info. (CMD-013)
pt_shell> append_to_collection xports [get_ports in*]
{"in0", "in1", "in2"}
pt_shell> append_to_collection xports CLOCK
{"in0", "in1", "in2", "CLOCK"}
```

SEE ALSO

add_to_collection (2), **foreach_in_collection** (2), **index_collection** (2),
remove_from_collection (2), **sizeof_collection** (2).

cell_of

Creates a collection of cells of the given pins. The **cell_of** command is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
string cell_of  
object_list
```

Data Types

```
object_list      list
```

ARGUMENTS

```
object_list  
    Creates a list of pins for which cells to get.
```

DESCRIPTION

The **cell_of** command creates a collection of cells owned by a list of pins. The command exists in PrimeTime for compatibility with Design Compiler. Complete information about the **cell_of** command can be found in the Design Compiler documentation. The supported method for getting pins of cells is to use the **get_cells** command with the *-of_objects* option.

SEE ALSO

```
get_cells(2)
```

change_histogram

Changes the graph and display attributes of an existing histogram.

SYNTAX

```
int change_histogram
    [-numbins number_of_bins]
    [-min min_value]
    [-max max_value]
    [-greater_than gt_value]
    [-less_than lt_value]
    [-midpoint mid_value]
    [-worst worst_side]
    [-value_label value_label]
    [-object_name object_name]
    [-title title_label]
    [-xlabel x_axis_label]
    [-ylabel y_axis_label]
    -view view_to_use
```

Data Types

<i>number_of_bins</i>	int
<i>min_value</i>	float
<i>max_value</i>	float
<i>gt_value</i>	float
<i>lt_value</i>	float
<i>mid_value</i>	float
<i>worst_side</i>	string
<i>value_label</i>	string
<i>object_name</i>	string
<i>title_label</i>	string
<i>x_axis_label</i>	string
<i>y_axis_label</i>	string
<i>view_to_use</i>	string

ARGUMENTS

-numbins *number_of_bins*
Specifies, between the integers 1 to 300, the number of bins to use when changing the histogram.

-min *min_value*
Specifies the minimum value to use in binning objects. Any object whose associated value is less than this value is to be dropped from the histogram with a warning message.

-max *max_value*
Specifies the maximum value to use in binning objects. Any object whose associated value is greater than this value is to be dropped from the histogram with a warning message.

`-greater_than gt_value`
 Specifies the non-inclusive least value limit to use in binning objects. Any object whose associated value is less than or equal to this value is to be dropped from the histogram with a warning message.

`-less_than gt_value`
 Specifies the non-inclusive greatest value limit to use in binning objects. Any object whose associated value is greater than or equal to this value is to be dropped from the histogram with a warning message.

`-midpoint mid_value`
 Specifies the midpoint value to use in interpreting histogram bins. An x-axis tick mark is placed on the midpoint value and all bins to the **worst** side of the midpoint are colored red while all bins to the other side of the midpoint are colored green. The **worst** side is **none** by default but can be set to **left** or **right** using the **-worst** option.

`-worst worst_side`
 Specifies whether values less than or greater than the midpoint are to be considered "worst." Allowed values are **left**, meaning that values less than midpoint are worse; **right**, meaning that values greater than the midpoint are worse; and **none** (the default), meaning that neither is worse.

`-value_label value_label`
 Specifies the label to place at the head of the value column in the list view accompanying the histogram.

`-object_name object_name`
 Specifies the label to place at the head of the object name column in the list view accompanying the histogram.

`-title title_label`
 Specifies the label to place at the top of the histogram graph.

`-xlabel x_axis_label`
 Specifies the label to place along the X axis of the histogram graph.

`-ylabel y_axis_label`
 Specifies the label to place along the Y axis of the histogram graph.

`-view name_of_view`
 Specifies the name of the histogram whose graph attributes are to be changed by the command. This argument is required.

DESCRIPTION

The **change_histogram** command changes the graph and display attributes of an existing histogram. If the number of bins, minimum, maximum, or midpoint values are changed, the objects are binned again. Other arguments affect only display attributes (for example, labels). The **-view** argument is required, and specifies the histogram view to which the new graph and display attributes are to be applied. You can use this command only while the GUI is active; type the command into the command line field at the bottom of the GUI console window.

Any graph or display attribute that is not changed by one of the arguments to this command remains unchanged. For default values of the attributes, see the **create_histogram** man page.

EXAMPLES

The following example shows changing the histogram graph in the view named "histogram_1" so that the objects are binned again using 4 bins in the value range greater than 0 and less than 10.0. The title changes to read "0.0 to 10.0".

```
pt_shell> change_histogram -title "0.0 to 10.0" -numbins 4 \  
          -greater_than 0.0 -less_than 10.0 -view histogram_1  
1
```

SEE ALSO

`create_histogram(2)`

change_selection

Changes the selection in the GUI.

SYNTAX

```
int change_selection
[-name slct_bus]
[-replace (default) ]
[-add ]
[-remove ]
[-toggle ]
[-type object_type]
[-clock_trees clock_tree_list]
[collection]
```

Data Types

<i>slct_bus</i>	string
<i>object_type</i>	string
<i>clock_tree_list</i>	list
<i>collection</i>	list

ARGUMENTS

-name *slct_bus*
Specifies the selection bus to which the change is made. The default is *global*, the default global selection bus.

-replace
Replaces current selection with the objects in the collection. This is the default behavior if no optional parameter is specified.

-add
Adds the objects in the collection to the current selection.

-remove
Removes the objects that are specified in the collection from current selection.

-toggle
Toggles the selection state of the objects that are specified in the collection.

-type *object_type*
Specifies the type to change. Only those items from the collection that are of the type specified are used to change the selection. If not specified, the entire collection is used. The *object_type* option can be one of the following types:

- design
- port

- net
- cell
- pin
- path (timing path)

`-clock_trees clock_tree_list`

Lists the clock trees available to use to change the selection. The type of change that is applied to the current selection with the clock tree list is specified by the optional parameters listed above.

`collection`

Specifies the collection of objects to use to change the selection. The type of change that is applied to the current selection with the collection is specified by the optional parameters listed above.

DESCRIPTION

The **change_selection** command changes the selection in the GUI. When selections are changed, the GUI updates all the relevant windows to reflect this change.

A collection of objects and the type of change are given as inputs to the command. The collection of objects could be returned as the result of another command such as the **get_designs** command. If the collection is empty and the *-replace* option is specified (or no option specified), the current selection is cleared.

If the *-type* option is used, only the type of objects specified are used to change the current selection. For example, if a *-type* design is used, only the design objects in the collection are used to change the current selection. If no *-type* option is specified, all the objects in the collection are used to change the current selection. For example, if the *-add* option is used with no *-type*, all the objects regardless of type of object are added to the current selection.

For information about collections, see the **collections** man page.

EXAMPLES

The following example replaces the current selection with the collection of design objects, regardless of type.

```
pt_shell> change_selection [get_designs]
```

This example adds cell U5 to the selection.

```
pt_shell> change_selection -add [get_cells U5]
```

This example removes all the objects of type pin from the current selection.

```
pt_shell> change_selection -remove -type pin [get_selection]
```

change_selection

This example clears all the selection.

```
pt_shell> change_selection ""
```

SEE ALSO

```
collections(2)  
filter(2)  
filter_collection(2)  
get_selection(2)  
query_objects(2)
```

characterize_context

Captures the timing context of a list of instances.

SYNTAX

```
string characterize_context  
[-timing]  
[-environment]  
[-design_rules]  
[-constant_inputs]  
[-no_boundary_annotations]  
cell_list
```

Data Types

cell_list list

ARGUMENTS

-timing
Characterizes timing information; for example, clocks, input and output delays, and timing exceptions.

-environment
Characterizes environment-related information; for example, operating conditions (process, temperature, and voltage), wire load model, capacitive loads on input and output pins, and driving cell information on input pins.

-design_rules
Characterizes design rules; for example, max_capacitance, max_transition, and max_fanout.

-constant_inputs
Characterizes logic constants propagated to input pins of the instance being characterized by the case analysis capability of PrimeTime.

-no_boundary_annotations
Disables characterization of annotated capacitance on boundary nets as annotated capacitance in the characterized instance. Instead, the port wire capacitance is adjusted to account for any difference between the estimated and annotated values. By default, PrimeTime characterizes annotated capacitance on boundary nets as annotated capacitance in the characterized instance.

cell_list
Specifies a list of instances to characterize.

DESCRIPTION

Captures the timing context of instances of subdesigns from the chip-level timing environment. The timing context of an instance is defined as waveforms of the clocks

affecting this instance, input arrival times, output required times, timing exceptions, design rules, propagated constants, wire load models, input drives, and capacitive loads.

The **characterize_context** command does not capture delays and parasitics annotated on the internal nets of the instance being characterized. To capture this information, use the **write_physical_annotations** command without the **-boundary_nets** option.

All categories are characterized if no option is specified. To characterize one or more categories of interest, specify the respective option or list of options.

EXAMPLES

The following command derives only the timing-related context information for instance I1 and I2.

```
pt_shell> characterize_context -timing {I1 I2}
```

The following command derives the design rule and the logic constant information from the context of cell I2.

```
pt_shell> characterize -constant_inputs -design_rules I2
```

SEE ALSO

```
remove_context(2)  
report_context(2)  
write_context(2)  
write_physical_annotations(2)
```

check_block_scope

Checks the scope of hierarchical blocks that were replaced with timing models during the top-level analysis.

SYNTAX

```
int check_block_scope  
-instances instance_list  
[-scope_scenario scenario_name]  
[-check_types chk_types]  
[-significant_digits digits]  
[-absolute_clock_arrival]  
[-nosplit]  
[-verbose]  
file_name
```

ARGUMENTS

- instances *instance_list*
Specifies the list of instances for which scope needs to be verified. The scope information for the blocks corresponding to these instances should be contained in the file specified by the *file_name* option.
- scope_scenario *scenario_name*
Specifies the scenario name for which the scope data needs to be used to verify the top-level instantiation of the block as timing models. The scope information in the data file should contain data corresponding to the given scenario. If you do not specify, a fixed default name is used internally.
- check_types *chk_types*
Specifies the types of scope checks to be performed. By default, when this option is not specified, the types of checks defined by the *hier_scope_check_defaults* environment variable are performed. Use this option to overrides the globally defined default scope check types.
- significant_digits *digits*
Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.
- absolute_clock_arrival
Indicates that the clock arrival times reach the block boundary pins are to be checked as absolute rather than relative latency values against those captured as scope information during original block-level analysis. By default, when this option is not used, the arrival windows for clock signals are checked as relative arrival requirements. For more information, see the DESCRIPTION section.

`-nosplit`

Specifies that the printed report should not split the lines even when they become long and unfit for the line column number settings. By default, the report lines are split, when necessary.

DESCRIPTION

This command should be used while performing top-level analysis in a bottom-up hierarchical design flow. At the block-level, timing analysis has been performed and a timing model is created through the **create_ilm** or **extract_model** command. In addition, scope data can be generated for the models by specifying appropriate options to these two commands. The **check_block_scope** command checks the scope of block instances to make sure that the timing model accurately reflects the full block.

The checks that are performed are dictated by the **hier_scope_check_defaults** variable.

For a given block, when multiple clocks interact with each other and, therefore, should be maintained synchronous to each other, it indicates that the relative skews among these clocks should be maintained to safely move the analysis from block-level to top-level. It is not necessary to require for each and every clock that the absolute arrival times at the block boundary for all instances of the block remain unchanged. It is sufficient for all these clocks to incur a similar amount of insertion delay when propagate through their top-level clock trees before reaching the block boundary as long as their inter-clock skew relations are maintained. This is also applicable to blocks with only one clock.

When the **-absolute_clock_arival** option is not specified, PrimeTime automatically derives a best common shift, separately for each synchronous clocking domain and each instance of the block need to be checked, based on the original arrival windows captured into the scope file and the actual arrival windows obtained at top-level. This global shift is then uniformly applied across all the clocks in that domain such that the number of scope violations reported is minimized. Note that for the same block, different global shift might be derived for different instances. The goal of deriving and applying the global shift is to minimize the scope violations reported and therefore need to be fixed potentially. It may or may not correlate directly to the actual common insertion delay for all the related clocks in that domain of an instance of the block.

This flow can be used in both PrimeTime and PrimeTime SI.

EXAMPLES

This example uses the scope information saved in `top/ilm.scope` file to verify that the timing models for the I1 and I2 instances are within the validated scope of the block.

```
pt_shell> check_block_scope -instances {I1 I2} top/ilm.scope
```

SEE ALSO

`create_ilm(2)` `extract_model(2)` `create_scenario(2)` `hier_scope_check_defaults(3)`

check_level_shifter

Alias for the **check_timing -override_defaults {signal level}**.

SYNTAX

```
int check_level_shifter  
[-verbose]
```

ARGUMENTS

-verbose
Prints a detailed report of all possible violations.

DESCRIPTION

In PrimeTime, the **check_level_shifter** command is an alias for:

check_timing -override_defaults {signal_level}

SEE ALSO

check_timing(2)

check_noise

Performs checking whether there are necessary data available to run the update_noise command.

SYNTAX

```
int check_noise  
[-verbose]  
[-nosplit]  
[-beyond_rail]  
[-include check_options]
```

Data Types

check_list string

ARGUMENTS

- verbose
 Enables verbose mode showing detailed information and pin names.
- beyond_rail
 Enables beyond_rail noise analysis check. By default, **check_noise** performs only within_rail analysis.
- nosplit
 Prevents line-splitting and facilitates writing software to extract information from the report output, because most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.
- include
 Specifies the types of checking. Available values are noise_driver and noise_immunity. The default is noise_immunity.

DESCRIPTION

It is possible to have invalid noise models in a library and run noise analysis without detecting them. This may result in inaccurate results. If the design is large, and it takes a long time to finish the noise analysis, it also causes longer turn around time. Moreover, it is very important that all pins in the design have correct noise immunity information. Otherwise, when the noise analysis is over, violations are not reported even if noise bumps are large enough to create violations.

You can execute the **check_noise** command before the noise analysis to validate the correctness of a design with respect to the noise analysis. It checks any invalid noise model, any pin without a model from the library, and the design. It checks if all pins in the design are 'noise constrained', for example, so their noise immunity can be calculated.

check_noise

The command may produce different results before and after `update_timing`. For example, case analysis can disable noise models, and pins can be unconstrained after `update_timing`. It is recommended that `check_noise` is executed after `update_timing` to consider design context.

In a verbose mode, the command shows cell and pin names that have no model or invalid models.

EXAMPLES

The following example shows the noise immunity check.

```
pt_shell> check_noise
```

Noise immunity type	above_low	below_high

user hyperbolic curve	8	8
user margin	1	1
library immunity table	0	0
library hyperbolic curve	0	0
library CCS noise immunity	3245	3245
library DC noise margin	0	0
none	0	0

The following example shows the noise driver check.

```
pt_shell> check_noise -include "noise_driver"
```

Noise driver check:

Noise driver type	above_low	below_high

CCS noise	520	520
library IV curve	121	121
library resistance	0	0
none	1	1

The verbose mode shows which pin is not constrained.

```
pt_shell> check_noise -include "noise_driver" -verbose
```

Noise driver check:

library pin name	Region	Status

top/inv1/Z	above_low	None
Noise driver type	above_low	below_high

CCS noise	520	520
library IV curve	121	121
library resistance	0	0

none

1

1

SEE ALSO

`update_noise(2)`

`report_noise(2)`

`set_noise_parameters(2)`

check_power

Shows possible power problems for design.

SYNTAX

```
string check_power  
[-verbose]  
[-significant_digits digits]  
[-override_defaults check_list]  
[-include check_list]  
[-exclude check_list]
```

Data Types

```
digits          int  
check_list      list
```

ARGUMENTS

-verbose
Shows detailed information about potential problems.

-significant_digits *digits*
Specifies the number of digits of precision to be displayed by warnings that show floating point numbers. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-override_defaults *check_list*
Overrides the checks in **power_check_defaults** using *check_list*. See the man page of **power_check_defaults** for its default value.

-include *check_list*
Adds the checks listed in *check_list* to the checks in **power_check_defaults**.

-exclude *check_list*
Subtracts the checks listed in *check_list* from the checks in **power_check_defaults**.

check_list
Gives the list of checks to be performed. Each element in this list is one of the following strings: *no_arrival_window*, *no_base_clock*, *out_of_table_range*, *missing_table*, *missing_function*.

DESCRIPTION

The **check_power** command checks structure of the design for potential power violations. This command is used to identify possible power calculation problems before updating power or before generating power reports.

This command also prints which checks it performs. If a check reveals a violation,

then the command also prints a message about the violation. By default, the message contains a summary of the violation. To get more information about violations, use the **-verbose** option.

This command without any options performs the checks defined by the **power_check_defaults** variable. To change the value of this variable, either redefine it or use the **-override_defaults** option. If the **-override_defaults** option is not used, the final list of checks to be performed is the checks in **power_check_defaults** plus the list of checks given by the **-include** option minus the list of checks given by the **-exclude** option. If the **-override_defaults** option is used with a **check_list**, the final list of checks to be performed is the checks in the **check_list** given by the option plus the list of checks given by the **-include** option minus the list of checks given by the **-exclude** option.

The alphabetically ordered list below shows the meaning of each check:

missing_function

Checks if library cells have function statements. The function statements are used during toggle propagation and event simulation.

missing_table

Checks if library cells have leakage power tables and dynamic power tables.

no_arrival_window

Checks whether pins have arrival window or not. This check helps to generate accurate cycle-based average waveform, zero delay VCD power analysis, and RTL VCD power analysis. In order to save the arrival windows user should set the variable **timing_save_pin_arrival_and_slack** to true before **update_timing/update_power**. Users seeing this violation are directed to check if timing is correct by using the **check_timing** command.

no_base_clock

Checks whether nets have power base clock or not. During updating power, the fastest clock is used as power base clock for these nets that do not have power base clock. However, this exposes some potential accuracy problem. Users seeing this violation are directed to check if timing is correct by using the **check_timing** command.

out_of_table_range

Checks if there are out-of-range ramp or load violations when looking up power tables for each cells. The out of range values together with cell name, pin name, and library cell name and its range is displayed if the **"-verbose"** option is used.

SEE ALSO

power_check_defaults(3)

check_timing(2)

report_constraint(2)

check_timing

Shows possible timing problems for design.

SYNTAX

```
string check_timing  
[-verbose]  
[-significant_digits digits]  
[-ms_min_separation delta]  
[-override_defaults check_list]  
[-include check_list]  
[-exclude check_list]
```

Data Types

<i>delta</i>	float
<i>digits</i>	int
<i>check_list</i>	list

ARGUMENTS

- verbose
Shows detailed information about potential problems.
- significant_digits *digits*
Specifies the number of digits of precision to be displayed by warnings that show floating point numbers. Allowed values are 0-13. The default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.
- ms_min_separation *delta*
Minimum separation value between master and slave clocks. The default minimum separation is 0.0.
- override_defaults *check_list*
Overrides the checks in the **timing_check_defaults** variable using the *check_list* option. For default information, see the **timing_check_defaults** man page.
- include *check_list*
Adds the checks listed in the *check_list* option to the checks in **timing_check_defaults** variable.
- exclude *check_list*
Subtracts the checks listed in the *check_list* option from the checks in **timing_check_defaults** variable.
- check_list
Gives the list of checks to be performed. Each element in this list is one of the following strings: *clock_crossing* , *data_check_multiple_clock*, *data_check_no_clock*, *generated_clocks*, *generic*, *latch_fanout*, *latency_override*, *loops*, *ms_separation*, *multiple_clock*, *no_clock*,

no_input_delay, pll_configuration, retain, signal_level, supply_net_voltage, unconnected_pg_pins, and unconstrained_endpoints.

DESCRIPTION

Checks the assertions and structure of the design for potential timing violations. This command is used to identify possible problems before generating timing or constraint reports.

This command also prints which checks it performs. If a check reveals a violation, the command also prints a message about the violation. By default, the message contains a summary of the violation. To get more information about violations, use the **-verbose** option.

This command without any options performs the checks defined by the **timing_check_defaults** variable. To change the value of this variable, either redefine it or use the **-override_defaults** option. If the **-override_defaults** option is not used, the final list of checks to be performed is the checks in the **timing_check_defaults** variable plus the list of checks given by the **-include** option minus the list of checks given by the **-exclude** option. If the **-override_defaults** option is used with a *check_list*, the final list of checks to be performed is the checks in the *check_list* given by the option plus the list of checks given by the **-include** argument minus the list of checks given by the **-exclude** argument.

The alphabetically ordered list below shows the meaning of each check:

clock_crossing

Checks clock interactions when there are multiple clock domains. If a clock launches one or more paths, which are captured by other clocks, it will have an entry in clock crossing report. If all paths between two clocks are false paths or they are exclusive/asynchronous clocks, the path is marked by *. If only part of paths are set as false paths or exclusive/asynchronous clocks, the path is marked by #.

data_check_multiple_clock

Warns if multiple clocked signals reach a data check register reference pin. The analysis will be done for each of the clocked domain separately. If **timing_enable_multiple_clocks_per_reg** is set to FALSE, only the one of the clocked signal will be analyzed.

data_check_no_clock

Warns if no clocked signal reaches a data check register reference pin. In this case, no setup or hold checks are performed on the constrained pin.

generated_clocks

Checks generated clock network. The master must be driven by a clock source. There cannot be loops of generated clocks. For example, the source of generated clock CLK1 cannot be used to generate clock CLK2 if CLK2 also is used to generate CLK1.

generic

Warns about generic (unmapped) cells in the design. The timing of paths through generic cells is inaccurate because generic cells have zero delay.

check_timing

ideal_clocks

Shows the clocks that are not defined as propagated using the /fBset_propagated_clock/fP command. Generally, all clocks should be propagated so that the clock network timing is accurately calculated. Especially, in presence of crosstalk, the delay changes induced by other nets on the clock network are not reflected in the calculated slacks in the design.

latch_fanout

Checks fanout of level-sensitive latches. A warning is issued if a level-sensitive latch fans out to itself. An information message also appears for a latch that fans out to a latch of the same clock (PHI1 to PHI1 path).

latency_override

Warns of clock latency specification conflicts. If clock source latency is defined for both a clock and its port (source pin), the source latency for clock object is ignored. If input_delay is set on clock port, which also has source latency specified, the input_delay is ignored as a source latency. Also warns if more than one clock latency fan out to any latch clock pin.

loops

Warns of combinational feedback loops. Combinational feedback loops are not analyzed. If the feedback loop is not broken by **set_disable_timing**, it is automatically broken by disabling one or more timing arcs.

ms_separation

Checks minimum separation of master and slave clock pulses on master/slave latches.

multiple_clock

Warns if multiple clocks reach a register clock pin. If more than one clock signal reaches a register clock pin, and **timing_enable_multiple_clocks_per_reg** is set to FALSE, then it is undefined which one is used for analysis. In this case, use **set_case_analysis** so only one clock can propagate from its sources to the register clock pin. Using this check and the **no_clock** check run significantly faster than other checks. Hence, to save time, user may want to issue these checks separately from other checks.

no_clock

Warns if no clock reaches a register clock pin. In this case, no setup or hold checks are performed on data pins related to that clock pin, and the path starting at the clock pin is not relative to a clock. For performance of this check, see the description in the **multiple_clock** check.

no_driving_cell

Warns if a port does not have any driving cell. This warning is issued only when the net connected to the port has parasitics. In such case, the accuracy of delay calculation could be impacted, as a default strong driver is assumed in absence of driving cell definition. Especially, in presence of crosstalk, a port with no driving cell could act as a strong aggressor which could lead to significant amount of pessimism in the analysis. Also, a port with no driving cell could act as a string victim, which could underestimate the crosstalk effect.

no_input_delay

Warns if no clock related delay specified on an input port, where it propagates to a clocked latch or output port. With **-verbose**, the port name will be listed. Note that with **timing_input_port_default_clock** set to 'true', a default clock will be assumed for the input port. Otherwise it will not be clocked, and the paths are unconstrained. In this case, if there is no input delay specified, **check_timing** will not generate warnings.

partial_input_delay

Warns if any port has partially defined input delay. This happens when **set_input_delay -min** is applied on a port to set the min input delay with respect to a clock, however no **set_input_delay -max** is applied to that port to specify the max delay, or vice versa. As a result, some paths starting from the port with partially defined input delay may become unconstrained and some potential violations could be missed.

pll_configuration

Warns if a problem is detected in the configuration of any phase locked loop (PLL) cells. For a PLL to be correctly configured, the PLL output clock should reach the PLL feedback pin. If a PLL is not correctly configured, it will not show any phase adjustment on the PLL output clock.

retain

Warns if any of the RETAIN values is larger than its corresponding delay value. The RETAIN values should be less than their corresponding delay values so that they be considered for hold violations. Otherwise, they may be considered for setup violations.

signal_level

Checks that the driver signal level matches the load signal level. The signal levels are determined from the cell specific operating conditions and rail voltages (or UPF), and from the following library attributes: **input_signal_level**, **output_signal_level**, **input_voltage**, **output_voltage**. The check is performed on all input pins that have **input_voltage** defined in the timing library. If the driver pin does not have **output_voltage** defined in the library then the voltage of the rail powering the driver pin is used as the output signal level. With PG libraries the signal level is determined from **input/output_signal_level_high** (exact voltage in .lib) first, then **input/output_signal_level** (reference to voltage_map in .lib) if the **related_power_pin** uses a **different_voltage** map, and finally from the instance specific voltage of the related power PG pin. If the load pin does not have **input_voltage** then matching is done based on driver and load rail voltages. Such matching (without library attributes **input/output_voltage**) is subject to tolerances set by **set_level_shifter_threshold** and **set_level_shifter_strategyP** commands. By default the number of mismatches is reported. With **-verbose** the mismatching driver, load and voltage difference are reported.

supply_net_voltage

Checks that each segment of UPF supply nets has voltage assigned to it by **set_voltage** command.

unconnected_pg_pins

Checks that each power and ground pin is connected to a UPF supply net. The connection can be implicit (e.g., power domain) or explicit (for example,

check_timing

connect_supply_net).

unconstrained_endpoints

Warns about unconstrained timing endpoints. This warning identifies timing endpoints (output ports and register data pins) that are not constrained for maximum delay (setup) checks. If the endpoint is a register data pin, it can be constrained by using **create_clock** for the appropriate clock source. You can constrain output ports using the **set_output_delay** or **set_max_delay** commands.

unexpandable_clocks

Warns if there are sets of clocks for which periods are not expandable with respect to each other. The checking is only done for the related clock domains, such as ones where there is at least one path from one clock domain to the other. This could be because of an incorrectly defined clock period for one or more of the clocks. Another possibility is when asynchronous clocks with unexpandable periods are interacting where they should have been defined in different clock domains.

pulse_clock_non_pulse_clock_merge

Warns if pulse clock and normal clock propagate to the same network.

pulse_clock_no_pulse_generator

Warns if the pulse clock constraints cannot be honored. This could be because pulse clock constraints have been set on clocks that do not drive pulse generators or the design may not have any pulse generators.

EXAMPLES

The following example checks timing of the current design and shows summary information of potential problems. The checks performed are those defined by **timing_check_defaults**.

```
pt_shell> check_timing
```

Information: Checking 'no_clock'.

Warning: There are 4 register clock pins with no clock.

Information: Checking 'no_input_delay'.

Information: Checking 'unconstrained_endpoints'.

Warning: There are 10 endpoints which are not constrained for maximum delay.

Information: Checking 'generic'.

Information: Checking 'latch_fanout'.

Warning: There are 2 level-sensitive latches which fanout to themselves.

Information: There are 2 level-sensitive latches which fanout to latches of the same clock.

Information: Checking 'loops'.

Warning: There are 6 timing loops in the design.

Information: Checking 'generated_clocks'.

The following example adds the **clock_crossing** check to the list of checks in **timing_check_defaults**.

```
pt_shell> check_timing -include {clock_crossing}
```

Information: Checking 'no_clock'.

Warning: There are 4 register clock pins with no clock.

Information: Checking 'no_input_delay'.

Information: Checking 'unconstrained_endpoints'.

Warning: There are 10 endpoints which are not constrained for maximum delay.

Information: Checking 'generic'.

Information: Checking 'latch_fanout'.

Warning: There are 2 level-sensitive latches which fanout to themselves.

Information: There are 2 level-sensitive latches which fanout to latches of the same clock.

Information: Checking 'loops'.

Warning: There are 6 timing loops in the design.

Information: Checking 'generated_clocks'.

Information: Checking 'clock_crossing'.

Information: Checking 'pll_configuration'.

The following example adds the **clock_crossing** check to the list of checks in **timing_check_defaults** and removes the **loops** check from the same list.

```
pt_shell> check_timing -include {clock_crossing} -exclude {loops}
```

Information: Checking 'no_clock'.

Warning: There are 4 register clock pins with no clock.

Information: Checking 'no_input_delay'.

Information: Checking 'unconstrained_endpoints'.

Warning: There are 10 endpoints which are not constrained for maximum delay.

Information: Checking 'generic'.

Information: Checking 'latch_fanout'.

Warning: There are 2 level-sensitive latches which fanout to themselves.

Information: There are 2 level-sensitive latches which fanout to latches of the same clock.

Information: Checking 'generated_clocks'.

Information: Checking 'clock_crossing'.

The following example removes the **retain** option from the list of checks in **timing_check_defaults**. Assuming that this check is not included in **timing_check_defaults**, there is nothing to remove. The output in this example is the same as running the command without the **-exclude** option.

```
pt_shell> check_timing -exclude {retain}
```

```
Information: Checking 'no_clock'.
```

```
Warning: There are 4 register clock pins with no clock.
```

```
Information: Checking 'no_input_delay'.
```

```
Information: Checking 'unconstrained_endpoints'.
```

```
Warning: There are 10 endpoints which are not constrained for maximum delay.
```

```
Information: Checking 'generic'.
```

```
Information: Checking 'latch_fanout'.
```

```
Warning: There are 2 level-sensitive latches which fanout to themselves.
```

```
Information: There are 2 level-sensitive latches which fanout to latches  
of the same clock.
```

```
Information: Checking 'loops'.
```

```
Warning: There are 6 timing loops in the design.
```

```
Information: Checking 'generated_clocks'.
```

The following example checks only latch fanout and shows detailed information.

```
pt_shell> check_timing -verbose -override_defaults {latch_fanout}
```

```
Warning: There are 2 level-sensitive latches which fanout to themselves.
```

```
Latch data pin
```

```
-----  
13/D
```

```
14/D
```

```
Information: There are 2 level-sensitive latches which fanout to latches  
of the same clock.
```

From Pin	To Pin	Clock	Level
11/G	12/D	C1	positive
11/G	13/D	C1	positive

SEE ALSO

```
create_clock(2)  
report_constraint(2)  
report_timing(2)  
set_clock_groups(2)  
set_case_analysis(2)
```

```
set_disable_timing(2)
set_false_path(2)
set_max_delay(2)
set_output_delay(2)
create_power_domain(2)
create_supply_net(2)
timing_check_defaults(3)
timing_enable_multiple_clocks_per_reg(3)
timing_input_port_default_clock(3)
report_default_significant_digits(3)
```

collections_and_querying

Describes the methodology for creating collections of objects and querying objects in the database.

DESCRIPTION

Synopsys applications build an internal database of the netlist and the attributes applied to it. This database consists of several classes of objects, including designs, libraries, ports, cells, nets, pins, and clocks. Most commands operate on these objects.

By definition a collection is a group of objects exported to the Tcl user interface.

Collections have an internal representation (the objects) and, sometimes, a string representation. The string representation is generally used only for error messages.

A set of commands to create and manipulate collections is provided as an integral part of the user interface. The collection commands encompass two categories: those that create collections of objects for use by another command, and one that queries objects for viewing. The result of a command that creates a collection is a Tcl object that can be passed along to another command. For a query command, although the visible output looks like a list of objects (a list of object names is displayed), the result is an empty string.

An empty string "" is equivalent to the empty collection, that is, a collection with zero elements.

Homogeneous and Heterogeneous Collections

A homogeneous collection contains only one type of object. A heterogeneous collection can contain more than one type of object. Commands that accept collections as arguments can accept either type of collection.

Lifetime of a Collection

Collections are active only as long as they are referenced. Typically, a collection is referenced when a variable is set to the result of a command that creates it or when it is passed as an argument to a command or a procedure. For example, if in PrimeTime you save a collection of ports:

```
pt_shell> set ports [get_ports *]
```

then either of the following two commands deletes the collection referenced by the *ports* variable:

```
pt_shell> unset ports  
pt_shell> set ports "value"
```

Collections can be implicitly deleted when they go out of scope. Collections go out of scope when the parent (or other antecedent) of the objects within the collection is deleted. For example, if our collection of ports is owned by a design, it is

implicitly deleted when the design that owns the ports is deleted. When a collection is implicitly deleted, the variable that referenced the collection still holds a string representation of the collection. However, this value is useless because the collection is gone, as illustrated in the following PrimeTime example:

```
pt_shell> current_design
{"TOP"}

pt_shell> set_ports [get_ports in*]
{"in0", "in1"}

pt_shell> remove_design TOP
Removing design 'TOP'...

pt_shell> query_objects $ports
Error: No such collection '_sel26' (SEL-001)
```

Iteration

To iterate over the objects in a collection, use the **foreach_in_collection** command. You cannot use the Tcl-supplied **foreach** iterator to iterate over the objects in a collection, because **foreach** requires a list; and a collection is not a list. In fact, if you use **foreach** on a collection, it will destroy the collection.

The arguments to **foreach_in_collection** are similar to those of **foreach**: an iterator variable, the collections over which to iterate, and the script to apply at each iteration. Note that unlike **foreach**, the **foreach_in_collection** command does not accept a list of iterator variables.

The following example is an iterative way to perform a query. For details, see the **foreach_in_collection** man page or the user guide.

```
pt_shell> \
foreach_in_collection s1 $collection {
    echo [get_object_name $s1]
}
```

Manipulating Collections

A variety of commands are provided to manipulate collections.

- **add_to_collection** - Takes a base collection and a list of element names or collections that you want to add to it. The base collection can be the empty collection. The result is a new collection. In addition, the **add_to_collection** command allows you to remove duplicate objects from the collection by using the **-unique** option.

- **remove_from_collection** - Takes a base collection and a list of element names or collections that you want to remove from it. For example, in PrimeTime:

```
pt_shell> set dports \
```



```
[remove_from_collection [all_inputs] CLK]
{"in1", "in2", "in3"}
```

- **compare_collections** - Verifies that two collections contain the same objects (optionally, in the same order). The result is "0" on success.
- **copy_collection** - Creates a new collection containing the same objects (in the same order) as a given collection. Not all collections can be copied.
- **index_collection** - Extracts a single object from a collection and creates a new collection containing that object. Not all collections can be indexed.
- **sizeof_collection** - Returns the number of objects in a collection.

Filtering

You can filter any collection by using the **filter_collection** command. It takes a base collection and creates a new collection that includes only those objects that match an expression.

Many of the commands that create collections support a **-filter** option, which allows objects to be filtered out before they are ever included in the collection. Frequently this is more efficient than filtering after they are included in the collection. The following example from PrimeTime filters out all leaf cells:

```
pt_shell> filter_collection \  
[get_cells *] "is_hierarchical == true"  
{"i1", "i2"}
```

The basic form of a filter expression is a series of relations joined together with AND and OR operators. Parentheses are also supported. The basic relation contrasts an attribute name with a value through a relational operator. In the previous example, *is_hierarchical* is the attribute, *==* is the relational operator, and *true* is the value.

The relational operators are

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

The basic relational rules are

- String attributes can be compared with any operator.

- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only true or false.

Additionally, existence relations determine if an attribute is defined or not defined, for the object. For example,

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are

```
defined
undefined
```

These operators apply to any attribute as long as it is valid for the object class. See the appropriate man pages for complete details.

Sorting Collections

You can sort a collection by using the **sort_collection** command. It takes a base collection and a list of attributes as sort keys. The result is a copy of the base collection sorted by the given keys. Sorting is ascending, by default, or descending when you specify the **-descending** option. In the following example, the PrimeTime command sorts the ports by direction and then by full name.

```
pt_shell> sort_collection [get_ports *] \
{direction full_name}
{"in1", "in2", "out1", "out2"}
```

Implicit Query of Collections

All commands that create implicitly collections query the collection when the command is used at the command line. The number of objects displayed is controlled by the **collection_result_display_limit** variable. Consider the following examples from PrimeTime:

```
pt_shell> set_input_delay 3.0 [get_ports in*]
1
pt_shell> get_ports in*
{"in0", "in1", "in2"}
pt_shell> query_objects -verbose [get_ports in*]
{"port:in0", "port:in1", "port:in2"}
pt_shell> set_iports [get_ports in*]
{"in0", "in1", "in2"}
```

In the first example, the **get_ports** command creates a collection of ports that is passed to the **set_input_delay** command. This collection is not the result of the primary command (**set_input_delay**), so it is not queried. The second example shows how a command that creates a collection automatically queries the collection when that command is used as a primary command. The third example shows the verbose

feature of the **query_objects** command, which is not available with implicit query. Finally, the fourth example sets the variable `iports` to the result of the **get_ports** command. Only in the final example does the collection persist to future commands until `iports` is overwritten, unset, or goes out of scope.

Controlling Deletion Effort

When a subset of objects in a design is removed, it is not always clear whether to remove the collection. The PrimeTime **collection_deletion_effort** variable controls the amount of effort expended to preserve collections. For complete details, see the **collection_deletion_effort** command man page.

Related Commands

For your convenience, related commands and variables are listed below by categories:

Common Commands:

- `add_to_collection`
- `compare_collections`
- `copy_collection`
- `foreach_in_collection`
- `index_collection`
- `query_objects`
- `remove_from_collection`
- `sizeof_collection`
- `filter_collection`
- `sort_collection`

PrimeTime Commands

- `all_clocks`
- `all_connected`
- `all_inputs`
- `all_instances`
- `all_outputs`
- `get_cells`
- `get_clocks`
- `get_designs`
- `get_generated_clocks`
- `get_lib_cells`
- `get_lib_pins`
- `get_libs`
- `get_nets`
- `get_path_groups`
- `get_pins`
- `get_ports`
- `get_qtm_ports`
- `get_timing_paths`

PrimeTime Variables:

- `collection_deletion_effort`

collection_result_display_limit

SEE ALSO

add_to_collection(2)
all_clocks(2)
all_connected(2)
all_inputs(2)
all_instances(2)
all_outputs(2)
compare_collections(2)
copy_collection(2)
filter_collection(2)
foreach_in_collection(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_generated_clocks(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)
get_nets(2)
get_path_groups(2)
get_pins(2)
get_ports(2)
get_qtm_ports(2)
get_timing_paths(2)
index_collection(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
sort_collection(2)
collection_deletion_effort(3)
collection_result_display_limit(3)

compare_collections

Compares the contents of two collections. If the same objects are in both collections, the result is 0 (like string compare). If they are different, the result is nonzero. The order of the objects can optionally be considered.

SYNTAX

```
int compare_collections
[-order_dependent] collection1 collection2
collection1           collection
collection2           collection
```

ARGUMENTS

`-order_dependent`

Indicates that the order of the objects is to be considered; that is, the collections are considered to be different if the objects are ordered differently.

`collection1`

Specifies the base collection for the comparison. The empty string (empty collection) is a legal value for the *collection1* argument.

`collection2`

Specifies the collection with which to compare to the *collection1* argument. The empty string (empty collection) is a legal value for the *collection2* argument.

DESCRIPTION

This command is used to compare the contents of two collections. By default, the order of the objects does not matter, so that a collection of cells u1 and u2 is the same as a collection of the cells u2 and u1. By using the `-order_dependent` option, the order of the objects is considered.

Either or both of the collections can be the empty string (empty collection). If two empty collections are compared, the comparison succeeds (that is, the **compare_collections** command considers them identical), and the result is 0.

EXAMPLES

The following example shows a variety of comparisons. Note that a result of 0 from the **compare_collections** command indicates success. Any other result indicates failure.

```
pt_shell> compare_collections [get_cells *] [get_cells *]
0
pt_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
pt_shell> set c2 [get_cells {u2 u1}]
{"u2", "u1"}
```

```
pt_shell> set c3 [get_cells {u2 u4 u6}]
{"u2", "u4", "u6"}
pt_shell> compare_collections $c1 $c2
0
pt_shell> compare_collections $c1 $c2 -order_dependent
-1
pt_shell> compare_collections $c1 $c3
-1
```

The following example builds on the previous example by showing how empty collections are compared.

```
pt_shell> set c4 ""
pt_shell> compare_collections $c1 $c4
-1
pt_shell> compare_collections $c4 $c4
0
```

SEE ALSO

`collections(2)`

compare_interface_timing

Compares two reports generated by the **write_interface_timing** command.

SYNTAX

```
int compare_interface_timing  
ref_timing_file  
cmp_timing_file  
[-output file_name]  
[-absolute_tolerance atol_list]  
[-percent_tolerance ptol_list]  
[-capacitance_tolerance ctol_list]  
[-noise_tolerance ntol_list]  
[-ignore ign_list]  
[-include cmp_list]  
[-quiet]  
[-nosplit]  
[-sort_by_worst]  
[-session session_name]  
[-significant_digits digits]
```

Data Types

<i>ref_timing_file</i>	string
<i>cmp_timing_file</i>	string
<i>file_name</i>	string
<i>session_name</i>	string
<i>atol_list</i>	list
<i>ptol_list</i>	list
<i>ctol_list</i>	list
<i>ntol_list</i>	list
<i>ign_list</i>	list
<i>cmp_list</i>	list
<i>session_name</i>	string
<i>digits</i>	int

ARGUMENTS

ref_timing_file
Specifies the name of the timing file to be used as the reference in the comparison.

cmp_timing_file
Specifies the name of the timing file to be compared to the reference timing file.

-output *file_name*
Specifies the name of the output file where the results of the comparison information is written. The information written to *file_name* specifies PASS/FAIL status for every parameter compared by the command.
By default, these results are written to the session transcript. All informational, warning, or error messages are still directed to the session

transcript, even if this option is specified.

-absolute_tolerance *atol_list*

Specifies the absolute error tolerance for time data. The default is zero. See below for a description of the tolerance format. If this option is used in conjunction with the **-percent_tolerance** option, both tolerances must be exceeded to cause a FAIL. See the **-capacitance_tolerance** and **-noise_tolerance** options for other absolute tolerances.

-percent_tolerance *ptol_list*

This option is similar to the **-absolute_tolerance** option except that it uses the percent relative error instead of an absolute error. The exception is that slack data ignores this option and uses only the absolute tolerance. The default is zero.

See below for a description of the tolerance format. If this option is specified in conjunction with either the **-absolute_tolerance**, **-capacitance_tolerance**, or **-noise_tolerance** option, both tolerances must be exceeded to cause a FAIL.

-capacitance_tolerance *ctol_list*

Specifies the absolute error tolerance for capacitance data. The default is zero.

See below for a description of the tolerance format. If this option is used in conjunction with the **-percent_tolerance** option, both tolerances must be exceeded to cause a FAIL. See the **-absolute_tolerance** and **-noise_tolerance** options for other absolute tolerances.

-noise_tolerance *ntol_list*

Specifies the absolute error tolerance for noise slack. The default is zero. See below for a description of the tolerance format. If this option is used in conjunction with the **-percent_tolerance** option, both tolerances must be exceeded to cause a FAIL. See **-absolute_tolerance** and **-capacitance_tolerance** for other absolute tolerances.

-ignore *ign_list*

Specifies a list of categories of slack or arc data to be excluded from the comparison. The following list describes the valid values:

max – Excludes all *max_rise* and *max_fall* delay type arcs from the slack and *transition_time* sections. The capacitance section is not examined because it contains only maximum information.

min – Excludes all *min_rise* and *min_fall* delay type arcs from the slack and *transition_time* sections.

unconstrained – Excludes all paths that are unconstrained for both reference and compare data files.

pass – Excludes all passing comparisons, so only fails are displayed.

async_default – Excludes the comparison of paths in the *async_default* path group. This option is obsolete and is replaced by specifying the *recover* and *removal* values of the **-include** option.

clock_gating – Excludes comparison of paths with the *clock_gating_default* path group. This option is obsolete and is replaced by specifying the *clock_gating_setup* and *clock_gating_hold* of the **-include** option.

input_to_register – Excludes comparison of input-to-register paths. This option is obsolete and is replaced by specifying the *setup* and *hold* values of the **-include** option.

register_to_output – Excludes comparison of register-to-output path. This

option is obsolete and is replaced by specifying the *max_seq_delay* and *min_seq_delay* of the *-include* option.

input_to_output – Excludes comparison of combinational paths. This option is obsolete and is replaced by specifying the *max_combo_delay* and *min_combo_delay* of the *-include* option.

-include *cmp_list*

Specifies a list of data sections from the **write_interface_timing** command output to be included in the comparison; only those sections are compared. By default, all sections are compared. The following list describes the keywords that are accepted in the *cmp_list* as valid:

slack – Specifies that only worst-case slacks are to be compared.

arc – Specifies that only arc values are to be compared.

transition_time – Specifies that only actual transition times on ports are to be compared.

capacitance – Specifies that only actual capacitances on ports are to be compared.

design_rules – Specifies that only design rules on ports are to be compared.

missing_arcs – Specifies that only arcs in the reference timing file that are missing from the compare timing file are to be included in the report.

max_combo_delay – Specifies that only maximum combination delay arc type paths are to be reported.

min_combo_delay – Specifies that only minimum combination delay arc type paths are to be reported.

max_seq_delay – Specifies that only maximum sequential delay arc type paths are to be reported.

min_seq_delay – Specifies that only minimum sequential delay arc type paths are to be reported.

setup – Specifies that only setup arc type paths are to be reported.

hold – Specifies that only hold arc type paths are to be reported.

recovery – Specifies that only recovery arc type paths are to be reported.

removal – Specifies that only removal arc type paths are to be reported.

clock_gating_setup – Specifies that only clock gating setup arc type paths are to be reported.

clock_gating_hold – Specifies that only clock gating hold arc type paths are to be reported.

noise – Specifies that the noise section should be included.

-quiet

Specifies an obsolete option. This option is now obsolete and has been replaced by the *-session quiet* option.

-session *session_name*

Controls the information that is printed to the session transcript. Note that this option affects only the session transcript, not the information written to the output file. The following list describes the keywords that are accepted for *session_name*:

quiet – Prevents information, warning, and error messages from appearing in the session transcript. Use this option when you want to see only the success/failure return code without any MV messages. The **-session** option used with this keyword replaces the now-obsolete **-quiet** option.

summary – Prints only the summary pass/fail count to the session transcript.

full – Prints the detailed report either to the session transcript or to a file if you specify the *-output* option. This is the default value.

`-nosplit`

Prevents line-splitting. This is most useful for performing difference comparisons on previous scripts or for post processing the script.

`-sort_by_worst`

Specifies that the output is to be sorted from negative to positive, with the worst failure first. After the FAIL section, all PASS lines are sorted alphabetically within each path attribute. By default, the output follows the order of the reference `write_interface_timing` data, which is sorted alphabetically within each path attribute (c paths, i paths including a and g, then o paths).

`-significant_digits digits`

Specifies the number of digits to the right of the decimal point that are to be reported following the method used in the **report_timing** command. Allowed values are 0-13. The default value is 2. Use this option if you want to override the default. See below for a more detailed description of significant digits.

DESCRIPTION

This command compares two interface timing files (called the "reference" and "comparison" files) previously generated by the **write_interface_timing** command. The result is PASS if the timing parameter values in the two files are the same or within a specified tolerance. The result is FAIL if both columns have data and the tolerance is exceeded. If all comparisons in the report are PASS, the return code for the **compare_interface_timing** command is 0. If one or more comparisons result in FAIL, the return code is 1. Any other return code (or no return code) indicates a command error. If either file format does not conform to the **write_interface_timing** standard, the command issues a warning message.

The **compare_interface_timing** command lets you specify the input and output files for the comparison, the types of paths and timing parameters to compare or not compare, and the allowed tolerance levels that trigger comparison failures.

If the **timing_slew_propagation_mode** variable was set to *worst_arrival* when the **write_interface_timing** command was run on the reports being compared, this variable is expected to be in the same state when running the **compare_interface_timing** command. Otherwise, some combinational paths might not be properly compared.

If the reference and comparison files were generated using different contexts then the two files may contain different timing arcs along with different slacks. If an arc exists in the comparison file but not in the reference file, the **compare_interface_timing** command ignores the extra arcs. If an arc exists in the comparison file but not in the reference file, the **compare_interface_timing** command reports the slack value for the reference file, but either "N.C." or "----" for the comparison file slack. The slack difference is marked as "----", and the status is FAIL. The comparison slack is marked as "----" if the arc is not clock gating or recovery or removal.

The slack for a missing arc is mark as "N.C.", meaning not critical, if it exists in the reference but not in the comparison file, and the arc type is clock gating or recovery or removal. These arcs fall into the ****clock_gating_default**** or ****asynchronous_default**** clock groups, respectively. Each of these groups can have

several clocks in them, and under certain circumstances the **write_interface_timing** command sees the path to one clock as critical for the reference view and report that arc and slack, but another clock as critical in the comparison view and report that arc and slack.

The **compare_interface_timing** command deals with significant digits in two distinct ways. First, the internal computations are limited to the minimum precision of the two input files. This means that if the *ref_timing_file* was generated with three digits and the *cmp_timing_file* with four, the **compare_interface_timing** command uses three digits for all internal computations.

It is important to make the tolerances larger than a unit of the computational significant digits, or the **compare_interface_timing** command could generate false FAIL and PASS messages. This can happen because of rounding. For example, if the internal significant digits is two, and the absolute tolerance is set to 0.01, a difference of 0.014 is rounded to 0.01 and reported as a PASS even though it is outside the tolerance. Similarly, if the absolute tolerance is set to 0.009, a difference of 0.009 is rounded to 0.01 and reported as FAIL even though it is within tolerance. You should ensure that the number of digits of accuracy of the input files is greater than the accuracy of the tolerances.

A second use of significant digits is in report generation, which is limited to the minimum of the input precision and the significant digits that you specified. You specify the significant digits for the report through the *-significant_digits* option. For example, if the value for the *-significant_digits* option is five but the *ref_timing_file* was generated with four digits, the report is limited to four digits. Note that this use of significant digits does not affect the PASS or FAIL status, but a reported difference could look like it should have a different status if the difference is close to a tolerances and the difference rounded when writing the report.

Each tolerance is a list of either one or two floating point numbers. All tolerances are inclusive, and the sign of the values has no effect. If there is only one number, it serves to specify both the plus and minus tolerances. The plus tolerance is the absolute value of the number, and the minus tolerance is the inverse of the plus. For example, *atol_list* equal to 0.1 indicates that time differences of less than 0.1 and greater than -0.1 are considered to be passing.

If there are two numbers, the first specifies the minus bound and the second the plus. The minus tolerance is the inverse of the absolute value of the first number, and the plus tolerance is the absolute value of the second number. Having a pair of numbers allows different tolerances to be applied for negative and positive errors, which can be interpreted as optimism and pessimism depending on the arc type. For example, an *atol_list* equal to {0.2 -0.3} indicates that data differences of less than 0.3 and greater than -0.2 are considered passing.

EXAMPLES

The following example shows a variety of comparisons. Each item in parentheses (for example, *L1*, *L2*) represents the latch level associated with the slack number directly to the left. These values are included in the report wherever there is an *L#* attribute in the **write_interface_timing** output. At the end of the report is a summary of the total comparisons passed and failed broken down by data section.

```
pt-shell> compare_interface_timing demo_net.rpt demo_etm.rpt
? -include arc -percent_tolerance {10 5}
```

```
*****
```

```
Command: compare_interface_timing
         demo_net.rpt demo_etm.rpt
         -include arc
         -percent_tol 10.0 5.0
```

```
Design : top
```

```
Version: 2002.03-SI1
```

```
Date   : Mon Nov 26 11:20:01 2001
```

```
*****
```

From	To	Arc Type	Arc Value		Cmp	%Error	Status
			Ref				
in(r)	out(r)	max_combo_delay	2.63		2.21	15.97	FAIL
in(f)	out(r)	max_combo_delay	2.28		2.32	-1.75	PASS
in(r)	out(f)	min_combo_delay	0.17		0.30	-76.47	FAIL
in(f)	out(f)	min_combo_delay	0.51		0.51	0.00	PASS
in(r)	clk(r)	setup	2.63	(L1)	2.63	0.00	PASS
in(f)	clk(r)	setup	2.29	(L2)	2.29	0.00	PASS
in(r)	clk(f)	hold	0.17		0.17	0.00	PASS
clk(r)	out(r)	max_seq_delay	17.79		17.79	0.00	PASS
clk(r)	out(f)	max_seq_delay	18.20		18.20	0.00	PASS
clk(f)	out(r)	min_seq_delay	2.21		2.21	0.00	PASS
clk(f)	out(f)	min_seq_delay	1.80		1.80	0.00	PASS

	Totals	Arc Value	Transition Time	Capacitance	Rules
Passed	9	9	0	0	-
Failed	2	2	0	0	0
Total	12	12	0	0	0

SEE ALSO

```
write_interface_timing(2)
timing_slew_propagation_mode(3)
```

complete_net_parasitics

Completes partial parasitics annotated on all nets of the current design.

SYNTAX

```
string complete_net_parasitics  
[-complete_with completion_type]
```

Data Types

completion_type string

ARGUMENTS

-complete_with *completion_type*
Indicates that a net with partially annotated parasitics and missing segments is to be completed by inserting capacitances and resistances according to the *completion_type* value. Allowed values are *zero* (the default), which completes the net by inserting zero capacitances and resistances, and *wlm*, which completes the net by inserting capacitances and resistances derived from wire load models.
This option is equivalent to the **read_parasitics -complete_with** command.

DESCRIPTION

This command completes all nets in the design that have incomplete RC networks annotated from SPEF or SPF files.

If lumped parasitics (set using the **set_load** or **set_resistance** command) already exist on any of the nets, they are not overwritten; however, a warning message is issued.

Note: The **complete_net_parasitics** and **read_parasitics -complete_with** commands complete a net only if all missing segments are between two pins, and only if the nets are partially annotated. Nets are not affected if they are fully annotated or have no annotation at all. Also, the net must be hierarchical, so that if the parasitics for the block-level parts of a net are missing, those parasitics could exist in the top-level net. If any of these conditions are not met, you must manually correct the SPEF or DSPF file.

Use the **report_annotated_parasitics** command to view how the parasitics are completed.

After the completion of the nets, there is no direct way to remove only the completed segments. You can remove all parasitics read and annotated with the **read_parasitics** and **complete_net_parasitics** commands using the **remove_annotated_parasitics** command. You then reread the previously annotated parasitics using the **read_parasitics** command. The **reset_design** command removes all attributes from the current design, including annotated parasitics.

EXAMPLES

The following example completes the partially annotated parasitics from the file `spefffile.spef`, by inserting zero capacitances and resistances.

```
pt_shell> read_parasitics spefffile.spef  
pt_shell> complete_net_parasitics -complete_with zero
```

SEE ALSO

```
read_parasitics(2)  
remove_annotated_parasitics(2)  
report_annotated_parasitics(2)  
reset_design(2)
```

connect_net

Connects a net to specified pins or ports.

SYNTAX

```
int connect_net  
net  
object_spec
```

Data Types

<i>net</i>	string
<i>object_spec</i>	list

ARGUMENTS

net
Specifies the name of the net to which the pins and ports are to be connected.

object_spec
Specifies a list of pins or ports to connect to *net*.

DESCRIPTION

Connects pins and ports to a *net* in the current design. Like all other netlist editing commands, for the **connect_net** command to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. The **connect_net** command returns a 1 if successful and a 0 if unsuccessful.

The *net* and each pin and port specified in the *object_spec* must be in scope; that is, at or below the current instance.

There are two rules for the **connect_net** command:

- You cannot connect *net* to a pin that is already connected.
- You cannot connect across a hierarchical boundary. For example, you cannot connect a port to a net at a hierarchical level other than the top of the design.

EXAMPLES

The following example creates a cell and a net and then connects the new net to the output of the new cell:

```
pt_shell> create_net new_net1
```

Information: Created net 'new_net1' in design 'top'. (NED-016)
1

pt_shell> **create_cell u1 class/AN2**
Information: Created cell 'u1' in design 'top'. (NED-014)
1

pt_shell> **connect_net new_net1 u1/Z**
Information: Connected 'u1/Z' to 'new_net1'. (NED-018)
1

SEE ALSO

create_cell(2)
create_net(2)
disconnect_net(2)
size_cell(2)
swap_cell(2)
write_changes(2)

connect_power_domain

Connects power net information for the specified power domains.

SYNTAX

```
int connect_power_domain  
power_domains  
[-primary_power_net power_net]  
[-primary_ground_net ground_net]  
[-backup_power_net power_net]  
[-backup_ground_net ground_net]  
[-internal_power_net internal_power_net]  
[-internal_ground_net internal_ground_net]
```

Data Types

<i>power_domains</i>	collection
<i>power_net</i>	string
<i>ground_net</i>	string
<i>internal_ground_net</i>	string
<i>internal_ground_net</i>	string

ARGUMENTS

power_domains
Specifies the power domains for which to make the power net information connections.

-primary_power_net power_net
Specifies the primary power net information for the power domains.

-primary_ground_net ground_net
Specifies the primary ground net information for the power domains.

-backup_power_net power_net
Specifies the backup power net information for the power domains.

-backup_ground_net ground_net
Specifies the backup ground net information for the power domains.

-internal_power_net internal_power_net
Specifies the internal power net information for the power domains.

-internal_ground_net internal_ground_net
Specifies the internal ground net information for the power domains.

DESCRIPTION

The **connect_power_domain** command creates logical power connections for the specified power domains. All cells in the power domain inherit the power connections. You can override the inherited power connections for a cell by using the

connect_power_net_info command.

The primary power and ground nets are used to define the main power connections for the power domain.

The backup power and ground nets are used to define power connections for always-on logic, retention registers, isolation cells, and enable-level shifter cells.

The internal power and ground nets are used to connect power nets to the switching cells present inside the power domain.

EXAMPLE

The following example connects power net information to the TOP_DOMAIN and SUB_DOMAIN power domains.

```
prompt> connect_power_domain TOP_DOMAIN \  
-primary_power_net T_VDD \  
-primary_ground_net T_VSS  
1  
prompt> connect_power_domain SUB_DOMAIN \  
-primary_power_net A_VDD \  
-primary_ground_net A_VSS \  
-backup_power_net VDD_Backup \  
-backup_ground_net VSS_Backup  
1
```

SEE ALSO

```
create_power_domain (2)  
get_power_domains (2)  
report_power_domain (2)  
connect_power_net_info (2)
```

connect_power_net_info

Connects the specified power net information to the specified power pins.

SYNTAX

```
int connect_power_net_info  
object_list  
-power_pin_name power_pin_name  
-power_net_name power_net_name
```

Data Types

<i>object_list</i>	list
<i>power_pin_name</i>	string
<i>power_net_name</i>	string

ARGUMENT

```
object_list  
    Specifies the leaf cells for which the power net information connections are  
    made.  
  
-power_pin_name power_pin_name  
    Specifies the name of the power pin.  
  
-power_net_name power_net_name  
    Specifies the name of the power net.
```

DESCRIPTION

The **connect_power_net_info** command makes power net information connections for a specific power pin of a leaf cell. The pin-level connection overrides the domain-level connections made with the **connect_power_domain** command.

See **report_power_pin_info** for more information.

EXAMPLE

The following example connects the power net information for the PWR pin of the PD0_INST/I0 cell.

```
prompt> connect_power_net_info PD0_INST/I0 \  
    -power_pin_name PWR -power_net_name exp_VDD  
1
```

SEE ALSO

```
connect_power_domain (2)  
report_power_pin_info (2)
```

connect_supply_net

Connect the `supply_net` to specified `supply_ports/pins`. The command is part of UPF definition of virtual power and ground network.

SYNTAX for UPF mode

```
int connect_supply_net
```

```
supply_net_name  
[-ports list]
```

Data Types

<i>supply_net_name</i>	string
<i>list</i>	list

ARGUMENTS

`supply_net_name`
Specify the name of the `supply_net` to be connected.
The net must exist in the current scope.

`-ports list`
Specify which `supply_ports` to be connected with the `supply_net`. The name is hierarchical name.

DESCRIPTION for UPF mode

Specify which supply ports or instance power ground pins are to be connected with the `supply_net`.

DESCRIPTION

This command provides an explicit connect of a `supply_net` to any `supply_ports/pins` and overrides (has higher precedence than) the auto-connection semantics that might otherwise apply. If a design element is not connected explicitly to any `supply_net` using the **connect_supply_net** command, it shares the primary power/ground `supply_net` with the `power_domain` it belongs to.

The instance that contains the pin must also be in the extent of the same `power_domain`. A supply port cannot connect to two nets in the same domain. Note that the command does not specify whether the net connects to the inside or outside side of the port. The port side is auto-derived from the net domain and port scope.

This command returns 1 if it succeeded; otherwise, it returns 0.

EXAMPLES for UPF mode

The following example connects a `supply_net` VSS with `supply_port` VSS and ground pin

```
connect_supply_net
```

of cell INST_1/u1:

```
prompt> create_power_domain PD1 -elements INST_1
PD1
prompt> create_supply_net VSS -domain PD1
VSS
prompt> create_supply_port VSS -domain PD1
VSS
prompt> connect_supply_net VSS -ports VSS
1
prompt> connect_supply_net VSS -ports INST_1/u1/GND
1
```

SEE ALSO for UPF mode

```
create_supply_net(2)
report_supply_net(2)
create_supply_port(2)
create_power_domain(2)
get_supply_netst(2)
get_supply_ports(2)
```

copy_collection

Duplicates the contents of a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection copy_collection collection1  
collection collection1
```

ARGUMENTS

```
collection1  
    Specifies the collection to be copied. If the empty string is used for the  
    collection1 argument, the command returns the empty string (a copy of the  
    empty collection is the empty collection).
```

DESCRIPTION

The **copy_collection** command is an efficient mechanism for creating a duplicate of an existing collection. It is more efficient, and almost always sufficient, to simply have more than one variable referencing the same collection. For example, if you create a collection and save a reference to it in a variable **c1**, assigning the value of **c1** to another variable **c2** simply creates a second reference to the same collection:

```
pt_shell> set c1 [get_cells "U1*"]  
{"U1", "U10", "U11", "U12"}  
pt_shell> set c2 $c1  
{"U1", "U10", "U11", "U12"}
```

This has not copied the collection. There are now two references to the same collection. If you change the *c1* variable, *c2* continues to reference the same collection:

```
pt_shell> set c1 [get_cells "block1"]  
{"block1"}  
pt_shell> query_objects $c2  
{"U1", "U10", "U11", "U12"}
```

There may be instances when you really do need a copy, and in those cases, **copy_collection** is used to create a new collection that is a duplicate of the original.

EXAMPLES

The following example shows the result of copying a collection. Functionally, it is not much different than having multiple references to the same collection.

```
pt_shell> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
pt_shell> set c2 [copy_collection $c1]
{"U1", "U10", "U11", "U12"}
pt_shell> unset c1
pt_shell> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

SEE ALSO

collections (2).

cputime

Retrieves the overall user time associated with the current `pt_shell` process.

SYNTAX

```
float cputime  
[format]
```

Data Types

format string

ARGUMENTS

`format`
This argument takes a *print* style formatting string for a floating point number.

DESCRIPTION

This command returns the accumulated user time, in seconds, for the current `pt_shell` process. If the *format* string is omitted, an integer number of seconds is returned. If the *format* string is specified, a floating point number is returned. The number includes fractions of a second elapsed and is formatted in accordance with given specifier. For a detailed description of how to format a floating point argument, see the **print** UNIX man page.

```
pt_shell> cputime "%g"  
3.74
```

SEE ALSO

`mem(2)`

create_activity_waveforms

Creates activity waveforms from VCD.

Note: This command has been superceeded by the **write_activity_waveforms** command.

SEE ALSO

`write_activity_waveforms(2)`
`write_power_waveforms(2)`

create_cell

Creates cells in the current design.

SYNTAX

```
int create_cell
```

```
[-libraries lib_spec]
```

```
[-exact]
```

```
cell_list
```

```
lib_cell
```

Data Types

```
cell_list          list
```

```
lib_cell           string
```

ARGUMENTS

-libraries *lib_spec*

If this option is specified, PrimeTime resolves the *lib_cellP* option from the libraries contained in only the *lib_spec* option. Libraries are searched in the order in which they appear in *lib_spec*. The *lib_spec* value can be a list of library names or collections of libraries loaded into PrimeTime. You can obtain the latter by using the **get_libs** command. You cannot specify this option if a full library cell name has been specified.

-exact

Indicates that names are to be used exactly as specified. Use this option if you want to create a cell that contains a hierarchy or wildcard character as part of the cell name. For more information, see the Naming Cells With Special Characters section.

cell_list

Specifies a list of cells to be created.

lib_cell

Specifies the name of the library cell to which the specified cells are to be linked. The *lib_cell* option can be a library cell object or the name of a library cell. You can obtain the former by using the **get_lib_cells** command. The latter can either be the full library cell name, such as '*lib_name/lcell_name*', or the just the base name of the library cell, such as '*lcell_name*'. You cannot specify the **-libraries** option and explicitly specify the full library cell name. If you invoke PrimeTime with the **-multi_scenario** option, only the library cell base name must be used. For more information, see the Resolved Library Cells section.

DESCRIPTION

This command creates new cells in the current design. Like all other netlist editing commands, for the **create_cell** command to succeed, all of its arguments must succeed.

create_cell

If any arguments fail, the netlist remains unchanged. The **create_cell** command returns a 1 if successful and a 0 if unsuccessful.

Cells are created in scope; that is, at or below the current instance. Cell names are specified as for other commands, using the full hierarchical name relative to the current instance, as in the following example:

```
create_cell u1/u2/u3 class/AN2
```

This command attempts to create a cell named u3 in the hierarchical block u1/u2. The name to the right of the last hierarchical separator is the actual cell name, and the remainder is sent to a *search engine* to find a hierarchical block in which to create the new cell. The operation fails if any of the following are true:

- The search engine cannot find "u1/u2".
- The search engine finds multiple blocks that match "u1/u2".
- The search engine finds a leaf cell matching "u1/u2".

Currently, you cannot create new hierarchy; that is, you cannot create a cell that is an instance of a design. The *lib_cell* must be a leaf library cell.

After creating a cell, you can connect nets to its pins with the **connect_net** command. You can remove cells with the **remove_cell** command.

Resolving Library Cells

If the *lib_cell* has been specified in base name only format, such as without a library from which to resolve it from, PrimeTime resolves the library cell according to the following methodology.

If the *-libraries* option is specified, PrimeTime searches for library cells only in the libraries contained within the *lib_spec*.

Alternatively, if the above option is not specified, PrimeTime searches for the library cell in the libraries contained within the **link_path** variable.

The first library cell that is found is used.

Naming Cells With Special Characters

If you want to create a cell whose name contains the current hierarchical separator or wildcard characters used by the search engine, you must do the following:

- Use the **current_instance** command to change the scope to the block in which you want the new cell created.
- Use the **create_cell -exact** command to create the cell.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the changelist can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following example, cells are created in the current instance, and at a level below the current instance. Currently, u1 is an instance of a design that has multiple instances. Therefore, it is uniquified as part of the editing operation.

```
pt_shell> create_cell t1 class/AN2
Information: Created cell 't1' in design 'top'. (NED-014)
1
```

```
pt_shell> create_cell u1/t1 class/AN2
Uniquifying 'u1' (block1) as 'block1_0'.
Information: Created cell 't1' in 'top/u1'. (NED-014)
1
```

The following example creates cell a/b in the hierarchical block u1/u2. The first attempt fails, because the *exact* option was omitted. Here, u1/u2 is an instance of a design that has multiple instances; therefore, it is uniquified as part of the editing operation. The u1 was already uniquified in the previous example.

```
pt_shell> current_instance u1/u2
u1/u2

pt_shell> create_cell a/b class/AN2
Error: Could not create cell 'a/b':
      a not found. (NED-041)
Error: No changes made. (NED-040)
0
```

```
pt_shell> create_cell a/b class/AN2 -exact
Uniquifying 'u1/u2' (block2) as 'block2_0'.
Information: Created cell 'a/b' in 'top/u1/u2'. (NED-014)
1
```

In the following example, multiple cells are created using library cells chosen from user-specified libraries using only the library cell base name. The 'lib1' library is searched first as it appears in the *lib_spec* list first and then 'lib2' is searched.

```
pt_shell> create_cell -libraries {lib1 lib2} {u1 u2 u3} ND2
Information: Created cell 'u1' in design 'middle' with 'lib1/ND2'. (NED-014)
Information: Created cell 'u2' in design 'middle' with 'lib1/ND2'. (NED-014)
Information: Created cell 'u3' in design 'middle' with 'lib1/ND2'. (NED-014)
1
```

SEE ALSO

```
connect_net(2)
create_net(2)
current_instance(2)
remove_cell(2)
size_cell(2)
swap_cell(2)
write_changes(2)
link_path(3)
```

create_clock

Creates a clock object.

SYNTAX

```
string create_clock  
-period period_value  
[-name clock_name]  
[-waveform edge_list]  
[-add]  
[source_objects]
```

Data Types

<i>period_value</i>	float
<i>clock_name</i>	string
<i>edge_list</i>	list
<i>source_objects</i>	list

ARGUMENTS

-period *period_value*

Specifies the clock period in library time units. This is the minimum time over which the clock waveform repeats. The *period_value* type must be greater than or equal to zero.

-name *clock_name*

Specifies the name of the clock being created, enclosed in quotation marks. If you do not use this option, the clock gets the same name as the first clock source specified in the *source_objects* option. If you did not specify the *source_objects* option, you must use the -name option, which creates a virtual clock not associated with a port or pin. You can use both the -name and *source_objects* options to give the clock a more descriptive name than the first source pin or port. If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.

-waveform *edge_list*

Specifies the rise and fall edge times of the clock waveforms of the clock, in library time units, over an entire clock period. The first time in the *edge_list* is a rising transition, typically the first rising transition after time zero. There must be an even number of edges, and they are assumed to be alternating rise and fall. The edges must be monotonically increasing, except for a special case with two edges, where fall can be less than rise. The numbers should represent one full clock period. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of *period_value*/2.

-add

Specifies whether to add this clock to the existing clock or to overwrite it. Use this option to capture the case where multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you specify this option, you must also use the -name option.

Defining multiple clocks on the same source pin or port causes longer runtime and higher memory usage than a single clock, because PrimeTime must explore all possible combinations of launch and capture clocks. Use the **set_false_path** command to disable unwanted clock combinations.

source_objects

Specifies the objects used as sources of the clock. The sources can be ports, pins, or nets in the design. If you do not use this option, you must use the **-name** option, which creates a virtual clock not associated with a port, pin, or net. If you specify a clock on a pin that already has a clock, the new clock replaces the old one unless you use the **-add** option. When a net is used as the source, the first driver pin of the net is the actual source used in creating the clock.

DESCRIPTION

Creates a clock object. It is created in the current design and is applied to the specified *source_objects* value. If you do not specify a *source_objects* value, but give a *clock_name* value, a virtual clock is created. You can create a virtual clock to represent an off-chip clock for input or output delay specification. For more information about input and output delay, see the **set_input_delay** and **set_output_delay** man pages.

If one of the *source_objects* values is already the source of a clock, the source is removed from that clock. This clock is eliminated if it has just one source.

The **create_clock** command also defines the waveform for the clock. The clock can have multiple pulses per period.

The **create_clock** command is used together with the **set_clock_latency**, **set_clock_uncertainty**, **set_propagated_clock**, and **set_clock_transition** commands to specify properties of clock networks. By default, a new path group is created for the clock. This new path group brings together the endpoints related to this clock for cost function calculation. To remove the clock from its assigned group, use the **group_path** command to reassign the clock to another group or to the default path group. For more information, see the **group_path** man page.

The new clock has ideal clock latency and transition time; no propagated delay through the clock network is assumed and a transition time of zero is used at the clock source pin. To enable propagated latency for a clock network, use the **set_propagated_clock** command. To set an estimated latency, use the **set_clock_latency** command.

To show information about clocks in the design, use the **report_clock** command. To create a collection of clocks matching a pattern and optionally matching filter criteria, use the **get_clocks** command.

To undo **create_clock**, use the **remove_clock** command.

EXAMPLES

The following example creates a clock on a port named *PHI1* with a period of *10.0*, a rise at *5.0*, and a fall at *9.5*.

create_clock


```
pt_shell> create_clock PHI1 -period 10 -waveform { 5.0 9.5 }
```

The following example shows a clock named *PHI2* with a falling edge at 5 and a rising edge at 10 with a period of 10. Because the edges for the *-waveform* option should be ordered as first rise, then fall, and should increase in value, the fall edge can be given as 15. This places the next falling edge after the first rise edge at 10.

```
pt_shell> create_clock PHI2 -period 10 -waveform { 10 15 }
```

The following example creates a virtual clock *PHI2* with a period of 10.0, a rise at 0.0, and a fall at 5.0.

```
pt_shell> create_clock -name PHI2 -period 10 -waveform {0.0 5.0}
```

The following example creates a clock named *clk2* with multiple sources.

```
pt_shell> create_clock -name clk2 -period 10 \  
-waveform {2.0 4.0} {clkgen1/Z clkgen2/Z clkgen3/Z}
```

The following example creates a clock named *CLK* on pin *u13/Z* with a period of 25, a fall at 0.0, a rise at 5.0, a fall at 10.0, a rise at 15.0, and so on.

```
pt_shell> create_clock u13/Z -name CLK -period 25 -waveform { 5 10 15 25}
```

SEE ALSO

```
all_clocks(2)  
get_clocks(2)  
group_path(2)  
remove_clock(2)  
report_clock(2)  
set_clock_latency(2)  
set_clock_uncertainty(2)  
set_input_delay(2)  
set_output_delay(2)  
set_propagated_clock(2)  
set_clock_transition(2)
```

create_correlation

Creates a new correlation type.

SYNTAX

```
int create_correlation  
-name correlation_name  
[-constant constant_value |  
-cross_correlations {float_list} |  
-physical_distance {float_list}]
```

Data Types

<i>correlation_name</i>	string
<i>constant_value</i>	float
<i>float_list</i>	list

ARGUMENTS

-name *correlation_name*
Defines a name for this correlation.

-constant *<constant_value>*
Supported values are 0 (null auto-correlation), 1 (full auto-correlation), or a number between 0 and 1 (partial auto-correlation). A constant value of 0 means that the instances of the variation are statistically independent. A constant value of 1 means that the instances of the variation are all equal. For partial auto-correlation, it is assumed that the variation has a normal distribution. The constant value is the auto-correlation value between two instances of the variation.
This option is incompatible with the *-cross_correlations* and *-physical_distance* options.

-cross_correlations *{float_list}*
This option models cross-correlation. For example, the correlation between different variations (as opposed to the correlation between instances of the same variation, which is called autocorrelation). The variations are correlated with respect to a single timing object. For example, the variations for different timing objects are uncorrelated.
The sequence of numbers denote the lower-triangular part of the correlation matrix. These numbers must be strictly between -1.0 and +1.0. If the correlation between two random variables is 1, the two random variables must be equal. If the correlation is -1, one variable is equal to minus the other. These are uninteresting cases.
If (A, B, C, D) are four variations, the list is {corr(B,A) corr(C,A) corr(C,B) corr(D,A) corr(D,B) corr(D,C)}. The list of variations is set through the **set_variation_correlation** command. The number of variations must correspond to the number of cross-correlation terms entered. Also importantly, the covariance matrix must be positive definite. Otherwise, there is either a linear relationship between the variations (and one should consider the minimum number of variations for the analysis), or the estimated correlation values are inaccurate enough (causing the covariance matrix to

be inapplicable).

It is assumed that the vector of variations has a multivariate normal distribution - each variation should have a normal distribution.

-physical_distance {float_list}

This option models spatial correlation. An auto-correlation function is provided in the list. This list is of the form (d0 rho0 d1 rho1 ...), where rhoi is the value of the correlation function at di. Distances must be in nanometer unit (for example, 1,000,000 represents 1 millimeter). The values d0, d1, ..., which represent distances between devices, are such that $0 < d_0 < d_1 < \dots$. Since $\rho = 1.0$ at $d = 0.0$, this value-pair is omitted. Also, since a correlation function is symmetric, only the positive values d0, d1, ..., are entered. The correlation values rho0, rho1, ..., are in the (exclusive) range (-1.0, 1.0).

In the current release, spatial correlation applies only to device variations. To use this feature, the parasitics SBPF file must contain the pin physical locations. You must also set the **read_parasitics_load_locations** variable to *true*, prior to the **read_parasitics** command.

The command returns 1 upon success, 0 upon failure.

DESCRIPTION

Creates a correlation type.

EXAMPLES

The following example creates a correlation type with constant correlation 1.

```
pt_shell> create_correlation -name wid -constant 0
pt_shell> create_correlation -name d2d -constant 1
pt_shell> create_correlation -name part1 -constant 0.3
pt_shell> create_correlation -name cross -cross_correlations {0.3 -0.2 -0.05}
```

SEE ALSO

```
set_variation_correlation(2)
read_parasitics_load_locations(3)
read_parasitics(2)
```

create_distributed_farm

Creates a virtual distributed farm made up of hosts specified using the **add_distributed_hosts** command.

SYNTAX

```
int create_distributed_farm  
[-timeout seconds]  
[-min_hosts num_hosts]
```

ARGUMENTS

[-timeout seconds]
Specifies in seconds how long the **create_distributed_farm** command waits for remote hosts to come online. While the command is waiting no other commands can be executed at the master. If all the remote hosts come online before the timeout has expired, the command returns immediately on connection of the final host. If all the remote hosts do not come online before the timeout has expired, the command returns and accepts the remainder of the hosts online in the background. The value specified must be greater than or equal to zero. If the option is not specified, the default value of 21600s (6 hours) is used.

[-min_hosts num_hosts]
Specifies the minimum number of hosts that the **create_distributed_farm** command waits for to come online. While the command is waiting, no other commands can be executed at the master. As soon as the minimum number of hosts comes online, the command returns immediately and allow commands to execute at the master. The *-timeout* option to the command overrides the *-min_hosts* option; therefore, if the minimum number of hosts do not come online before the timeout has expired, the command is governed by the behavior of the *-timeout* option as described above. The value specified must be greater than or equal to zero. If the option is not specified, the default value of 256 hosts is used.

DESCRIPTION

This command creates a virtual farm by pooling together hosts from various sources (lsf, generic, grd, now). The various sources are specified using the **add_distributed_hosts** command.

EXAMPLES

In the following example, a virtual farm (made up of 11 hosts) is created. platinum1: 2x64 bit hosts, platinum2: 4x32 bit hosts, lsf: 5x32 bit hosts.

```
pt_shell> add_distributed_hosts -farm now -num_of_hosts 2 platinum1  
1  
pt_shell> add_distributed_hosts -farm now -num_of_hosts 4 platinum2 -32bit  
1  
pt_shell> add_distributed_hosts -farm lsf -submission_script "/bin/lsf_stub"  
-options { -R "tmp>300" -q} -num_of_hosts 5
```

create_distributed_farm

```
1
pt_shell> create_distributed_farm
1
```

SEE ALSO

```
add_distributed_hosts(2)
report_distributed_hosts(2)
```

create_eco_astro_constraints

Creates ECO Astro constraints to fix crosstalk delay, static timing, or static noise.

SYNTAX

```
int create_eco_astro_constraints
[-output file_name]
[-constraint_type list_of_constraint_types]
[-delay_type delay_type]
[-effort_level effort_level]
[-validate]
[-group path_group_name]
[-max_constraints number_of_constraints]
[-max_iteration number_of_iterations]
[-verbose]
[object_list net_or_paths]
```

Data Types

<i>file_name</i>	string
<i>list_of_constraint_types</i>	list
<i>delay_type</i>	string
<i>effort_level</i>	string
<i>path_group_name</i>	string
<i>number_of_constraints</i>	int
<i>number_of_iterations</i>	int
<i>nets_or_paths</i>	list

ARGUMENTS

- output *file_name*
Specifies the name of the output constraint file. If not specified, the default name, *constraints.dat*, is used.
- constraint_type *list_of_constraint_types*
Specifies types of constraints to generate. List one or more types of delta_delay, stage_delay, noise, and double_switching. The default value is {delta_delay stage_delay}.
- delay_type *delay_type*
Specifies the delay type of the constraints. Values are *min*, *max*, and *min_max*. The default value is *max*.
- effort_level *effort_level*
Specifies the effort level to generate timing constraints. The default value is low.
If low, PrimeTime SI generates constraints for large delays. If high, PrimeTime SI tries to remove all timing violations by iterative approach. Thus, the high effort reduces the number of violations, but might increase the number of constraints and runtime.

-validate
 Specifies whether to validate the constraints in PrimeTime SI after constraint generation. PrimeTime SI reads back and annotate the constraints to check the new timing with the generated constraints.

-group *path_group_name*
 Specifies clock groups for which you want to generate constraints.

-max_constraints *number_of_constraints*
 Specifies the maximum number of constraints to be generated. If not specified, the default value is 1000.

-max_iteration *number_of_iterations*
 Specifies maximum number of iterations in high effort level. If not specified, the default value is 3.

-verbose
 Turns on verbose mode.

object_list *net_or_paths*
 Specifies a list of nets or paths to be analyzed. If not specified, PrimeTime SI locates them automatically.

DESCRIPTION

This command creates ECO constraints for Astro to fix delta delay, stage delay, and noise. Astro uses these constraints to perform ECO fixing.

By specifying the **-constraint_type** option, you can control what types of constraints are generated. If no option is specified, PrimeTime SI generates constraints to make the slacks of the paths going through SI bottleneck nets positive.

By default, PrimeTime SI reduces delta delay first, then reduces stage delay, if necessary. It does so by considering delta delay and stage delay at the same time when it generates constraints, and it optimizes the constraints as well as the number of constraints.

If you want to fix only delta delay violations, use the **-constraint_type** option with **delta_delay** specified.

This command must be used at the ECO fixing stage before sign off. It means the design is almost clean, but there are a few timing violations or PrimeTime SI issues that you need to fix before sign-off. You must not use this command if your design has too many violations. It might result in generating a large number of constraints that Astro cannot accept.

You can control the number of constraints by changing **-max_constraints** options.

To fix for the double_switching violations in the design use **-constraint_type double_switching**.

You can check the new timing with the generated constraints if you enable **-validate** analysis. PrimeTime SI reads the constraints back to itself, and run incremental update timing to see how much improvement has been achieved. This option is

especially useful when you want to check your ECO fix before place-and-route.

Without any effort level specified, PrimeTime SI uses low effort by default and generate constraints to reduce the worst timing violations. However, if you would like to generate more complete set of constraints to remove most of timing violations, use high effort level. PrimeTime SI spends more time for its analysis and iterates until no violations are found. The *-max_iteration* option limits the number of iterations.

EXAMPLES

The following example is used to run the **create_eco_astro_constraints** command to fix delta delay only. This generates constraints to fix delta delay.

```
create_eco_astro_constraints -output \  
my_constraint.dat -constraint_type delta_delay
```

If fixing delta delay is not enough to make slack positive, use stage delay fixing in addition to delta delay fixing as shown below:

```
create_eco_astro_constraints \  
-constraint_type {delta_delay stage_delay}
```

The following is used to run the **create_eco_astro_constraints** command in high effort level. This setting tries to remove all violations by spending more time on analysis.

```
create_eco_astro_constraints -effort_level high
```

If you like to check timing with the generated constraints, specify the *-validate* option as following:

```
create_eco_astro_constraints -validate
```

You can see verbose messages by specifying the *-verbose* option as follows:

```
create_eco_astro_constraints -verbose
```

The following fixes noise violations:

```
create_eco_astro_constraints -constraint_type {noise}
```

The following fixes delta, stage delay, and noise at the same time:

```
create_eco_astro_constraints \  
-constraint_type {delta_delay stage_delay noise}
```

If you want to fix only the worst timing paths, use the following.


```
set paths [get_timing_paths]  
create_eco_astro_constraints $paths
```

If you want to fix only particular nets, use the following.

```
set nets [get_nets my_nets*]  
create_eco_astro_constraints $nets
```

SEE ALSO

```
update_noise(2)  
update_timing(2)
```

create_generated_clock

Creates a generated clock object.

SYNTAX

```
string create_generated_clock
[-name clock_name]
-source master_pin
[-divide_by divide_factor | -multiply_by multiply_factor |
  -edges edge_list ]
[-combinational]
[-duty_cycle percent]
[-invert]
[-edge_shift edge_shift_list]
[-add]
[-master_clock clock]
[-pll_output output_pin]
[-pll_feedback feedback_pin]
source_objects
```

Data Types

<i>clock_name</i>	string
<i>master_pin</i>	list
<i>divide_factor</i>	int
<i>multiply_factor</i>	int
<i>percent</i>	float
<i>edge_list</i>	list
<i>edge_shift_list</i>	list
<i>clock</i>	string
<i>output_pin</i>	list
<i>feedback_pin</i>	list
<i>source_objects</i>	list

ARGUMENTS

- name *clock_name***
Specifies the name of the generated clock. If you do not use this option, the clock receives the same name as the first clock source specified in the **-source** option. If you specify the **-add** option, you must use the **-name** option and the clocks with the same source must have different names.
- source *master_pin***
Specifies the master clock source (a clock source pin in the design) from which the clock waveform is to be derived. Note that the actual delays (latency) for the generated clock are computed using its own source pins and not the *master_pin* option.
- divide_by *divide_factor***
Specifies the frequency division factor. If the *divide_factor* value is 2, the generated clock period is twice as long as the master clock period.

create_generated_clock

`-multiply_by multiply_factor`

Specifies the frequency multiplication factor. If the *multiply_factor* value is 3, the generated clock period is one-third as long as the master clock period.

`-edges edge_list`

Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must be not less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.

`-combinational`

The source latency paths for this type of generated clock only includes the logic where the master clock propagates. The source latency paths do not flow through sequential element clock pins, transparent latch data pins, or source pins of other generated clocks.

`-duty_cycle percent`

Specifies the duty cycle, in percentage, if frequency multiplication is used. Duty cycle is the high pulse width.

`-invert`

Inverts the generated clock signal (in the case of frequency multiplication and division).

`-edge_shift edge_shift_list`

Specifies a list of floating point numbers that represents the amount of shift, in library time units, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge shifts specified must be equal to the number of edges specified. The values can be positive or negative; positive indicating a shift later in time, while negative indicates a shift earlier in time. For example, 1 indicates that the corresponding edge is to be shifted by one library time unit.

`-add`

Specifies whether to add this clock to the existing clock or to overwrite it.

Use this option to capture the case where multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin.

Ideally, one generated clock must be specified for each clock that fans in to the master pin.

If you specify this option, you must also use the *-name* and *-master_clock* options.

Defining multiple clocks on the same source pin or port causes longer runtime

and higher memory usage than a single clock, because PrimeTime must explore

all possible combinations of launch and capture clocks.

Use the **set_false_path** command to disable unwanted clock combinations.

-master_clock *clock*

Specifies the master clock to be used for this generated clock if multiple clocks fan into the master pin. If you specify this option, you must also use the *-add* option.

-pll_output *output_pin*

Specifies the output pin of the PLL which is connected to the feedback pin. For single output PLLs, this pin is same as the pin on which the generated clock is defined.

-pll_feedback *feedback_pin*

Specifies the feedback pin of the PLL. There should be a path in the circuit connecting one of the outputs of the PLL to this feedback pin.

source_objects

Specifies a list of ports, pins or nets defined as generated clock source objects. When a net is used as the source, the first driver pin of the net is the actual source used in creating the generated clock.

DESCRIPTION

Creates a generated clock object in the current design and also defines a list of objects as generated clock sources in the current design. You can specify a pin or a port as a generated clock object. The command also specifies the clock source from which it is generated. The advantage of using this command is that whenever the master clock changes, the generated clock automatically changes.

The generated clock can be created as one of the following:

- A frequency divided clock by using the *-divide_by* option
- A frequency multiplied clock by using the *-multiply_by* option
- A special divide by one by using the *-combinational* option
- An edge-derived clock by using the *-edges* option

The frequency-divided or frequency-multiplied clock can be inverted by using the *-invert* option. The shifting of edges of the edge-derived clock is specified by using the *-edge_shift* option. The *-edge_shift* option is used for intentional edge shifts and not for clock latency.

The number of edges specified by *-edges* to make one period of the generated clock waveform must be an odd number equal to or greater than 3. For example, in the following command edge 1 indicates the first rising of the generated clock, edge 3 indicates the first falling of the generated clock, and edge 5 indicates the next rising of the generated clock. Note that the period of the generated clock is determined by the final entry in the edge list:

create_generated_clock

create_generated_clock -source clk -edges { 1 3 5 } [get_pins flop/Q]

Non-increasing edges as -edges { 1 1 3 } is allowed and usually used with -edge_shift to produce a generated clock pulse independent of the duty cycle of the master clock itself.

If a generated clock is specified with a *divide_factor* value that is a power of 2 (1, 2, 4, ...), the rising edges of the master clock are used to determine the edges of the generated clock. If the *divide_factor* value is not a power of two, the edges are scaled from the master clock edges.

Using the **create_generated_clock** command on an existing *generated_clock* object overwrites its attributes. The **generated_clock** objects are expanded to real clocks at the time of analysis.

The **set_clock_latency**, **set_clock_uncertainty**, **set_propagated_clock**, and **set_clock_transition** commands can reference the *generated_clock*.

For internally generated clocks, PrimeTime automatically computes the clock source latency if the master clock of the generated clock has propagated latency and no user-specified value for generated clock source latency exists. If the master clock is ideal and has source latency and there is no user-specified value for the generated clock's source latency, zero source latency is assumed.

To display information about generated clocks, use the **report_clock** command.

EXAMPLES

The following example creates a frequency divide_by 2 generated clock.

```
pt_shell> create_generated_clock -divide_by 2 \
        -source [get_pins CLK] [get_pins pinf]
```

The following example creates a frequency divide_by 3 generated_clock. If the master clock period is 30, and master waveform is {24 36}, the generated clock period is 90 with waveform {72 108}.

```
pt_shell> create_generated_clock -divide_by 3 \
        -source [get_pins CLK] [get_pins div3/Q]
```

The following example creates a frequency multiply_by 2 generated_clock with a duty cycle of 60%.

```
pt_shell> create_generated_clock -multiply_by 2 -duty_cycle 60 \
        -source [get_pins CLK] [get_pins pinf1]
```

The following example creates a frequency multiply_by 3 generated_clock with a duty cycle equal to the master clock duty cycle. If the master clock period is 30, and master waveform is {24 36}, the generated clock period will be 10 with waveform {8 12}.

```
pt_shell> create_generated_clock -multiply_by 3 \  
         -source [get_pins CLK] [get_pins div3/Q]
```

The following example creates a generated clock whose edges are edges 1, 3, and 5 of the master clock source.

```
pt_shell> create_generated_clock -edges {1 3 5} \  
         -source [get_pins CLK] [get_pins pinf2]
```

The following example shows the generated clock in the previous example with each derived edge shifted by 1 time unit.

```
pt_shell> create_generated_clock -edges {1 3 5} -edge_shift {1 1 1} \  
         -source [get_pins CLK] [get_pins pinf2]
```

The following example creates an inverted clock.

```
pt_shell> create_generated_clock -divide_by 1 -invert
```

The following example creates a rising edge pulse triggered by the rising edge of its master clock.

```
pt_shell> create_generated_clock -edges {1 1 3} -edge_shift {0 5 0} \  
         -source [get_pins CLK] [get_pins pinf2]
```

The following example creates a falling edge pulse triggered by the rising edge of its master clock with a period 10.

```
pt_shell> create_generated_clock -edges {1 1 3} -edge_shift {0 5 0} \  
         -invert -source [get_pins CLK] [get_pins pinf2]
```

The following example creates a frequency doubling pulse. A rising edge pulse triggered by both rising and falling edges of its master clock with a period of 10.

```
pt_shell> create_generated_clock -edges {1 1 2 2 3} -  
edge_shift {0 2.5 0 2.5 0} \  
         -source [get_pins CLK] [get_pins pinf2]
```

SEE ALSO

```
check_timing(2)  
create_clock(2)  
get_generated_clocks(2)  
remove_generated_clock(2)  
report_clock(2)  
set_clock_latency(2)  
set_clock_transition(2)  
set_clock_uncertainty(2)
```

```
create_generated_clock
```

```
set_false_path(2)
set_propagated_clock(2)
```

create_ilm

Extracts interface logic model and writes it to a new directory. Also, sets the **is_interface_logic_pin** attribute on pins of the current design that are part of its interface logic model.

SYNTAX

```
int create_ilm
[-script_format format]
[-instances instance_list]
[-ignore_ports port_list]
[-auto_ignore]
[-verbose]
[-ignore_boundary_pins pin_list]
[-include incl_list]
[-verification_script]
[-latch_level levels]
[-context_borrow]
[-keep_ignored_fanout]
[-include_pins pin_list]
[-critical_pins]
[-traverse_disabled_arcs]
[-parasitics_options para_options]
[-sdf_options sdf_options]
[-block_scope]
[-block_scope_only]
[-scope_scenario scenario_name]
[-validate valid_list]
```

Data Types

<i>port_list</i>	list
<i>level</i>	int
<i>instance_list</i>	list
<i>pin_list</i>	list
<i>incl_list</i>	list
<i>para_options</i>	list
<i>sdf_options</i>	list
<i>scenario_name</i>	string

ARGUMENTS

-script_format *format*
Specifies the output format for the script. Allowed values are *ptsh* (the default) for *pt_shell*, *dcsh* for *dc_shell*, and *dctl* for *dc_shell-t*.

-instances *instance_list*
Specifies the list of instances for which ILMs need to be generated. Separate ILMs are generated for each instance and written out to directories named after the instance name.

-ignore_ports *port_list*

Specifies a list of input or output ports whose fanout or fanin is to be ignored when setting the **is_interface_logic_pin** attribute. Substitute the list of ports you want for the *port_list* option. Clock ports are automatically ignored; you do not have to specify them in this list.

You can use this option to exclude the fanout of ports connected to chip-level nets (for example, scan enable and reset) from impacting the contents of interface logic. If these nets are not explicitly ignored, they cause unnecessary logic (for example, internal registers) to be part of the interface logic on the current design. You can also use this option to selectively generate the interface logic for a subset of the ports on a block; for example, an interface logic model (ILM) can model only the timing behavior on a subset of the ports on a block.

By default, all nonclock input and all output ports are used in identifying the contents of interface logic. The **-ignore_ports**, **-ignore_boundary_pins**, and **-auto_ignore** options are mutually exclusive.

-auto_ignore

Enables automatic determination of ports whose fanout should be ignored when setting the **is_interface_logic_pin** attribute. A port is ignored if the percentage of the total registers in the design in the transitive fanout of the port exceeds a specified threshold percentage contained in the **ilm_ignore_percentage** variable (default 25).

You can use this option to help you identify the test enable and reset ports of your design. Before using it, examine the current value of the **ilm_ignore_percentage** variable and reset it, if necessary. Note that the **-auto_ignore** option might potentially ignore ports you do not want to ignore, or fail to ignore ports you want to ignore. Carefully read the messages issued by this command when you use this option to see which ports have been ignored and what percentage of registers to which they fanned out.

The **-ignore_ports**, **-ignore_boundary_pins** and **-auto_ignore** options are mutually exclusive.

-verbose

Indicates that information about the number of cells and nets in the original design and in the ILM netlist is to be written to standard output.

-ignore_boundary_pins *pin_list*

Specifies a list of boundary pins whose fanout or fanin is to be ignored when placing the **is_interface_logic_pin** attribute while extracting an instance ILM. Works similar to the **-ignore_ports** option but works for instance ILMs. This option can only be used along with the **-instances** option.

The **-ignore_ports**, **-ignore_boundary_pins** and **-auto_ignore** options are mutually exclusive.

-include *incl_list*

Specifies the additional categories to be included in the ILM. Allowed values are *net_pins*, *boundary_cells*, *si_delay_pins* and *si_noise_pins*.

The *net_pins* value specifies that all pins on non-clock interface logic nets and propagated clock interface logic nets are to be included in the netlist. Use this option if the interface logic model (ILM) will be used in non-SDF flows and delay calculation will be performed using the information contained in the ILM. Preserving all pins on a net maintains correct pin capacitance information for the net. The *net_pins* does not affect non-propagated clock nets; that is, nets in the clock network that have user-specified source

latency.

The *boundary_cells* value specifies that the ILM has to include the boundary cells for all input ports. By default, the boundary cells would be present for all inputs except for the ignored ports. This option allows users to include boundary cells for the ignored input ports also so that DRC checks can be performed at top level.

The *si_delay_pins* value specifies that the ILM has to include additional logic required to perform cross-talk delay analysis at the chip-level. Since the cross-talk flow involves detailed parasitics, the *si_delay_pins* value also turns the *net_pins* value on.

The *si_noise_pins* value specifies that the ILM has to include additional logic required to perform noise analysis at the chip-level. Since the noise flow involves detailed parasitics, the *si_noise_pins* value also turns the *net_pins* value on.

-verification_script

Specifies that a script needs to be generated that can be used for the verification of ILM. By default, **create_ilm** command generates a script that can be used when the ILM is instantiated at upper level of hierarchy. This option additionally generates a script that can be used to validate the ILM against the original block from which the ILM was generated.

-latch_level levels

Specifies the number of logic levels over which time borrowing can occur for latch chains that are a part of the interface logic. Substitute the number of levels you want for the *levels* option. By default, all latches found in interface logic are assumed to be potential borrowers. Thus, all logic from I/O ports to flip-flops or output ports are identified as belonging to interface logic.

When you use this option, note that the value of the *level* option must account for the borrowing behavior of latches only on interface timing paths. If n represents a latch level, then $n + 1$ represents the number of latches maintained in latch chains that originate at input ports. The $n + 1$ th latch functions as an edge-triggered register and decouples I/O paths from internal paths. Use this option only when there are latches in the interface logic for a block. Specifying this option might potentially result in less accurate models, because not all latches are allowed to borrow. The *-context_borrow* and *-latch_level* options are mutually exclusive.

-context_borrow

Specifies that latch borrowing at the interface should be established based on the current context defined for the design. So, latches that borrow based on arrival times defined on input ports and clocks defined on the design are traced through, but path tracing stops at nonborrowing latches. The *-context_borrow* and *-latch_level* options are mutually exclusive.

-keep_ignored_fanout

Specifies that the fanout from ignored input ports to interface logic is to be maintained in the model. Thus, ignored ports are not used to identify interface logic, but connections between ignored ports and interface logic are preserved. Timing slacks for ignored ports might differ from those reported on the original netlist because connections from ignored ports to internal registers are not preserved in the model.

-include_pins *pin_list*

Specifies a list of pins that must be included in the interface logic. Substitute the list of pins you want for the *pin_list* option. This option can be used to optionally add internal pins to the interface logic. After the interface logic is determined, this list of pins is added to the interface logic. You can use this option to define additional interface logic pins that will otherwise be removed in the model. There will not be any affect on pins that are already in the interface logic. Use this option in conjunction with the **all_fanin** and **all_fanout** commands to include a particular cone of logic in the model.

-critical_pins

Specifies that critical pins interface logic must be identified. This option allows you to extract a model that keeps only the pins in the critical paths of the design.

You can use this option to extract a critical pins interface logic. The model generated contains the interface logic pins in the critical paths of the design. The model is compact compared to normal interface logic models, but might not be as accurate outside of the current context. This model can be used only to do critical path analysis. It might take much longer to generate this model than the normal model.

The **-include {si_delay_pins}** option is not currently supported for critical pins ILM.

-traverse_disabled_arcs

Specifies that the interface logic should not be affected by disabled arcs/pins on the design. If specified, this option forces the traversal of disabled arcs while determining interface logic.

-parasitics_options *para_list*

Specifies that the parasitics need to be generated for ILM. Allowed values for the list are: *spef_format*, *sbpf_format*, *input_port_nets* and *constant_nets*.

The *spef_format* value specifies that the parasitics are to be written in SPEF. The *sbpf_format* value specifies that the parasitics are to be written in SBPF. The *input_port_nets* value indicates that parasitic information is to be written out for nets connected to the input ports on the current design. By default, this information is not written out.

The *constant_nets* value indicates that parasitic information is to be written out for constant nets also. By default, this information is not written out. When the **-parasitics_options** value is specified, all the net pins of ILM nets are included. In other words, the **-include {net_pins}** option is always assumed when the **-parasitics_options** value is used.

-sdf_options *sdf_options*

Specifies that SDF needs to be generated for ILM. Allowed values for the list are: *annotated*, *no_edge*, *2.1_version*, *3.0_version*, *input_port_nets*, *output_port_nets*.

The *annotated* value indicates that the SDF is to include only timing arcs that have been annotated with the **read_sdf**, **set_annotated_delay**, or **set_annotated_check** commands.

The *no_edge* value indicates that the generated SDF is not to include any edges (posedge or negedge) for combinational IOPATHs.

The *2.1_version* value selects which SDF version of 2.1. This is the default version.

The *3.0_version* value selects which SDF version of 3.0.

The *input_port_nets* value indicates that the SDF file is to include delays of nets connected to input ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available.

The *output_port_nets* value indicates that the SDF file is to include delays of nets connected to output ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available.

-block_scope

Specifies that the block scope information needs to be captured for the timing model. If specified, this creates a separate file that contains scope information for the block that will be replaced by ILM at the top-level analysis. Note that the scope information that will be captured and stored is not impacted by the **hier_scope_check_defaults** variable.

-block_scope_only

Specifies that only the block scope information needs to be captured, and not the ILM. This option is useful for the later runs when an ILM was already generated but the scope of block-level validation has changed. If specified, this creates a separate file that contains scope information for the block that will be replaced by ILM at the top-level analysis. Note that the scope information that will be captured and stored is not impacted by the **hier_scope_check_defaults** variable.

-scope_scenario *scenario_name*

Specifies the name of the scenario for which the scope data needs to be labeled and later checked for. The scope information captured and stored in the scope data file will be marked as corresponding to the given scenario. If not specified, a fixed default name is used internally.

-validate *valid_list*

Specifies what should be validated after the model is created. The allowed value is timing.

DESCRIPTION

Extracts an interface timing model (ILM) for the design or given list of instances. The ILM is written to a directory with name of the design, or the hierarchical name of instance with slash ('/') replaced by underscore ('_'). The location of this potentially new directory is given by the **pt_ilm_dir** variable.

The command generates the following files: *dir_name/ilm.v* (Verilog design for the ILM) *dir_name/ilm_inst.pt.gz* (script to apply the instance constraints)

Additionally, the command might generate: *dir_name/ilm.{sbpf, spef.gz}* (Parasitics if *-parasitics_options* is given) *dir_name/ilm.sdf* (SDF file if *-sdf_options* is given) *dir_name/ilm_verif.pt.gz* (verification script for the design) *dir_name/ilm.txt* (list of aggressor annotation pins if *-include {si_delay_pins}* is given) *<dir_name>/ilm.scope* (scope information for the block being replaced by model)

Also, the command sets the *is_interface_logic_pin* attribute on pins of the current design that are part of its interface logic.

The interface logic on a block contains all cells whose timing is impacted by, or impacts, the external environment of a block. The following list describes such parts of interface logic:

- All cells in timing paths that lead from input ports to registers or output ports that terminate the paths.
- All cells in timing paths that lead to output ports from registers or input ports that originate the paths.
- Clock trees that drive interface registers; including any registers in the clock tree. Clock-gating circuitry is part of interface logic if it is driven by external ports, but not if it is driven by registered outputs on a block.

Notice that interface logic does not include internal register-to-register paths and logic on a block associated only with these paths.

Additional logic may be included using the `-include_pins` option. If the `-include {si_delay_pins}` option is given, additional register-to-register logic may be pulled in that is needed for cross-talk analysis at chip level.

This command implicitly performs an `update_timing` on the design if required. You can review the objects you have identified as interface logic by using the **`get_ilm_objects`** command.

Also, the command can be used to generate scope information for the block that will be replaced with the ILM at top-level.

If the `-validate` option is given and variable **`hier_modeling_version`** is set to 2.0, the model will be validated automatically after the model is created.

EXAMPLES

The following example extracts ILM and sets the **`is_interface_logic_pin`** attribute on all nonclock pins in the current design, except for pins in the fanin/fanout of ports `port1`, `port2`, and `port3`.

```
pt_shell> create_ilm -ignore_ports [get_ports {port1 port2 port3}]
```

The following example extracts ILM for instance `I1`, additionally includes three levels of logic from internal pin `I1/ff/Q` and also includes internal cross-talk pins needed for top-level cross-talk analysis.

```
pt_shell> set_ilm_pins [all_fanout -level 3 \
-flat [get_pin I1/ff/Q]]
```

```
pt_shell> create_ilm -instances {I1}\
-include_pins $ilm_pins -include {si_delay_pins}
```

The following example extracts a critical pins interface timing model.

```
pt_shell> create_ilm -critical_pins -include {net_pins}
```

The following example extracts ILM by placing the **is_interface_logic_pin** attribute on all nonclock pins in the current design, except for pins identified by the **-auto_ignore** option. Because the value of the **ilm_ignore_percentage** variable is currently 35, pins are ignored if the percentage of the total design registers in their transitive fanout is greater than 35. The **-latch_level** option with a value of 1 states that the first latch encountered in IO paths might potentially borrow, but the second latch can be treated as an edge-triggered device. Thus, two levels of latches are maintained in the interface logic. Also, it creates scope information for the block.

```
pt_shell> printvar ilm_ignore_percentage  
ilm_ignore_percentage = "35"
```

```
pt_shell> create_ilm -auto_ignore -latch_level 1 -block_scope
```

SEE ALSO

```
hier_modeling_version(3)  
pt_ilm_dir(3)  
ilm_ignore_percentage(3)  
check_block_scope(2)  
hier_scope_check_defaults(3)
```

create_lcd_operating_condition

Creates an LCD operating condition by using different operating conditions in the library.

SYNTAX

```
int create_lcd_operating_condition  
[-op_worst op_worst_name]  
[-op_nominal op_nom_name]  
[-op_best op_best_name]  
[-mult_worst op_worst_mult]  
[-mult_nominal op_nom_mult]  
[-mult_best op_best_mult]  
[-library lib_name]  
[name]
```

Data Types

<i>op_worst_name</i>	string
<i>op_nom_name</i>	string
<i>op_best_name</i>	string
<i>op_worst_mult</i>	float
<i>op_nom_mult</i>	float
<i>op_best_mult</i>	float
<i>lib_name</i>	string
<i>name</i>	string

ARGUMENTS

-op_worst *op_worst_name*
Specifies the worst case operating condition name to be used in LCD.

-op_nominal *op_nom_name*
Specifies the nominal operating condition name to be used in LCD.

-op_best *op_best_name*
Specifies the best case operating condition name to be used in LCD.

-mult_worst *op_worst_mult*
Specifies the multiplier to be applied to worst case operating condition.

-mult_nominal *op_nom_mult*
Specifies the multiplier to be applied to nominal operating condition.

-mult_best *op_best_mult*
Specifies the multiplier to be applied to best case operating condition.

-library *lib_name*
Specifies the name of the library for the LCD operating condition.

name
Specifies the name for the LCD operating condition.

DESCRIPTION

Creates an LCD operating condition in the specified library. A technology library contains a fixed set of operating conditions. This command allows you to create an LCD operating condition that adds delay uncertainty to each gate based on LCD derived from the operating conditions already existing in the library or the operating conditions that you created. You can use these LCD operating conditions to calculate gate delay in min-max mode to analyze timing with statistical effects.

LCD for one delay path considers using three operating conditions (worst case, nominal, and best case) with different weights to compute delays per mode (early mode and late mode).

When calculating a delay, a given operating condition with the correspondent output loading is used. Currently, the delay rules (DCM) only support worst-case and best-case pin capacitances. In order to support nominal delay the assumption is made so that the output loading is linear between best-case and worst-case operating conditions. Therefore, the output loading for nominal is the sum of best case and worst case divided by 2.

For slew propagating, if the sum of the weights is larger than 1.0, unwanted pessimism is introduced because of the ripple effect of the dependency of output slew on input slew of a gate. To avoid this behavior, the slew at the output of each delay path is divided by the sum of the weights for the given operating condition that was created out of an LCD.

To see the operating conditions defined for a library, use the **report_lib** command.

To set LCD operating conditions on the **current_design**, use the **set_operating_conditions** command.

EXAMPLES

The following examples show how to create LCD operating conditions and how to set them so that you can use them.

```
pt_shell> create_lcd_operating_condition -op_worst wccom -op_nominal nocom -  
op_best bccom  
-mult_worst 0.7 -mult_nominal 0.2 -mult_best 0.1 -library library_name lcd_late  
  
pt_shell> create_lcd_operating_condition -op_worst wccom -op_nominal nocom -  
op_best bccom  
-mult_worst 0.1 -mult_nominal 0.2 -mult_best 0.7 -library library_name lcd_early  
  
pt_shell> set_operating_conditions -max lcd_late -min lcd_early -  
analysis_type on_chip_variation
```

SEE ALSO

set_operating_conditions(2)
report_lib(2)

create_net

Creates nets in the current design.

SYNTAX

```
int create_net  
[-exact] net_list
```

Data Types

net_list list

ARGUMENTS

-exact

Indicates that names are to be used exactly as specified. Use this option if you want to create a net that contains a hierarchy or wildcard character as part of the net name. For more information, see the section entitled "Naming Nets with Special Characters".

net_list

Specifies a list of nets to be created.

DESCRIPTION

The **create_net** command creates new nets in the current design. Like all other netlist editing commands, for the **create_net** command to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. The **create_net** command returns a 1 if successful and a 0 if unsuccessful.

The **create_net** command behaves almost identically to the **create_cell** command. Nets are created in scope; that is, at or below the current instance. Net names are specified as for other commands, using the full hierarchical name relative to the current instance, as in the following example:

```
create_net u1/u2/new_net class/AN2
```

This command attempts to create a net named *new_net* in the hierarchical block *u1/u2*. The name to the right of the last hierarchical separator is the actual net name, and the remainder is sent to the search engine to find a hierarchical block in which to create the new net. The operation fails if any of the following are true:

- The search engine cannot find "u1/u2".
- The search engine finds multiple blocks that match "u1/u2".
- The search engine finds a leaf cell matching "u1/u2".

- The search engine finds a net, port or hierarchical pin matching "u1/u2/new_net".

Naming Nets With Special Characters

If you want to create a net whose name contains the current hierarchical separator or wildcard characters used by the search engine, you must do the following:

- Use **current_instance** to change the scope to the block in which you want the new net created.
- Use **create_net -exact** to create the net.

For examples, see the EXAMPLES section.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

The following example creates nets in the current instance, and at a level below the current instance. Currently, u1 is an instance of a design that has multiple instances. Therefore, it is uniquified as part of the editing operation.

```
pt_shell> create_net new_net1
Information: Created net 'new_net1' in design 'top'. (NED-016)
1
```

```
pt_shell> create_net u1/new_net1
Uniquifying 'u1' (block1) as 'block1_0'.
Information: Created net 'new_net1' in 'top/u1'. (NED-016)
1
```

The following example creates net a/b in the hierarchical block u1/u2. The first attempt fails, because the **-exact** option was omitted. Here, u1/u2 is an instance of a design that has multiple instances; therefore, it is uniquified as part of the editing operation. u1 was already uniquified in the previous example.

```
pt_shell> current_instance u1/u2
u1/u2

pt_shell> create_net a/b
Error: Could not create net 'a/b':
      a not found. (NED-041)
Error: No changes made. (NED-040)
0
```

```
pt_shell> create_net a/b -exact
Uniquifying 'u1/u2' (block2) as 'block2_0'.
Information: Created net 'a/b' in 'top/u1/u2'. (NED-016)
1
```

SEE ALSO

`create_cell(2)`
`remove_net(2)`
`size_cell(2)`
`swap_cell(2)`
`write_changes(2)`

create_operating_conditions

Creates a new set of operating conditions in a library.

SYNTAX

```
int create_operating_conditions  
-name name -library library_name  
-process process_value -temperature temperature_value  
-voltage voltage_value [-tree_type tree_type]  
[-calc_mode calc_mode]  
[-rail_voltages rail_value_pairs]
```

Data Types

<i>name</i>	string
<i>library_name</i>	string
<i>process_value</i>	float
<i>temperature_value</i>	float
<i>voltage_value</i>	float
<i>tree_type</i>	string
<i>calc_mode</i>	string
<i>rail_value_pairs</i>	Tcl list

ARGUMENTS

operating conditions, creating

-name *name*
Specifies the name of the new set of operating conditions.

-library *library_name*
Specifies the name of the library for the new operating conditions.

-process *process_value*
Specifies the process scaling factor for the operating conditions. Allowed values are 0.0 through 100.0.

-temperature *temperature_value*
Specifies the temperature value, in degrees Celsius, for the operating conditions. Allowed values are -300.0 through +500.0.

-voltage *voltage_value*
Specifies the voltage value, in volts, for the operating conditions. Allowed values are 0.0 through 1000.0.

-tree_type *tree_type*
Specifies the tree type for the operating conditions. Allowed values are *best_case_tree*, *balanced_tree* (the default), or *worst_case_tree*. The tree type is used to estimate interconnect delays by providing a model of the RC tree.

-calc_mode *calc_mode*

For use only with DPCM libraries. Specifies the DPCM delay calculator mode for the operating conditions; analogous to the **-process** option used in Synopsys libraries. Allowed values are *unknown* (the default), *best_case*, *nominal*, or *worst_case*. The default behavior (*unknown*) is to use worst case values during analysis similarly to *worst_case*. If **-rail_voltages** are specified, the command sets all (*worst_case*, *nominal*, and *best_case*) voltage values.

-rail_voltages *rail_value_pairs*

Specifies a list of name-value pairs that defines the voltage for each specified rail. The name is one of the rail names defined in the library; the value is the voltage to be assigned to that rail. By default, rail voltages are as defined in the library; use this option to override the default voltages for specified rails.

DESCRIPTION

"environmental characteristics" "operating conditions" Creates a new set of operating conditions in the specified library. A technology library contains a fixed set of operating conditions; this command allows you to create new, additional operating conditions.

To see the operating conditions defined for a library, use **report_lib**.

To set operating conditions on the current design, use the **set_operating_conditions** command.

EXAMPLES

The following example creates a new set of operating conditions called WC_CUSTOM in the library "tech_lib", specifying new values for process, temperature, voltage, and tree type. By default, rail voltages remain as defined in the library.

```
pt_shell> create_operating_conditions -name WC_CUSTOM \  
-library tech_lib -process 1.2 -temperature 30.0 -voltage 2.8 \  
-tree_type worst_case_tree
```

The following example creates a new set of operating conditions called OC3, in the library "IBM_CMOS5S6_SC", specifying new values for process, temperature, voltage, and rail voltages for rails VTT and VDDQ. By default, the tree type *balanced_tree* is used.

```
pt_shell> create_operating_conditions -name OC3 \  
-lib IBM_CMOS5S6_SC -proc 1.0 -temp 100.0 -volt 4.0 \  
-rail_voltages {VTT 3.5 VDDQ 3.5}
```

SEE ALSO

set_operating_conditions(2)
report_lib(2)

create_operating_conditions

create_power_domain

Creates a power domain at the specified scope, which provides a power supply distribution network.

SYNTAX

```
string create_power_domain  
domain_name  
[-elements element_list]  
[-include_scope]  
[-scope instance_name]
```

Data Types

<i>domain_name</i>	string
<i>element_list</i>	list
<i>instance_name</i>	string

ARGUMENTS

domain_name
Specify the name of the power domain to be created. The name should be a simple (non-hierarchical) name.
If there is a power domain with the same name in the specified scope, the power domain cannot be created. If there is a hierarchical/leaf cell instance or port with the same name in the specified scope, the power domain cannot be created either.
This option must be specified.

-elements *element_list*
Specify a list of cells which are added as the extent of the power domain. The list of cells should be hierarchical, macro, or IO pad cells, but PrimeTime accepts any cell. Specified cells cannot be added in other power domains of the same scope.
If neither **-elements** nor **-include_scope** is specified, the power domain consists of the current scope and any of its children not specified as elements in another **create_power_domain** command.

-include_scope
If this option is specified, the scope of the power domain is also included in the extent of the power domain. This means all elements within the current scope will get the supply as the power domain.

-scope *instance_name*
Specify in which scope the power_domain is going to be created. The instance name is the name of a hierarchical cell.
If this option is not specified, the power_domain will be created in current scope.

DESCRIPTION

The *create_power_domain* command enables you to create a power domain in the specified scope. A power_domain is a collection of design elements that share a primary power and ground power net. The logic hierarchy level where a power domain is created is called the scope of the power domain. Any design elements that belong to a power domain are said to be in the extent of that power domain. While a design element can be in the scope of a number of power domains, it can only be in the extent of one power domain.

A power domain could have several supply_nets, supply_nets are connected to power_domain via supply_port. And a power_switch of a power_domain could be used to turn on/off the power supply of part/whole power domain. Supply_net, supply_port and power_switch could be created using command *create_supply_net*, *create_supply_port* and *create_power_switch* respectively.

This command returns collection object (the newly created power domain). It returns null string if it failed.

Non-UPF Mode (Power Domains Mode):

This command works in power domains mode as well, and take following options instead.

int **create_power_domain**

domain_name

[-power_down]

[-power_down_ctrl *list*]

[-power_down_ack *list*]

[-power_down_ctrl_sense <0 or 1>]

[-object_list *list*]

-power_down

Use this to specify the new power domain is a power down domain. This switch is a prerequisite for -power_down_ctrl switch.

-power_down_ctrl *list*

Use this to specify the power down control net for the new power domain. The switch -power_down is required in order to use this option. It is also a prerequisite for -power_down_ack option.

-power_down_ack *list*

Use this to specify the power down acknowledge net for the new power domain. The switch -power_down_ctrl is required in order to use this option.

-power_down_ctrl_sense <0 or 1>

Use this to specify the sense of the power down control net for the new power domain. It takes values of 0 or 1. It specifies the active low or active high sense type for power control signal. The switch -power_down_ctrl is required in order to use this option.

-object_list *list*

A list of hierarchical cells that are associated with the new power domain. When this option is absent, the new power domain is assumed to be the top

create_power_domain

level domain. For each design, there can only be one top level domain. Also, each hierarchical cell can only be associated with one power domain. The **create_power_domain** command creates a new power domain for the current design. The power domains for each design must be uniquely named. There can be only one design-wise top level power domain.

Use `-power_down` to specify a new domain is a power down domain. An always-on power domain is created without this switch. For a power down domain, use `-power_down_ctrl` to specify an optional power down control net; use `-power_down_ack` to specify an optional power down acknowledge net.

Use `-object_list` to associate a list of hierarchical cells with the new power domain. When this option is not used, the new power domain is considered as top-level power domain. Each hierarchical cell can only belong to one power domain.

By default, power down control and power acknowledge signals are marked dont touch. Set variable `dont_touch_power_domain_control_nets` to false if you do not want these signal marked dont touch.

EXAMPLES

UPF Mode:

The following example creates two power_domains in scope INST1:

```
prompt> create_power_domain PD1 -elements INST1/SUB_INST -scope INST1
{"INST1/PD1"}
prompt> create_power_domain PD2 -elements INST1/SUB_INST -scope INST1
Error: Cell 'INST1/SUB_INST' is already in the extent of power domain 'PD1'. (UPF-001)
prompt> create_power_domain PD2 -scope INST1 -include_scope
{"INST1/PD2"}
```

Non-UPF Mode:

The following examples use the command to create power domains.

```
prompt> create_power_domain TOP_DOMAIN
1
prompt> create_power_domain SUB_DOMAIN -power_down \
    -power_down_ctrl [get_nets pd_ctrl] \
    -power_down_ack [get_nets pd_ack] \
    -object_list [get_cells mid1]
1
```

SEE ALSO

```
report_power_domain(2),
help UPF,
help "power domains",
power_domains_compatibility(3).
```

create_power_group

Creates a power group of cells in the current design.

SYNTAX

```
int create_power_group  
-name name  
[-object_list]  
[-default]
```

Data Types

<i>name</i>	string
<i>object_list</i>	list

ARGUMENTS

-name name
Specifies the name for the power group. The name should not conflict with names of existing power groups.

-object_list
Specifies a list of cells to be included in this power group. This option is mutual exclusive with the *-default* option.

-default
Indicates that the specified power group is predefined and the default list of cells are to be included in the power group. This option is mutual exclusive with the *-object_list* option.

DESCRIPTION

Group a list of cells of special interests (not necessarily in the same hierarchy) to form a power group. The **update_power** and **report_power** commands are able to generate power waveform and power report for this power group.

The cells contained in power groups are leaf cells. If a hierarchical cell is specified in the *-object_list* option, all the leaf cells contained by the hierarchical cell, instead of the hierarchical cell itself, are added to the power group.

There are several power groups predefined and automatically created by the tool. However, if a predefined power group has been removed, the *-default* option can be used to regenerate it.

The following is a list of the predefined power groups ordered by priority from high to low. Note that the predefined power groups are not supposed to be overlapping with one another. If one cell happens to belong to more than one group, it is put into the group with the higher priority.

io_pad
All I/O PAD cells.

memory
All memory cells.

black_box
All black boxes.

clock_network
Cells in the clock network. The contents is consistent with the results from the **get_clock_network_objects -type cell** command. If the **power_clock_network_include_clock_gating_network** variable is set to *true*, the contents is consistent with the results from the **get_clock_network_objects -type cell -include_clock_gating_network** command.

register
The latches, flip-flops driven by the clock network. The contents is consistent with the results from the **get_clock_network_objects -type register** command. If the **power_clock_network_include_clock_gating_network** variable is set to *true*, the contents is consistent with the results from the **get_clock_network_objects -type register -include_clock_gating_network** command.

sequential
All other sequential cells.

combinational
All other combinational cells.

EXAMPLES

In the following example, a power group called `clock_tree` is created to accommodate all the clock tree cells.

```
pt_shell> create_power_group -name clock_tree [get_clock_network_objects -type cell]
1
```

In the following example, the predefined power group `clock_network` has been removed earlier, now you want to create it again with the default list of cells.

```
pt_shell> create_power_group -name clock_network -default
1
```

SEE ALSO

```
report_power_groups(2)
remove_power_groups(2)
get_power_group_objects(2)
update_power(2)
report_power(2)
power_clock_network_include_clock_gating_network(3)
```

create_power_net_info

Creates a power net.

SYNTAX

```
int create_power_net_info  
power_net_name  
-power  
-gnd  
[-switchable]  
[-nominal_voltages nominal_voltage_list]  
[-voltage_ranges voltage_range_list]
```

Data Types

<i>power_net_name</i>	string
<i>nominal_voltage_list</i>	list
<i>voltage_range_list</i>	list

ARGUMENTS

power_net_name
The name of the new power net info to be created.

-power
Specifies the type of the new power net is "power". Mutually exclusive with -gnd option.

-gnd
Specifies the type of the new power net is "gnd". Mutually exclusive with -power option.

-switchable
Specifies that this power net is switchable (can be cut off externally, or, if driven by an internal switch, cut off internally). This option can only be used with -power.

-nominal_voltages *nominal_voltage_list*
Specifies the list of nominal voltages this power net has been designed to operate. The actual operating voltage of the power net can deviate from the nominal within a certain tolerance as specified by *voltage_ranges_list*. This optional argument must be specified together with *voltage_ranges_list*. This option can only be used with -power.

-voltage_ranges *voltage_range_list*
Specifies the list of allowed voltage ranges around the nominal voltages that this power net has been designed to operate. This optional argument must be specified together with *nominal_voltages_list*. This option can only be used with -power.

DESCRIPTION

The **create_power_net_info** command creates a new power net info for the current design. The power net is either of type power or of type gnd.

EXAMPLES

The following examples use the command to create power nets.

```
prompt> create_power_net_info VSS_net -gnd
1
prompt> create_power_net_info VDD1_net -power
1
prompt> create_power_net_info VDD2_net -power \
    -switchable -nominal_voltages {0.9 1.08} \
    -voltage_ranges {0.88 0.92 1.05 1.10}
1
```

SEE ALSO

report_power_net_info (2)
set_voltage (2)

create_power_rail_mapping

Map the power rails defined in the libraries to the physical power rails existing in the design.

SYNTAX

```
string create_power_rail_mapping
design_rail
[-lib_rail_name power_rail_name]
[-cells cell_list]
[-off_condition condition]
[-default]
```

Data Types

<i>design_rail</i>	string
<i>power_rail_name</i>	string
<i>cell_list</i>	list
<i>condition</i>	string

ARGUMENTS

design_rail
Defines the power rail used in the design.

-lib_rail_name power_rail_name
Specifies the power rail defined in the library. It can take the power rail name defined in the library and also take "all". The power rail name can be the name of the **voltage_map** command defined in the library. By using the *-lib_rail_name all* option, the mapping is applied to all the power rails in each library associated with the library cells. If the *-lib_rail_name* option is not used, the mapping is applied to the default library power rail in each library associated with the library cells.

-cells cell_list
Specifies the instance names or collections of instances. The instance can be hierarchical or leaf instances. For an instance block, only the top instance needs to be specified. All the instances inside such block will be affected by the mapping. Without this option, the mapping will be created for the whole design.

-off_condition condition
Specifies the power-off condition for the design power rail. At power-off state, the portion of design that is powered off by the specific design power rail will not contribute any dynamic or static power. The Boolean expression takes net names plus operators. The operators supported are: (,), !, +, *, &, |, ^, '. When the expression is evaluated to be "1", the design power rail is in power-off state. If the expression is evaluated to be "0" or any unknown state, the design power rail is in power-on state. Please make sure that the states of the nets used in the Boolean expression are clearly defined to avoid any unexpected results. If the *-power_off* option is omitted, the design power rail is assumed to be on all the time.

When switching activity information is specified by the VCD file, the state of the power control signal nets are monitored. The power-off expression is evaluated if there is any state change on these control nets. When a specific power rail is powered-off, neither dynamic nor static power will be dissipated on the cell (or part of cell) powered by this rail.

When switching activity information is specified by statistical toggle rate and state probability, the state probability of the power rail being on is calculated. As a default, only leakage power is scaled by the state probability of the associated power rail being on. Set the **power_scale_dynamic_power_at_power_off** variable to true, if dynamic power needs to be scaled too. Dynamic power scaling is only needed if the statistical switching activity includes toggles happened when the rail is being powered off.

-default

Specifies the design power rail to be the default rail in the design. If not specified, the design power rail defined by the first **create_power_rail_mapping** command is assigned as the default design rail.

DESCRIPTION

Use this command to map the power rails defined in the libraries to the physical power rails existing in the design. Such rail mapping is done at cell (instance) level, which gives user the flexibility to connect the same library cell to different physical power rails in a design. The design power rails specified in this command are virtual power rails, which means that the rail connections do not actually exist in the netlist. No voltage level violation will be checked for such virtual rail connections. It is allowed to connect cells with different voltages to the same virtual design power rail. This command enables PrimeTime PX to create collections of instances on different power rails. Thus, provides the capability and flexibility to create rail-based power reporting at design level. This command also enables PrimeTime PX to be sensitive to the power-on/off control signals associated with the design power rails.

Single-rail or multi-rail cell in PrimeTime PX is based on its cell definition in technology library. A multi-rail cell can have rail specific internal or leakage power tables defined in the library. So that power consumption can be reported for different rails.

This command can create the rail mapping for the whole design or for specific instances (hierarchical or leaf-level). Multiple **create_power_rail_mapping** commands can be issued, the latter overwrites the previous settings if it overlaps.

For single-rail design, if not specified, the default library rail will be mapped to the default design rail internally.

The **report_power_rail_mapping** command can be used to report existing power rail mapping information.

EXAMPLES

The following example shows how to create rail mapping and generate rail-based power reports for a multi-rail design. There are three instance blocks in this design:

block1, block2 and ls_block. Block block1 and block2 are both single-rail instance blocks with all the leaf cells powered by the default library power rail defined in each associated technology library. Block block1 is connected to design power rail VDD1 in the design while block block2 is connected to design power rail VDD2 in the design. Block ls_block is a dual-rail instance block with rail V1 and V2 defined in its cells' technology library. Library power rail V1 is connected to design power rail VDD1 and library power rail V2 is connected to design power rail VDD2. The **current_power_rail** command can select the rail of interest and the **update_power** command can generate the power analysis results just for the interested rails. As a result, report Power_VDD1.rpt file will only contain the power consumed on design power rail VDD1 and report Power_VDD2.rpt will only contain the power consumed on design power rail VDD2. Since the **current_power_rail all** command will mark all the design power rails to be interested, report Power_all.rpt file will contain the total power consumed for entire design.

```
pt_shell> create_power_rail_mapping VDD1 -cells block1
pt_shell> create_power_rail_mapping VDD2 -cells block2
pt_shell> create_power_rail_mapping VDD1 -lib V1 -cells ls_block
pt_shell> create_power_rail_mapping VDD2 -lib V2 -cells ls_block
pt_shell> current_power_rail VDD1
pt_shell> update_power
pt_shell> report_power > Power_VDD1
pt_shell> current_power_rail VDD2
pt_shell> update_power
pt_shell> report_power > Power_VDD2
pt_shell> current_power_rail all
pt_shell> update_power
pt_shell> report_power > Power_all
```

SEE ALSO

```
report_power_rail_mapping(2)
current_power_rail(2)
update_power(2)
report_power(2)
read_vcd(2)
read_saif(2)
set_switching_activity(2)
power_scale_dynamic_power_at_power_off(3)
```


create_power_switch

Creates a power switch at the specified power domain. This command is supported only in UPF mode.

SYNTAX

```
string create_power_switch  
switch_name  
-domain domain_name  
-output_supply_port {port_name supply_net_name}  
-input_supply_port {port_name supply_net_name}  
-control_port {port_name net_name}  
-on_state {state_name input_supply_port {boolean_function}}  
[-off_state {state_name {boolean_function}}]  
[-on_partial_state {state_name input_supply_port {boolean_function}}]  
[-error_state {state_name {boolean_function}}]  
[-ack_port {port_name net_name [{boolean_function}}]  
[-ack_delay {port_name delay}]
```

Data Types

<i>switch_name</i>	string
<i>domain_name</i>	string
<i>port_name</i>	string
<i>supply_net_name</i>	string
<i>net_name</i>	string
<i>boolean_function</i>	list
<i>delay</i>	string
<i>state_name</i>	string
<i>input_supply_port</i>	string

ARGUMENTS

switch_name
Specifies the name of the power switch to be created. The name should be a simple (non-hierarchical) name. If a power switch already exists with the same name in the specified power domain, the power switch cannot be created. This option is required.

-domain *domain_name*
Specifies which power domain contains the power switch. If the power domain with the specified name does not exist in the current scope, the command fails. This option is required.

-output_supply_port {*port_name* *supply_net_name*}
Specifies the name of the output port of the power switch and the supply net where the port connects. If a port exists with the same name on the power switch, the command fails. If there is no supply net with the same name in the current scope, the command fails. This option is required.

`-input_supply_port {port_name supply_net_name}`
 Specifies the name of the input port of the power switch and the supply net where the port connects. If a port exists with the same name on the power switch, the command fails. If there is no supply net with the same name in the current scope, the command fails.
 This option is required and you can specify it more than once. One power switch can have multiple input supply ports.

`-control_port {port_name net_name}`
 Specifies the name of the control port of the power switch and the logical net where this port connects. If a port with the specified name already exists on the power switch, the command fails. If a net with the specified name does not exist, the command fails.
 This option is required and can be specified more than once. One power switch can have multiple control ports.

`-on_state {state_name input_supply_port {boolean_function}}`
 Specifies a named on state, the relevant input supply port, and its Boolean function.
 This option can be specified multiple times.

`-off_state {state_name {boolean_function}}`
 Specifies a named off state and its relevant Boolean function.
 This option is currently ignored in PrimeTime.

`-on_partial_state {state_name input_supply_port {boolean_function}}`
 Specifies a named on_partial state and its relevant Boolean function.
 This option is currently ignored in PrimeTime.

`-error_state {state_name {boolean_function}}`
 Specifies a named error state and its relevant Boolean function.
 This option is currently ignored in PrimeTime.

`-ack_port {port_name net_name [{boolean_function}]}`
 Specifies the name of the acknowledge port of the power switch and the logical net where this port connects. If a port with the specified name already exists on the power switch, the command fails. If a net with the specified name does not exist, the command fails.
 If this option is not specified, the power switch will not have an acknowledge port. Optionally, a Boolean function can also be specified.
 This option is currently ignored in PrimeTime.

`-ack_delay {port_name delay}`
 Specifies the acknowledge port on the switch and the corresponding acknowledge delay.
 This option is currently ignored in PrimeTime.

DESCRIPTION

This command enables you to create a power switch at the specified power domain. The switch is created within the scope of the power domain. Each power switch must be connected with an input supply net and an output supply net. The power switch can be connected with an acknowledge logical net and several control logical nets. Each net is connected with a power switch via a switch port. The switch ports are

automatically created if the power switch is created successfully.

This command returns the full name of the power switch (from the current scope) upon success and the command returns a null string upon failure.

EXAMPLES

The following example creates power switch SW1 within power domain PD1:

```
pt_shell> create_power_switch SW1 -domain PD1 \  
          -output_supply_port {vout VNO2} \  
          -input_supply_port {vin1 VNI1} \  
          -control_port {ctrl_small ON1} \  
          -on_state {full_s vin1 {ctr_small}} \  
          \
```

SEE ALSO

```
report_power_switch(2)  
get_power_switches(2)
```

create_qtm_constraint_arc

Creates a constraint arc for a quick timing model.

SYNTAX

```
string create_qtm_constraint_arc  
[-name arc_name]  
[-setup]  
[-hold]  
[-from port_name]  
[-to port_spec]  
[-edge triggering_edge]  
[-path_type name]  
[-path_factor multiplication_factor]  
[-value constraint_value]
```

Data Types

<i>arc_name</i>	string
<i>port_name</i>	string
<i>port_spec</i>	list
<i>triggering_edge</i>	string
<i>name</i>	string
<i>multiplication_factor</i>	float
<i>constraint_value</i>	float

ARGUMENTS

-name *arc_name*
Specifies the name of the constraint arc.

-setup
Creates a setup arc.

-hold
Creates a hold arc.

-from *port_name*
Specifies the constraining port name. Define this port as a clock port.

-to *port_spec*
Specifies the constrained port. Define this port as an input/inout port.

-edge *triggering_edge*
Specifies the triggering edge of the clock (rise/fall).

-path_type *name*
Specifies the path type you want to use to set the delay.

-path_factor *multiplication_factor*
Specifies the multiplication factor for the path type.

create_qtm_constraint_arc

`-value constraint_value`

Specifies the delay value in terms of the time units you use for the model.

DESCRIPTION

This command allows you to create a constraint arc in a quick timing model. The `-from` port denotes the constraining port, and the `-to` port denotes the constrained port. The `-from` port must be defined as a clock port and the `-to` port must be an input/inout port.

The `-setup` and `-hold` switches are used to specify either a setup arc or a hold arc. You must use one of the switches, but you cannot use both simultaneously. To specify the triggering edge of the clock, use the `-edge` option. The option takes `rise/fall` as valid values.

To specify the delay value, use the `-path_type` option. You can specify the delay as a value in terms of the time units used in the design. Use either the `-path_type` or the `-value` option; you cannot use both simultaneously.

If the constraint arc is a setup arc, the quick timing model (QTM) parameter global setup time is added to the value of the specified setup. If the constraint arc is a hold arc, the global hold time is added to the value of hold specified. You must define the global setup time before creating any setup arc and global hold time before creating any hold arc. The command checks to see if these parameters are defined. Global setup and time, and global hold time are defined using the **set_qtm_global_parameter** command.

To see the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, see the **create_qtm_model** man page. For a more detailed description about QTM, see the *PrimeTime Modeling User Guide*.

EXAMPLES

The following command creates a setup arc from the positive edge of constraining pin CLK to IN1 (constrained pin) with a delay equivalent to 2 times that of path type path1.

```
pt_shell> create_qtm_constraint_arc -setup -edge rise -from CLK -to IN1
-path_type path1 -path_factor 2
```

The following command creates a hold arc from the positive edge of constraining pin CLK to IN1 (constrained pin) with a delay equivalent to 2 times that of path type path1.

```
pt_shell> create_qtm_constraint_arc -hold -edge rise -from CLK -to IN1
-path_type path1 -path_factor 2
```

The following command creates a hold arc from the positive edge of constraining pin CLK to IN1 (constrained pin) with a delay of 3 time units.

```
pt_shell> create_qtm_constraint_arc -hold -edge rise -from CLK -to IN1 -value 3.0
```

SEE ALSO

```
create_qtm_delay_arc(2)  
create_qtm_model(2)  
create_qtm_path_type(2)  
save_qtm_model(2)  
report_qtm_model(2)  
get_qtm_ports(2)
```

create_qtm_delay_arc

Creates a delay arc for a Quick Timing Model (QTM).

SYNTAX

```
string create_qtm_delay_arc  
[-name arc_name]  
-from port_spec  
-to port_spec  
[-edge triggering_edge]  
[-path_type path_type]  
  
[-path_factor multiplication_factor]  
[-value delay_value]
```

Data Types

<i>arc_name</i>	string
<i>port_spec</i>	list
<i>port_spec</i>	list
<i>triggering edge</i>	string
<i>path_type</i>	string
<i>multiplication_factor</i>	float
<i>delay_value</i>	float

ARGUMENTS

-name *arc_name*
Specifies the name of the delay arc.

-from *port_spec*
Specifies the QTM port name or a collection of QTM ports, which is the start point of the delay arc. For an edge triggering arc this must be a clock.

-to *port_spec*
Specifies the QTM port name or a collection of QTM ports. This port must be output/inout type.

-edge *triggering_edge*
Specifies the triggering edge (rise or fall)

-path_type *name*
Specifies the path type you want to use to set the delay.

-path_factor *multiplication_factor*
Specifies the multiplication factor for the path type.

-value *delay_value*
Specifies the delay value in terms of the time units you use for the model.

DESCRIPTION

This command allows you to create a delay arc in a QTM from a QTM port or collection of QTM ports to a port or collection of QTM ports. If you specify a list of ports for the **from** and **to** ports, a delay arc is created from each start port to each end port.

To create a edge triggering arc, use the **-edge** option. For an edge triggering arc, the **clk_to_output** delay (specified as a global parameter) is added to the delay value specified.

You can use the **-path_type** option to specify the delay value or you can specify the delay as a value in terms of the time units used in the design. Use either the **-path_type** or the **-value** option; you cannot use both options simultaneously.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to the **create_qtm_model** man page. For a more detailed description about QTM, please refer to the *PrimeTime User Guide*.

EXAMPLES

The following command creates a delay arc from input ports {A, B, C} to output port D with a delay value of 2.0 time units.

```
pt_shell> create_qtm_delay_arc -from {A, B, C} -to D -value 2.0
```

The following command creates an arc from clock CLK to output port OUT with a delay equivalent to 3 times that of path type 'path1'.

```
pt_shell> create_qtm_delay_arc -from CLK -to OUT -path_type path1 -path_factor 3
```

The following example uses wildcard to match all ports matching "CL*".

```
pt_shell> create_qtm_delay_arc -from CL* -to OUT -path_type path1 -path_factor 3
```

The following example uses a collection for the QTM port 'CLK':

```
pt_shell> create_qtm_delay_arc -from [get_qtm_ports CLK] -to OUT -path_type path1 -path_factor 3
```

SEE ALSO

```
create_qtm_constraint_arc(2)
create_qtm_model(2)
create_qtm_path_type(2)
get_qtm_ports(2)
report_qtm_model(2)
save_qtm_model(2)
```

create_qtm_delay_arc

create_qtm_drive_type

Creates a drive type in a Quick Timing Model (QTM) description.

SYNTAX

```
string create_qtm_drive_type -lib_cell lib_cell_name
[-input_pin pin_name]
[-output_pin pin_name]
[-input_transition_rise rtrans]
[-input_transition_fall ftrans]
drive_type_name

string lib_cell_name
string input_pin_name
string output_pin_name
float   rtrans
float   ftrans
string drive_type_name
```

ARGUMENTS

-lib_cell *lib_cell_name*
Specifies the library cell name in the technology library.

-input_pin *pin_name*
Specifies the input pin in the *lib_cell* is the start point of the arc and ends in the output pin.

-output_pin *pin_name*
Specifies the output pin in the *lib_cell* that specifies the drive.

-input_transition_rise *rtran*
Specifies the input rising transition time associated with the **-input_pin** to compute the delay for the drive arc. If you do not include this option, the delay table for rising input of the drive arc will be loaded from library as is.
Use the **-input_transition_rise** and **-input_transition_fall** options to capture the transition time associated with the *input_pin*. This can obtain more accurate information on the transition time and delay time for the defined drive arc.

-input_transition_fall *ftran*
Specifies the input falling transition time associated with the *input_pin* to compute the delay for the drive arc. If you do not include this option, the delay table for falling input of the drive arc will be loaded from library as is.

drive_type_name
Specifies the name given to the defined drive type.

DESCRIPTION

This command can be used to create a drive type. The defined drive type can be used to set the drives on output ports. The drive type is specified by referring to a `lib_cell` in the technology library. The `lib_cell` is specified by using the **-lib_cell** option. This `lib_cell` must be present in the technology library.

The output pin which drives the port is specified by using the **-output_pin** option. In addition, you can specify the input pin by using the **-input_pin** option. This option selects an arc that drives the output pin of the `lib_cell`.

If neither the input pin nor the output pin is specified, an arbitrary arc is chosen to define the drive type. If the input pin is specified and output pin is not specified, an arbitrary arc starting from the input pin defines the drive type. If the output pin is specified and input pin is not specified, the arbitrary delay arc coming into the output pin defines the drive type. If both input and output pins are specified, the `lib_cell` must have a combinational arc from the input pin to the output pin.

To use this command you must read in the technology library and then specify it by using the **set_qtm_technology** command.

To show the information about the current QTM model, use **report_qtm_model** command.

For a basic description QTM, see the **create_qtm_model** man page. For a more detailed description, see Chapter 12 of the *PrimeTime User Guide*.

EXAMPLES

In the following examples it is assumed that you have loaded the technology library, and the QTM technology is set. Do this by using the following the sequence of commands as a guide, substituting the name of your technology library.

```
pt_shell>read_db my_technology_library.db
```

```
pt_shell>set_qtm_technology -library my_technology_library
```

The following command creates a drive type equivalent to the `lib_cell` **BUF1**.

```
pt_shell> create_qtm_drive_type -lib_cell BUF1 drive1
```

The following command creates a drive type **drive2** equivalent to `lib_cell` **BUF2** and specifies the output pin to use.

```
pt_shell> create_qtm_drive_type -lib_cell BUF2 -output_pin Y drive2
```

SEE ALSO

create_qtm_load_type (2), **create_qtm_model** (2), **create_qtm_path_type** (2), **report_qtm_model** (2), **save_qtm_model** (2), **set_qtm_port_drive** (2), **set_qtm_technology** (2).

create_qtm_generated_clock

Creates a QTM generated_clock.

SYNTAX

```
string create_qtm_generated_clock  
-source master_clock_name  
[-divide_by divide_factor | -multiply_by multiply_factor]  
[-invert]  
generated_clock_name
```

"Data Types

<i>master_clock_name</i>	string
<i>generated_clock_name</i>	string
<i>divide_factor</i>	int
<i>multiply_factor</i>	int

ARGUMENTS

generated_clock_name
Specifies the name of the generated clock. If the name has been used in defining a port or internal pin, then the generated clock will be defined on the existing port or internal pin. Otherwise, a new same-named internal pin will be created to be the source of the generated clock.

-source *master_clock_name*
Specifies the name of the clock defined as master source of the generated clock. The master source must be defined as a clock, or a generated clock, prior to the generated clock definition.

-divide_by *divide_factor*
Specifies the frequency division factor. If the *divide_factor* value is 2, the generated clock period is twice as long as the master clock period.

-multiply_by *multiply_factor*
Specifies the frequency multiplication factor. If the *multiply_factor* value is 3, the generated clock period is one-third as long as the master clock period.

-invert
Inverts the generated clock signal (in the case of frequency multiplication and division).

DESCRIPTION

This command creates a QTM generated clock. A generated clock can be defined on the external port of the QTM, as well as the internal pin in the QTM. After defining the generated clock, the source port/pin can be used in other QTM commands the same way as those port/pins defined by the **create_qtm_port** command, i.e. delay arcs from or

to the generated clock source pin can be defined with the **create_qtm_delay_arc** command; constraint arcs related to the generated clock can be defined with the **create_qtm_constraint_arc** command.

For a basic description about QTM, please refer to the **create_qtm_model** man page. For a more detailed description about QTM, refer to the *PrimeTime Model User Guide*.

EXAMPLES

The following example creates a divide-by 2 generated clock on an internal pin, assume there is no port defined with name "GCLK_int", with master clock CLK.

```
pt_shell> create_qtm_generated_clock GCLK_int -source CLK -multiply_by 2
```

The following example creates a generated clock on port GCLK, assume GCLK has already been defined as a port with command `create_qtm_port`.

```
pt_shell> create_qtm_generated_clock GCLK -source CLK -divide_by 2 -invert
```

SEE ALSO

```
create_qtm_model(2)
report_qtm_model(2)
save_qtm_model(2)
get_qtm_ports(2)
set_qtm_delay_arc(2)
set_qtm_constraint_arc(2)
```

create_qtm_load_type

Creates a load type for a Quick Timing Model (QTM) description.

SYNTAX

```
string create_qtm_load_type  
-lib_cell name  
[-input_pin pin_name]  
load_type name
```

Data Types

<i>lib_cell_name</i>	string
<i>input_pin_name</i>	string
<i>load_type name</i>	string

ARGUMENTS

-lib_cell *name*
Specifies the name of the library cell in the technology library.

-input_pin *pin_name*
Specifies the input pin in the lib_cell to specify capacitance.

load_type name
Specifies the name given to the defined load type.

DESCRIPTION

This command is used to create a load type. A load type can be used to define the load on a QTM input port. The load type is specified by referring to the *lib_cell* option in the technology library. The **lib_cell** command is specified using the -*lib_cell* option. The **lib_cell** command must be present in the specified technology library.

The input pin can be specified by using the -*input_pin* option. If the input pin is not specified, an arbitrary input pin is chosen to define the load type.

To use this command you must first read in the technology library, then specify the technology library by using the **set_qtm_technology** command.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to the **create_qtm_model** man page. For a more detailed description about QTM, refer to the *PrimeTime Model User Guide*.

EXAMPLES

In the following examples the technology library is loaded by you and the QTM

technology is set. This is done by the following sequence of commands:

```
pt_shell>read_db my_technology_library.db  
pt_shell>set_qtm_technology -library my_technology_library
```

The following command creates a load type 'load1' using cell OR1. Since the input pin name is not specified, QTM selects an arbitrary input pin.

```
pt_shell> create_qtm_load_type -lib_cell OR1 load1
```

The following command creates a load type 'load2' using cell OR1, input pin A.

```
pt_shell> create_qtm_load_type -lib_cell OR1 -input_pin A load2
```

SEE ALSO

```
create_qtm_drive_type(2)  
create_qtm_model(2)  
create_qtm_path_type(2)  
report_qtm_model(2)  
save_qtm_model(2)  
set_qtm_port_load(2)
```

create_qtm_model

Begins the definition of a Quick Timing Model (QTM) description.

SYNTAX

string **create_qtm_model** *model_name*

Data Types

model_name string

ARGUMENTS

model_name
Specifies the name of the generated model.

DESCRIPTION

Specifies the name of a new quick timing model (QTM) to be created by PrimeTime. QTMs are temporary timing models that can be created quickly for a block using PrimeTime commands. The purpose of these commands is to provide timing information without the necessity of writing a detailed timing model.

QTMs are intended to be used early in the design cycle to describe rough initial timing of a block. These models will eventually be replaced, either by some other detailed timing model or by the netlist of the block, to obtain more accurate timing. For a more detailed description of QTMs, see the *PrimeTime Model User Guide*.

A QTM can be saved as a Synopsys DB file, which can be instantiated in a design in the same way library cells or ITS models are instantiated. Both Design Compiler and PrimeTime accept designs with instantiated QTMs. To save a QTM model, use the **save_qtm_model** command.

Other commands in the QTM command set must be placed between a **create_qtm_model** and **save_qtm_model** command. Many QTM commands exist for specifying, configuring and examining QTMs. The following is not an exhaustive list, but gives examples of tasks you can perform using QTM commands:

- To create a QTM port, use the **create_qtm_port** command.
- To create constraint arcs and delay arcs, use the **create_qtm_constraint_arc** and the **create_qtm_delay_arc** commands, respectively.
- To set the drive of a QTM output port, use the **set_qtm_port_drive** command.
- To set the load of a QTM input port, use **set_qtm_port_load** command.

- To show the information about the current QTM model, use the **report_qtm_model** command.

EXAMPLES

The following command creates a new QTM model with the name adder.

```
pt_shell> create_qtm_model adder
```

SEE ALSO

```
create_qtm_constraint_arc(2)
create_qtm_delay_arc(2)
create_qtm_drive_type(2)
create_qtm_load_type(2)
create_qtm_path_type(2)
create_qtm_port(2)
get_qtm_ports(2)
report_qtm_model(2)
save_qtm_model(2)
set_qtm_global_parameter(2)
set_qtm_port_drive(2)
set_qtm_port_load(2)
set_qtm_technology(2)
```


create_qtm_path_type

Creates a path type in a Quick Timing Model (QTM) description.

SYNTAX

```
string create_qtm_path_type  
-lib_cell name  
[-input_pin pin_name]  
[-output_pin pin_name]  
[-fanout count]  
path_type name
```

Data Types

<i>name</i>	string
<i>pin_name</i>	string
<i>count</i>	integer-range
<i>path_type name</i>	string

ARGUMENTS

-lib_cell *name*
Specifies the name of the library cell in the technology library.

-input_pin *pin_name*
Specifies the input pin in the *-lib_cell* option, which is the start point of the arc, to define the path type.

-output_pin *pin_name*
Specifies the output pin in the *-lib_cell* option, which is the end point of the arc, to define the path type.

-fanout *count*
Specifies the average fanout (number of pins) to consider while computing the delay of the path type. The fanout count can range from 1 to 1000. By default this is 1.

path_type name
Specifies the name given to the defined path type.

DESCRIPTION

This command is used to create a path type. A path type mimics an arc in a library cell. An arc in the QTM model is defined in terms of the delays of the path type. The path type is specified by referring to a library cell in the technology library. The library cell is specified using the *-lib_cell* option. The library cell must be present in the technology library specified.

You can specify the average fanout count the arc fans out to while defining the path type. It is assumed the arc fans out to the first input pin of the same library cell, while calculating the delays. The input pin is specified by using the -

input_pin option. The output pin is specified by using the *-output_pin* option.

If neither input pin nor the output pin is specified, an arbitrary delay arc from the library cell is chosen to define the path type.

If the input pin is specified and output pin is not specified, an arbitrary delay arc starting from the input pin defines the path type.

If the output pin is specified and input pin is not specified, an arbitrary delay arc coming into the output pin defines the path type.

If both input and output pins are specified, the library cell must have an arc from the input pin to the output pin.

To use this command you must read in the technology library then specify the technology library by using the **set_qtm_technology** command.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to the **create_qtm_model** man page. For a more detailed description about QTM, refer to the PrimeTime Modeling User Guide.

EXAMPLES

In the following examples the technology library is loaded by the user and the QTM technology is set. This is done by the following sequence of commands:

```
pt_shell>read_db my_technology_library.db
```

```
pt_shell>set_qtm_technology -library my_technology_library
```

The following command creates a path type path1 by using lib_cell AN2, with an average fanout count of 2 (uses default input, output pins).

```
pt_shell> create_qtm_path_type -lib_cell AN2 -fanout 2 path1
```

The following command creates a path type path2 by using cell AN2, with an average fanout count of 2, and using pin 'A' as the starting point for the path type.

```
pt_shell> create_qtm_path_type -lib_cell AN2 -input_pin A -fanout 2 path2
```

The following command creates a path type path3 by using cell AN2, with an average fanout count of 2, using 'A' as the start point of the arc, and pin 'Z' as the end

point of the arc.

```
pt_shell> create_qtm_path_type -lib_cell AN2 -input_pin A -output_pin Z -  
fanout 2 path3
```

SEE ALSO

```
create_qtm_constraint_arc(2)  
create_qtm_delay_arc(2)  
create_qtm_drive_type(2)  
create_qtm_load_type(2)  
create_qtm_model(2)  
report_qtm_model(2)  
save_qtm_model(2)
```

create_qtm_port

Creates a Quick Timing Model (QTM) port.

SYNTAX

```
string create_qtm_port  
-type port_type  
port_list
```

Data Types

```
port_type      string  
port_list      list
```

ARGUMENTS

```
-type port_type  
    Specifies the type of port. The port can be one of the following types: input,  
    output, inout, internal or clock. If you want a port to be a clock, define  
    it as a clock port.  
  
port_list  
    Specifies the list of QTM ports you want created.
```

DESCRIPTION

This command creates QTM port. You can create a single port or a list of ports with this command. The ports can be input, output, inout, internal or clock. Any port that constrains another port or has a edge triggered launch arc originating from it, has to be defined to be of the type 'clock'.

Bused ports are also defined by giving the start and end index (A[0:5]). Bused ports cannot be internal.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to the **create_qtm_model** man page. For a more detailed description about QTM, please refer to the *PrimeTime User Model Guide*.

EXAMPLES

The following example creates an input bused QTM port A[0:5].

```
pt_shell> create_qtm_port A[0:5] -type input
```

The following example creates a clocked QTM port CLK.

```
pt_shell> create_qtm_port CLK -type clock
```

The following example creates QTM output ports A, B, C, and D.

```
pt_shell> create_qtm_port {A B C D} -type output
```

The following example creates QTM internal pin D_int.

```
pt_shell> create_qtm_port D_int -type internal
```

SEE ALSO

```
create_qtm_model(2)  
report_qtm_model(2)  
save_qtm_model(2)  
get_qtm_ports(2)  
set_qtm_port_drive(2)  
set_qtm_port_load(2)
```

create_scenario

Creates a scenario for multi-scenario analysis.

SYNTAX

Boolean **create_scenario**

ARGUMENTS

```
-name name  
[-common_data file_list]  
[-specific_data file_list]  
[-common_variables variable_list]  
[-specific_variables variable_list]  
[-image image]
```

stringname

list *file_list*

list *variable_list*

string *image*

-name

A unique string used to identify and to refer to a single scenario.

[-common_data *file_list*]

A list of files that contains commands and data that is common to more than one scenario. These files will be used to create a common image for all scenarios which share the same common data. This option cannot be used in conjunction with the -image option.

[-common_variables *variable_list*]

A list of variables that are common to more than one scenario. These variables will be used in conjunction with the common_data files to determine commonality between scenarios. The value of the variable currently at the master will be set on the slave prior to the sourcing of the common data files. This option cannot be used in conjunction with the -image option.

[-specific_data *file_list*]

A list of files that contains commands and data that is specific to this scenario. This data will not be used on any other scenario or in the common image generation. This data that is specific to the scenario is sent directly to the slave and executed once the common design image has been generated. This option can be used in conjunction with the -image option.

[-specific_variables *variable_list*]

A list of variables that are specific to this scenario. The value of the variable currently at the master will be set on the slave prior to the sourcing of the specific data files. This option can be used in conjunction with the -image option.

[-image *image*]

This option allows a scenario to be created with a previously saved scenario image. The image can be an image generated within a standard primetime

create_scenario

session, an image generated with a `save_session` call from the DMSA master or a `current_image` generated during a DMSA session. The `-specific_variables` and the `-specific_data` options can be specified in conjunction with the `-image` option. The `specific_variables` are set after the image has been loaded and then the `specific_data` files are executed.

DESCRIPTION

The **`create_scenario`** command creates a scenario based on the data supplied by the above arguments. A scenario can be created with a set of scripts and variables. When a set of scenarios share the same `common_data` and `common_variables`, a common design image will be created which will be shared across the set of scenarios. Alternatively a scenario can be created with a previously saved image. This command does not actually start the analysis. The analysis will be started when the first merged reporting or `remote_execute` command is issued.

EXAMPLES

In the following example, a scenario is created named `scen1`. `scen1` is described by the files `common_s1`, and `specific_s1`.

```
pt_shell> create_scenario -name scen1 -common_data {common_s1.pt}  
-specific_data {specific_s1.pt}  
1
```

In the following example, an iterative loop creates a set of scenarios. The same script `single.tcl` is used for all scenarios but the corner variable distinguishes the scenarios

```
foreach corner {bc tc wc} {  
  create_scenario  
    -name ${corner}  
    -common_variables {corner}  
    -common_data "single.tcl"  
}
```

SEE ALSO

`remove_scenario(2)`, `get_current_scenario (2)`, `report_multi_scenario_design (2)`,

create_si_context

Generates an SI context for selected blocks of the design. A top level design and full chip binary parasitics can also be generated.

SYNTAX

```
Boolean create_si_context  
[-include include_list]  
[-instances instance_list]  
[-parasitics_options para_options]  
[-top_inst instance_name]  
[-no_design_parasitics]
```

Data Types

<i>include_list</i>	list
<i>instance_list</i>	list
<i>para_options</i>	list
<i>instance_name</i>	string

ARGUMENTS

- include *include_list***
If this option is not specified, the context script will be generated for cross-talk. You can use this option to specify PrimeTime SI to generate the context scripts for cross-talk, noise or both. Allowed values are *xtalk* and *noise*. The generated context will be the same, only the script will be different.
- instances *instance_list***
Indicates that PrimeTime SI is to extract contexts for the given list of instances or for all the top level instances of the design.
- parasitics_options *para_options***
Specifies that a parasitic file has to be generated to be used to annotate the (block + context) design. The allowed values for the *para_options* options are *spef_format* and *sbpf_format*.
- top_inst *instance_name***
Specifies that PrimeTime should generate a top level design for the given instance. By default, PrimeTime generates top level design for the given design. This option can be used in conjunction with the *-instances* option to generate contexts and top level design for an arbitrary hierarchy. Note that all the instances provided in the *-instances* option must be the child instances of the top level instance provided in this option.
- no_design_parasitics**
By default, this command also generates full chip parasitics in SBPF format. Use this option to indicate PrimeTime not to generate the SBPF file. This option should be used if your parasitic extraction tool generated SBPF.

DESCRIPTION

The **create_si_context** command generates contexts for all top level instances of the design or for the list of instances given in the *-instances* option. This context can be used to perform signal integrity analysis at the block level.

The effect of context on the block is the same as that of the top level and other blocks on the block. Hence, context enables accurate signal integrity analysis for block level analysis as if the block is being analyzed in the context of the full chip.

The command also generates a top level design and a top level script that can be used to perform chip level analysis after block level analysis is complete and respective ILMs are generated for each block. User can direct PrimeTime to generate contexts and top level design for an arbitrary level of hierarchy instance using the *-top_inst* and *-instances* options.

The command creates one directory for each block (for which a context is being extracted) and outputs the context files inside the directory. Each directory contains a verilog context design (named wrapper.v), a TCL script (named wrapper.tcl) that can be used in performing block level analysis, a text file that contains aggressor annotation points (named wrapper.txt) that can be used later for annotating arrivals/slews for accurate block level analysis. The command also generates an infinite arrival script (named wrapper.pt.gz) that can be used to perform conservative block analysis in the context. The command can generate this arrival script for cross-talk, noise or both depending on the *-include* option. In addition, if the *-parasitics_options* option is given, a parasitic file (named wrapper.sbpf or wrapper.spef.gz) gets generated. Also at the level of this new directory, a top level verilog file (named design.v), a top level TCL script (named design.tcl) are also generated. In addition, if the *-no_design_parasitics* option is not given, a full chip binary parasitic file (named design.sbpf) gets generated.

The **create_si_context** command is affected by the **pt_ilm_dir** environment variable. The top design files and directories for individual blocks are created at the location given by the **pt_ilm_dir** variable.

EXAMPLES

The following example generates contexts for all top level designs along with context parasitics in SBPF and a top level design.

```
pt_shell> create_si_context -parasitics_options {sbpf_format}
```

The following example generates contexts for instances "blockA" and "blockB" along with context parasitics in SPEF.

```
pt_shell> create_si_context -instances {blockA blockB} \  
-parasitics_options {spef_format}
```

SEE ALSO

`pt_ilm_dir(3)`
`write_arrival_annotations(2)`
`create_ilm(2)`

create_supply_net

Creates a supply net defined for the specified power domain. The supply net is created in the logic hierarchy at the same scope as the specified power domain.

SYNTAX

```
string create_supply_net  
supply_net_name  
-domain domain_name  
[-reuse]  
[-resolve unresolved | parallel]
```

Data Types

```
supply_net_name    string  
domain_name        string
```

ARGUMENTS

supply_net_name
Specify the name of the supply net to be created. The name should be a simple (nonhierarchical) name.
In the scope of the specified power_domain, if there is a supply net, logical hierarchical net, or logical port with the same name, the *supply_net_name* cannot be created. One exception is that when you use the *-reuse* option, the name must be the same with another existing supply_net in the same scope, and no checking is applied in this situation.

-domain domain_name
Specify which power domain this supply_net is defined for. The power domain must exist.

-reuse
Reuse an existing supply net instead of creating a new one. You can associate one supply net can be with several power domains. If this option is specified, the supply net must exist.

-resolve
Resolution mechanism that determines the state and voltage of the supply net. This is intended also for checking allowed connectivity of supply nets through switches. PrimeTime currently accepts and stores this option, but does not use its value.

DESCRIPTION

The *create_supply_net* command enables you to create a supply net defined in the specified power domain. A supply net connects supply ports and pins.

Each power domain has a primary power supply net and a primary ground supply net, and it could have several other supply nets. If the command succeeds, a supply net is created in the scope of power domain. It is currently neither primary power net

nor primary ground net of the power domain. Use the *set_domain_supply_net* command to set it as the primary power/ground net.

This command returns the full name of the supply net (from the current scope), if successful. It returns null string if it fails.

EXAMPLES

The following example creates a supply_net A_VDD in power_domain PD1, and also recreates A_DD in another power_domain PD2 using the *-reuse* option:

```
prompt> create_power_domain PD1 -element INST1
PD1
prompt> create_power_domain PD2 -element INST2
PD2
prompt> create_supply_net A_VDD -domain PD1
A_VDD
prompt> create_supply_net A_VDD -domain PD2 -reuse
A_VDD
```

SEE ALSO

```
create_power_domain(2)
report_supply_net(2)
set_domain_supply_net(2)
```

create_supply_port

Creates a supply port in specified power domain or in current scope if no power domain is specified.

SYNTAX

```
string create_supply_port  
supply_port_name  
[-domain domain_name]  
[-direction <in|out>]
```

Data Types

```
supply_port_name    string  
domain_name         string
```

ARGUMENTS

supply_port_name
Specify the name of the supply port to be created. The name should be a simple (non-hierarchical) name if *-domain* is specified and should be a full (hierarchical) name if *-domain* is not specified.
In the scope of specified power_domain, if there is a supply_port, logical port, or logical hierarchical net with the same name, the supply_port cannot be created.

-domain domain_name
Specify which power_domain where this port defines a supply net connection point. The power domain must exist in the current scope.

-direction <in|out>
Defines how state information is propagated through the supply network as it is connected to the port. If the port is an input port, the state information of the external supply net connected to the port shall be propagated into the domain. Likewise, for an output port, the state information of the internal supply net connected to the port shall be propagated outside of the domain. The default value is in.

DESCRIPTION

The *create_supply_port* command enables you to create a supply port at the scope of the specified power domain or at the current scope. A supply port provides connection point for the supply net.

This command returns the supply_port object upon success. It returns null string if it failed.

EXAMPLES

The following example creates a supply port VN1 at the scope of the PD1 power

domain:

```
prompt> create_power_domain PD1 -element INST1
PD1
prompt> create_supply_port VN1 -domain PD1
VN1
```

SEE ALSO

```
create_supply_net(2)
create_power_domain(2)
get_supply_ports(2)
```

create_variation

Creates a new variation.

SYNTAX

```
collection create_variation
[-name variation_name]
[-parameter_name parameter_name]
-type distribution_type
-values values_list
[-user_defined]
[-lower_bound lower_bound]
[-upper_bound upper_bound]
```

Data Types

<i>variation_name</i>	string
<i>parameter_name</i>	string
<i>distribution_type</i>	string
<i>values_list</i>	list
<i>lower_bound</i>	float
<i>upper_bound</i>	float

ARGUMENTS

-name *variation_name*
Defines a unique name for the variation.

-parameter_name *parameter_name*
This option is one of the parameter names from the **set_variation_library** or **read_parasitics** command. If the *parameter_name* option is missing, the variation is not associated with and has no effect on any design.

-type *distribution_type*
Defines the type of this distribution. The supported types are *constant*, *discrete*, *empirical*, *lognormal*, *normal*, *pwl*, and *uniform*.

-values *values_list*
A list of floating point numbers that are the arguments for the distribution type.

-user_defined
This option creates a variation whose magnitude comes from user-specified variation characteristics, rather than from parameter data in a variation library.

-lower_bound *lower_bound*
The lower truncation point for this distribution.

-upper_bound *upper_bound*
The upper truncation point for this distribution.

DESCRIPTION

Creates a new variation with the given name and returns a collection containing that variation.

If the *name* is present, it must be a unique variation name. The variation is given this name, which can be used by the **get_variations** command to obtain a reference to this variation.

The *parameter_name* option determines which parameter the variation is associated with. It must be one of the parameters from the **set_variation_library** command or it must be defined in the parasitics file and read in using the **read_parasitics** command. More than one variation can be associated with one parameter.

If the *parameter_name* option is missing, the variation is not associated with any parameter and has no effect on any design; however, it can be used by the variation toolbox commands, such as the **add_variation**, **sub_variation**, **max_variation**, and **min_variation** commands.

The *-user_defined* option creates a special type of variation that does not require variation library data to create timing variation. For these variations, the value of the variation is interpreted as a percentage of underated nominal delay, which is then applied to this nominal delay as timing variation. User-defined variations require a name and cannot have a parameter name, as they do not correspond to any variation library data. However, user-defined variations can still be used together with parameter-based variations to add additional user-defined variation beyond the library-based variations.

The *type*, *values*, *lower_bound*, and *upper_bound* options define the variation's distribution. The values for the different distribution types are as follows:

name	values
constant	c
discrete	x1 p1 x2 p2 ... xn pn
empirical	x1 x2 ... xn
lognormal	mean sigma
normal	mean sigma
pwl	x1 f1 x2 f2 ... xn fn
uniform	a b

EXAMPLES

The following example loads a variation library with parameters "len" and "vth", and then creates a variation of name "Lfab" with parameter_name "len":

```
create_variation
```



```
pt_shell> set_variation_library -parameter_names "len vth" -  
values "100 0.2394" maxvth.db  
pt_shell> create_variation -parameter_name len -name Lfab -type normal -  
values {0 1}]  
_sel23
```

The following example creates a user-defined variation which yields a 1-sigma behavior equal to 5% of the nominal delay:

```
pt_shell> create_variation -user_defined -name user -type normal -values {0 0.05}]  
_sel24
```

SEE ALSO

```
remove_variation(2)  
get_variations(2)  
get_attributes(2)  
get_variation_attributes(2)  
set_variation_library(2)  
read_parasitics(2)  
set_variation(2)  
reset_variation(2)  
add_variation(2)  
sub_variation(2)  
min_variation(2)  
max_variation(2)
```

current_design

Sets or gets the current design in **PrimeTime**.

SYNTAX

```
string current_design [design_name]
```

```
string design_name
```

ARGUMENTS

design_name

Specifies the working or focal design for many **PrimeTime** commands. If *design_name* is not specified, **current_design** returns a collection containing the current design. If *design_name* refers to a design that cannot be found, an error is issued and the working design remains unchanged.

DESCRIPTION

Sets or gets the working design for many **PrimeTime** commands. Without arguments, **current_design** returns a collection containing the current working design. The combination of the current design and current instance defines the context for many **PrimeTime** commands.

To display designs currently available in **PrimeTime**, use **list_designs**.

EXAMPLES

The following example uses **current_design** to show the current context and to change the context from one design to another.

```
pt_shell> current_design  
{ "TOP" }
```

```
pt_shell> list_designs  
Design Registry:  
  ADDER                /designs/dbs/my_design.db:ADDER  
  FULL_ADDER           /designs/dbs/my_design.db:FULL_ADDER  
  FULL_SUBTRACTOR      /designs/dbs/my_design.db:FULL_SUBTRACTOR  
  HALF_ADDER           /designs/dbs/my_design.db:HALF_ADDER  
  HALF_SUBTRACTOR      /designs/dbs/my_design.db:HALF_SUBTRACTOR  
  SUBTRACTOR           /designs/dbs/my_design.db:SUBTRACTOR  
*   TOP                /designs/dbs/my_design.db:TOP
```

```
pt_shell> current_design ADDER  
{ "ADDER" }
```

```
pt_shell> current_design
```

```
current_design
```

```
{"ADDER"}
```

The **current_design** command can be used as a parameter to other **pt_shell** commands. In the following example, **remove_design** is used to delete the working design from **pt_shell**.

```
pt_shell> current_design
{"TOP"}
pt_shell> remove_design [current_design]
Removing design 'TOP'...
1
pt_shell> current_design
Error: Current design is not defined. (DES-001)
pt_shell>
```

SEE ALSO

current_instance (2), **list_designs** (2).

current_instance

Sets the working instance object in **pt_shell** and enables other commands to be used relative to a specific instance in the design hierarchy.

SYNTAX

```
string current_instance [instance]
```

```
string instance
```

ARGUMENTS

instance

Specifies the working instance (cell) in **pt_shell**. If *instance* is not specified, the focus returns to the top level of the hierarchy in the current design. If *instance* is ".", the current instance remains unchanged. If *instance* is "..", the context is moved up one level in the instance hierarchy. If *instance* begins with "/", **pt_shell** sets both the current design and the current instance. More complex examples of *instance* arguments are described in EXAMPLES.

DESCRIPTION

The **current_instance** command sets the working instance in **pt_shell**. An instance is a cell in the hierarchy of a design. This command differs from **current_design**, which changes the working design, then sets the current instance to the top level of the new current design. The combination of the current design and current instance defines the context for many **pt_shell** commands.

current_instance traverses the design hierarchy similar to the way the UNIX "cd" command traverses the file hierarchy. **current_instance** operates with a variety of *instance* arguments:

- If no *instance* argument is specified, the focus of **pt_shell** is returned to the top level of the hierarchy.
- If *instance* is ".", the current instance is returned and no change is made.
- If *instance* is "..", the current instance is moved up one level in the design hierarchy.
- If *instance* is a valid cell (or cell name) at the current level of hierarchy, the current instance is moved down to that level of the design hierarchy.
- Multiple levels of hierarchy can be traversed in a single call to **current_instance** by separating multiple cell names with slashes. For example, **current_instance U1/U2**

current_instance

sets the current instance down two levels of hierarchy.

- The "." directive can also be nested in complex *instance* arguments. For example the command **current_instance "../..MY_INST"** attempts to move the context up two levels of hierarchy, then down one level to the **"MY_INST"** cell.

EXAMPLES

The following example uses **current_instance** to move up and down the design hierarchy. The **all_instances** command is also used to list instances of a specific design relative to the current instance.

```
pt_shell> current_design TOP
{"TOP"}

pt_shell> current_instance U1
U1

pt_shell> current_instance "."
U1

pt_shell> query_objects [all_instances ADDER]
{"U1", "U2", "U3", "U4"}

pt_shell> current_instance U3
U1/U3

pt_shell> current_instance "../U4"
U1/U4

pt_shell> current_instance
Current instance is the top-level of design 'TOP'.
```

In the following example, changing the **current_design** resets the **current_instance** to the top level of the new design hierarchy.

```
pt_shell> current_design
{"TOP"}

pt_shell> current_instance "U2/U1"
U2/U1

pt_shell> current_design ADDER
{"ADDER"}

pt_shell> current_instance .
Current instance is the top-level of design 'ADDER'.
```

The following example uses **current_instance** to go to an instance of another design. The **current_design** is set by the new design whose name is the name after the first slash of the given instance name.

```
pt_shell> current_design
{"TOP"}

pt_shell> current_instance U1
U1

pt_shell> current_instance "/TOP/U2"
U2

pt_shell> current_instance "/ALARM_BLOCK/U6"
U6

pt_shell> current_design
{"ALARM_BLOCK"}
```

SEE ALSO

all_instances (2), **current_design** (2), **list_designs** (2), **query_objects** (2).

current_power_rail

Sets or gets the power rails in a multi-rail design to be included in power analysis. As the default (if not specified), all the power rails in the design are included in power analysis. This command has no effect on single rail design.

SYNTAX

```
int current_power_rail  
[rail_name_list]
```

Data Types

<i>rail_name_list</i>	object_list
<i>rail_name</i>	string

ARGUMENTS

rail_name_list
Specifies the names of the power rails in the design to be included in power analysis. If the *rail_name_list* option is not specified, the **current_power_rail** command returns a collection of power rails being included in power analysis. If the *rail_name* option refers to a power rail that is not defined in the design, an error is issued and the current rail settings remains unchanged.

DESCRIPTION

Sets or gets the power rails in a multirail design to be included in power analysis. Without arguments, the **current_power_rail** command returns a collection of power rails being included in power analysis.

The **current_power_rail** command provides the control of calculating power consumption for the rails of interest. Power consumption data can be generated for different combinations of power rails in a multi-rail design. Only the power dissipated on the power rails in the current rail list will be calculated and reported. For cell internal and leakage power, the rail-based power numbers are calculated based on the rail specific power tables in the technology library. For cell switching power, the rail-based power numbers are calculated based on the output pin's power rail. This command has no effect on single rail design.

Before using this command, you define the design power rails by using the **create_power_rail_mapping** command. The **report_power_rail_mapping** command can be used to show the defined design power rails, library power rails, and rail mapping information.

The following power analysis results will be affected by this command in a multi-rail design:

- average power report
- time-based power report
- power related attributes: *total_power*

dynamic_power
internal_power
switching_power
leakage_power
x_transition_power
glitch_power
peak_power

Use the **current_power_rail all** command to resume the default behavior, which includes all the power rails in the design.

EXAMPLES

The following example shows generating rail-based power analysis results. The first **report_power** command will generate the power analysis results for the part of design with supply voltage of VDD1 and VDD2. The second **report_power** command will generate the results for the part of design which is supplied by VDD1.

```
pt_shell> create_power_rail_mapping VDD1 -cells block1
pt_shell> create_power_rail_mapping VDD2 -cells block2
pt_shell> current_power_rail {VDD1 VDD2}
pt_shell> report_power
pt_shell> current_power_rail VDD1
pt_shell> report_power
```

SEE ALSO

```
create_power_rail_mapping(2)
report_power_rail_mapping(2)
update_power(2)
report_power(2)
report_power_calculation(2)
```


current_scenario

Selects a subset of the scenarios in the current session for analysis.

SYNTAX

```
int current_scenario  
[-all]  
[scenario_list]
```

Data Types

scenario_list list

ARGUMENTS

[scenario_list]
 A list of scenarios in the current session to analyze.

[-all]
 Selects all scenarios in the current session for analysis.

DESCRIPTION

This command is only available if you invoke the **pt_shell** command with the the *-multi_scenario* option.

In multi-scenario analysis, after the scenarios have been created, the **current_session** command is used to focus on a set of scenarios for analysis. The **current_scenario** command is used to focus the analysis on a specific subset of the scenarios in the current session, only those scenarios selected will receive subsequent commands.

To determine which scenarios are in the current session and their focus, use the **report_multi_scenario_design** command with the *-session* option.

EXAMPLES

In the following example, three scenarios named scen1, scen2 and scen3 are created.

scen1 is created using common_s1.pt and specific_s1.pt scen2 is created using common_s2.pt and specific_s2.pt scen3 is created using common_s3.pt and specific_s3.pt

The scenarios scen1, scen2 and scen3 are selected for the current session which is then displayed. Notice that all three scenarios are in command focus.

The analysis is focused on only scen2 and scen3 using the **current_scenario** command and the session is displayed again. Notice that scen1 is now only in session focus but scen2 and scen3 are in command focus. i.e. only scen2 and scen3 will be considered for analysis.

```

1
pt_shell> create_scenario -name scen1 -common_data {common_s1.pt}
          -specific_data {specific_s1.pt}
1
pt_shell> create_scenario -name scen2 -common_data {common_s2.pt}
          -specific_data {specific_s2.pt}
1
pt_shell> create_scenario -name scen3 -common_data {common_s3.pt}
          -specific_data {specific_s3.pt}
1
pt_shell> current_session {scen1 scen2 scen3}
1
pt_shell> report_multi_scenario_design -session

```

```

*****
Report : multi_scenario_design
Version: V-2004.06
Date   : Tue Apr 13 05:50:10 2004
*****

```

Session details

Number of scenarios in current session: 3

Focus	Scenario	(Image provider)
Command	scen1	(scen1)
Command	scen2	(scen2)
Command	scen3	(scen3)

```

1
pt_shell> current_scenario {scen2 scen3}
1
pt_shell> report_multi_scenario_design -session

```

```

*****
Report : multi_scenario_design
Version: V-2004.06-Beta2
Date   : Tue Apr 13 05:51:53 2004
*****

```

Session details

Number of scenarios in current session: 3

Focus	Scenario	(Image provider)
Session	scen1	(scen1)
Command	scen2	(scen2)
Command	scen3	(scen3)

current_scenario

1

SEE ALSO

`current_session(2)`
`remove_scenario(2)`
`report_multi_scenario_design(2)`

current_session

Selects a set of scenarios for analysis.

SYNTAX

string **current_session** [*scenario list*] [-all]

ARGUMENTS

[*scenario_list*]

A list of scenarios to be brought into focus for analysis.

[-all]

All defined scenarios to be brought into focus for analysis.

DESCRIPTION

This command is only available if the user invokes pt_shell with the -multi_scenario option.

In order to set the current session, the following resources are required

- at least one PrimeTime license for slave usage. The number of PrimeTime licenses available for slave usage and be queried using the **report_multi_scenario_design** command.
- at least one host has been added. The number of hosts that have been added can be queried using the **report_distributed_hosts** command.
- at least one host is online. The number of hosts that have come online can be queried using the **report_distributed_hosts** command.

Once all the resource requirements are available and the scenarios created, the user must select a set of scenarios to analyse using the **current_session** command. Calling this command with a list of scenarios brings those scenarios into focus for the current session (session focus) and into focus for receiving commands (command focus) from the master. The current session can be set with all or a subset of the created scenarios.

The current_session command returns a collection containing the names of all the scenarios in the current_session.

EXAMPLES

In the following example, a configuration and three scenarios named scen1, scen2 and scen3 are created. The current_session is then set such that scen2 and scen3 are in session and command focus.

scen1 is created using common_s1.pt and specific_s1.pt scen2 is created using common_s2.pt and specific_s2.pt scen3 is created using common_s3.pt and specific_s3.pt

current_session

The scenarios scen2 and scen3 are brought into focus.

```
pt_shell> create_scenario -name scen1 -common_data {common_s1.pt} \  
          -specific_data  {specific_s1.pt}  
1  
pt_shell> create_scenario -name scen2 -common_data {common_s2.pt} \  
          -specific_data  {specific_s2.pt}  
1  
pt_shell> create_scenario -name scen3 -common_data {common_s3.pt} \  
          -specific_data  {specific_s3.pt}  
1  
pt_shell> current_session {scen2 scen3}  
{"scen2", "scen3"}
```

SEE ALSO

`current_scenario` (2), `remove_scenario`(2), `report_multi_scenario_design` (2),
`report_distributed_hosts` (2)

define_design_mode_group

Defines a design mode group with a set of design modes.

SYNTAX

```
string define_design_mode_group  
[-group_name name]  
mode_list
```

Data Types

<i>name</i>	string
<i>mode_list</i>	list

ARGUMENTS

-group_name *name*
Specifies the name of the design mode group. By default, if no design mode group is specified, a default design mode group is created and named "default".

mode_list
Specifies a list of design modes to be created and included in the design mode group.

DESCRIPTION

The **define_design_mode_group** command defines a set of design modes to be placed in a design mode group for your design. You select one of the design modes to be the active mode using the **set_mode -type design** command. There are only two valid states for a design mode group. The default state of all modes enabled or the state of one mode enabled and all others disabled. Setting one of the modes to be active automatically sets the rest of the modes in the group inactive (disabled).

You can map cell modes or paths to a design mode using the **map_design_mode** command. When you select the active design mode with the **set_mode -type design** command, the modes of the cells specified for that design mode are also active. Any paths mapped to disabled design modes automatically become false paths.

Unlike design modes, which are user-specified, cell modes are predefined, and are in the library. You can find out what cell modes are available using the **report_mode** command. For more information about cell modes, see the manual page for the **set_mode** command.

To see a report of the design modes specified for the current design, use the **report_mode -type design** command.

To remove design modes, use the **remove_design_mode** command. Removing a design mode also removes any cell mode or path specifications previously mapped to the design mode using the **map_design_mode** command.

EXAMPLES

The following example defines two design modes for the current design: SYSTEM and TEST.

```
pt_shell> define_design_mode_group {SYSTEM TEST}
```

The following example defines two independent design mode groups. The first command defines a design mode group named RW that contains design modes READ and WRITE. The second command defines a design mode group named LT that contains design modes LATCH and TRANSPARENT.

```
pt_shell> define_design_mode_group -group RW {READ WRITE}  
pt_shell> define_design_mode_group -group LT {LATCH TRANSPARENT}
```

The following command enables the design mode READ and disables the design mode WRITE, because READ and WRITE belong to the same design mode group RW.

```
pt_shell> set_mode -type design READ
```

SEE ALSO

```
remove_design_mode(2)  
map_design_mode(2)  
report_mode(2)  
reset_mode(2)  
set_mode (2)
```

define_qtm_attribute

Defines a new user-defined attribute for a class of Quick Timing Model (QTM) objects.

SYNTAX

```
string define_qtm_attribute  
-type data_type  
-class obj_class  
attr_name
```

Data Types

<i>data_type</i>	string
<i>obj_class</i>	string
<i>attr_name</i>	string

ARGUMENTS

-type *data_type*
Specifies the data type of the attribute. The supported data types are string, int, float, and Boolean.

-class *obj_class*
Specifies the class of QTM objects the new attribute is defined for. The valid object classes are lib, lib_cell or lib_pin. Attribute for 'lib' class applies to the library. Attribute for 'lib_cell' class applies to the QTM cell. Attribute for 'lib_pin' class applies to the QTM ports/pins.

attr_name
Specifies the name of the attribute.

DESCRIPTION

Use **define_qtm_attribute** command to define a new user-defined attribute that is applicable to QTM objects. A user-defined attribute is any attribute which PrimeTime does not understand by default. Those attributes already understood or reserved by PrimeTime for various classes of objects are considered application attributes and should not be re-defined. If you need to set the application attributes that are applicable to the classes of QTM objects, just use **set_qtm_attribute** directly. There is no need to define the attribute before setting them. But for user attributes that are not reserved by the application, you have to define them before set them on objects. Note that the **list_attributes** command will not show any attributes defined for QTM.

After save the QTM in proper library, those user-defined QTM attributes can be imported from db files. But you must define those user attributes you want to import with the **define_user_attribute** command and the **-import** option before the library

with the QTM cell is read.

For more information, see the **define_user_attribute** man page.

EXAMPLES

The following example defines QTM attributes of various types.

```
pt_shell> define_qtm_attribute my_str_attr \  
?          -class lib_pin -type string  
pt_shell> define_qtm_attribute my_int_attr \  
?          -class lib_cell -type int  
pt_shell> define_qtm_attribute my_float_attr \  
?          -class lib -type float
```

SEE ALSO

```
set_qtm_attribute(2)  
remove_qtm_attribute(2)  
define_user_attribute(2)
```

define_scaling_lib_group

Defines a group of libraries to support voltage and/or temperature scaling.

SYNTAX

```
string define_scaling_lib_group  
library_list
```

Data Types

```
library_list      list
```

ARGUMENTS

```
library_list  
    A list of libraries to support voltage and/or temperature scaling.
```

DESCRIPTION

The **define_scaling_lib_group** command defines a group of libraries that PrimeTime interpolates between for voltage and/or temperature scaling.

The command must be issued after the design is read and linked. If the design is not already linked, this command auto-links the design.

Only one of the libraries in the *library_list* option should be in the **link_path** command; the remaining libraries will be read automatically after the design is linked to complete the group.

You can have more than one group to cover different portions of a design, but the groups cannot share libraries.

Once a group is in effect, any attempt to use the **set_rail_voltage** command and/or the **set_operating_condition** command outside the range of the applicable group results in a **DEL-012** error.

You can see if a group was used for delay calculation by using the **report_delay_calculation** command. The report shows the results using the link library, then the scaling library group results if applicable and valid.

A minimum of two libraries is required for one-dimensional scaling (i.e. voltage **or** temperature), and a minimum of four libraries is required for two-dimensional scaling (i.e. voltage **and** temperature).

Presently scaling with library groups is only supported for nonlinear delay model (NLDM) constraints, RC delay calculations with Composite Current-Source (CCS) data, and mixed CCS/NLDM data.

The **report_lib_groups** command can be used to report all the scaling groups defined by the **define_scaling_lib_group** command.

EXAMPLES

The following example defines a scaling library group for two voltage domains about the nominal:

```
pt_shell> set link_path "*" 1.1.db"
pt_shell> link_design
pt_shell> define_scaling_lib_group { 0.9.db 1.1.db 1.3.db }
```

SEE ALSO

```
link_design(2)
set_rail_voltage(2)
set_operating_conditions(2)
report_delay_calculation(2)
report_lib_groups(2)
```

define_user_attribute

Defines a new user-defined attribute.

SYNTAX

```
string define_user_attribute  
-type data_type  
-classes class_list  
[-range_min min]  
[-range_max max]  
[-one_of values]  
[-import]  
[-quiet]  
attr_name
```

Data Types

<i>data_type</i>	string
<i>class_list</i>	list
<i>min</i>	double
<i>max</i>	double
<i>values</i>	list
<i>attr_name</i>	string

ARGUMENTS

-type *data_type*
Specifies the data type of the attribute. The supported data types are string, int, float, double, and boolean.

-classes *class_list*
Defines the attribute for one or more of the classes. The valid object classes are design, port, cell, pin, net, lib, lib_cell or lib_pin.

-range_min *min*
Specifies *min* value for numeric ranges. This is only valid when the *data_type* is int or double. Specifying a minimum constraint without a maximum constraint creates an attribute which accepts a value $\geq min$.

-range_max *max*
Specifies *max* value for numeric ranges. This is only valid when the *data_type* is int or double. Specifying a maximum constraint without a minimum constraint creates an attribute which accepts a value $\leq max$.

-one_of *values*
Provides a list of allowable strings. This is only valid when the data type is string.

-import
Import this attribute from a design or library database.

define_user_attribute

-quiet

Does not report any messages.

attr_name

Specifies the name of the attribute.

DESCRIPTION

Use the **define_user_attribute** command to define a new user-defined attribute. A user-defined attribute is any attribute which PrimeTime does not understand by default. The **list_attributes** command will show all attributes which PrimeTime understands.

You can apply attributes to most object classes in PrimeTime. You can use these to mark interesting cells or nets, or store values you computed, and so on. PrimeTime cannot use these attributes, but you can use them in scripts, procedures, and so on. You can list the attributes you defined using the **list_attributes** command.

User-defined attributes can be imported from a design or library database once they have been defined. A design or library database is a db or ddc format file, or a Milkyway database. You must define attributes you want to import with the **-import** option. Note that imported attributes do not appear on the design objects until the design is linked. Attributes imported from a design or library database will be inherited through the hierarchy. Note that attributes defined for designs will be inherited onto cells, and attributes defined for ports will be inherited onto pins. The **define_user_attribute** command will set up these relationships for you automatically if you do not do it explicitly.

EXAMPLES

The following example defines attributes of various types. Note that more than one class can be specified. Also note that attr_i is defined as imported from a design or library database. Finally, attribute design1 is imported from a design or library database, and PrimeTime defines it for cells as well.

```
pt_shell> define_user_attribute attr_s \
?          -class {cell net} -type string
pt_shell> define_user_attribute attr_i \
?          -class cell -type int -import
pt_shell> define_user_attribute design1 \
?          -class design -type int -import
Information: Inferred definition of attribute 'design1' for class 'cell'
             because it is imported for class 'design' (ATTR-7)
```

In the following example, attr_ir1 is defined as ≥ 0 and ≤ 100 , attr_ir2 is defined as ≥ 0 with no maximum, and attr_ir3 is defined as ≤ 100 with no minimum.

```
pt_shell> define_user_attribute attr_ir1 \
?          -class cell -type int \
?          -range_min 0 -range_max 100
pt_shell> define_user_attribute attr_ir2 \
```

```
?          -class cell -type int -range_min 0
pt_shell> define_user_attribute attr_ir3 \
?          -class cell -type int -range_max 100
```

In the following example, define attribute attr_oo for cells. The attribute is a string, but can only be set to A, B, C, or D.

```
pt_shell> define_user_attribute attr_oo \
?          -class cell -type string -one {A B C D}
```

After these attributes have been defined, you can list the attribute definitions using the **list_attribute** command. Notice how the 'attr_i' attribute is marked as importable from db, meaning a design or library database.

```
pt_shell> list_attributes
*****
Report : List of Attribute Definitions
Design :
*****
```

```
Properties:
  A - Application-defined
  U - User-defined
  I - Importable from db (for user-defined)
```

Attribute Name	Object	Type	Properties	Constraints
attr_i	cell	int	U,I	
attr_ir1	cell	int	U	0 to 100
attr_ir2	cell	int	U	>= 0
attr_ir3	cell	int	U	<= 100
attr_oo	cell	string	U	A, B, C, D
attr_s	cell	string	U	
attr_s	net	string	U	
design1	design	int	U,I	
design1	cell	int	U	

SEE ALSO

```
get_attribute(2)
list_attributes(2)
read_db(2)
read_ddc(2)
read_milkyway(2)
remove_user_attribute(2)
report_attribute(2)
set_user_attribute(2)
```

derive_clocks

Creates clocks on source pins in design.

SYNTAX

```
string derive_clocks  
-period period_value  
[-waveform edge_list]
```

Data Types

<i>period_value</i>	float
<i>edge_list</i>	list

ARGUMENTS

-period *period_value*
Specifies the clock period of the automatically derived clocks. The clock period has a value greater than or equal to zero (value ≥ 0).

-waveform *edge_list*
Specifies the rise and fall edge times of the clock, in library time units, over an entire clock period. It defines the clock edge specification. The first time that is listed is a rising transition; typically the first rising transition after time zero. There must be an even number of increasing times and alternating rise and fall times. If you do not specify an *edge_list* value, the command assumes a default waveform that has a rise edge of 0.0 and a fall edge of $\text{period\_value}/2$.

DESCRIPTION

Creates clocks on source pins in design. This command finds the clock source ports and pins that must have clocks defined so that every register clock pin has a clock. Then on each source set, the command creates a clock in the same way as the **create_clock** command does. The clocks are created with the specified waveform or period. For a description of commands that use clock objects, see the **create_clock** command man page.

EXAMPLES

The following example creates a clock on the source pin with a clock period of 20:

```
pt_shell> derive_clocks -period 20
```

SEE ALSO

`create_clock(2)`
`get_clocks(2)`
`group_path(2)`
`remove_clock(2)`
`report_clock(2)`
`set_clock_latency(2)`
`set_clock_uncertainty(2)`
`set_input_delay(2)`
`set_output_delay(2)`

disconnect_net

Disconnects a net from specified pins or ports, or from all pins and ports.

SYNTAX

```
int disconnect_net net object_spec | -all
stringnet
list object_spec
```

ARGUMENTS

-all

Indicates that all pins and ports are to be disconnected from *net*. **-all** and *object_spec* are mutually exclusive.

net

Specifies the name of the net to be disconnected.

object_spec

Specifies a list of pins or ports to be disconnected from *net*. **-all** and *object_spec* are mutually exclusive.

DESCRIPTION

The **disconnect_net** command disconnects pins and ports from a *net* in the current design. Like all other netlist editing commands, for **disconnect_net** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **disconnect_net** returns a 1 if successful and a 0 if unsuccessful.

The *net* and each pin and port specified in the *object_spec* must be in scope; that is, at or below the current instance. There are three rules for **disconnect_net**:

- You cannot disconnect an unconnected pin or a port.
- You cannot disconnect a pin or a port that is not connected to *net*.
- You cannot disconnect a hierarchical pin or a port that has the same name as *net*.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design

cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following example, the first command attempts to connect port in1 to net new_net1, but in1 is already connected to a net. The second command determines the net to which in1 is connected; the third command disconnects the port from its net. The fourth and final command connects in1 to new_net1.

```
pt_shell> connect_net new_net1 [get_ports in1]
Error: Could not connect 'in1' to 'new_net1'
```

```
Object is already connected - disconnect it first (NED-042)
Error: No changes made. (NED-040)
0
pt_shell> set pnet [get_nets -of_objects [get_ports in1]]
{"in1"}
pt_shell> disconnect_net $pnet [get_ports in1]
Information: Disconnected 'in1' from 'net1'. (NED-019)
1
pt_shell> connect_net new_net1 [get_ports in1]
Information: Connected 'in1' to 'new_net1'. (NED-018)
1
```

SEE ALSO

connect_net (2), **create_cell** (2), **create_net** (2), **size_cell** (2), **swap_cell** (2), **write_changes** (2).

drive_of

Determines the drive resistance of the specified library cell pin. The **drive_of** command is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
float drive_of  
[-rise] [-fall]  
[-wire_drive]  
[-piece val]  
lib_cell_pin  
  
stringval  
stringlib_cell_pin
```

ARGUMENTS

-rise
Get rise drive value.

-fall
Get fall drive value.

-wire_drive
Not supported by PrimeTime.

-piece val
Not supported by PrimeTime.

lib_cell_pin
Specifies the name of the library cell pin for which to get the drive resistance, or a collection that contains the library cell pin.

DESCRIPTION

The **drive_of** command returns the drive resistance of the given library cell pin. If neither **-rise** nor **-fall** is specified, the greater value of rise and fall is returned.

The **drive_of** command exists in PrimeTime for compatibility with Design Compiler. Complete information about the **drive_of** command can be found in the Design Compiler documentation. The supported method for getting the capacitance of a library cell pin is by using the **get_attribute** command with either the **drive_resistance_rise** or **drive_resistance_fall** attribute.

SEE ALSO

get_attribute (2).

estimate_clock_network_power

Virtually generate a clock tree and estimate its power.

SYNTAX

```
string estimate_clock_network_power  
lib_cell  
[-max_fanout fanout]  
[-wire_load_model wire_load_name]  
[-library lib_name]  
[-input_transition transition]  
[-clocks clock_list]  
[-include_registers]  
[-ignore_clock_gating]
```

Data Types

<i>lib_cell</i>	string
<i>fanout</i>	int
<i>wire_load_name</i>	string
<i>lib_name</i>	string
<i>transition</i>	float
<i>clock_list</i>	list

ARGUMENTS

lib_cell
Specifies the buffer or inverter to be used to build the clock tree.

-max_fanout fanout
Specifies the maximum fanout number that the specified buffer can drive. The default number is 32.

-wire_load_model wire_load_name
Specifies the wire load model to be used to compute the wire capacitance for the nets in the clock tree. The default is the wire load model set to the design.

-library lib_name
Specifies the library where the specified wire load model is defined.

-input_transition transition
Specifies the clock input transition. If not specified, use the clock transition annotated to the clock by the **set_clock_transition** command. If there is no annotation either, the default to 0.

-clocks clock_list
Specifies the clocks to be estimated for clock tree power. The default is all the clocks in the design.

-include_registers
Indicates the power of registers driven by the clock is also included in the

power report.

`-ignore_clock_gating`

Indicates that existing clock tree cells like the clock gating cells are ignored when building the clock tree.

DESCRIPTION

The **`estimate_clock_network_power`** command virtually creates a clock tree for each clock and calculates power for the generated clock tree. The clock trees are built on the fly and the original design is not touched or modified. The following describes how the clock tree is built:

Find all the nets in the clock network. For each net in the clock network, insert buffers to build a clock tree, so that the fanout of each buffer in the clock tree is no bigger than the max fanout. The command tries to build a tree as balanced as possible such that the delays from the root to the leafs do not vary a lot. All the driven registers are put at the same and lowest level of the tree. The deviation of buffer fanouts at the same tree level is kept as small as possible.

The output load of each buffer is calculated using wire load model. The input transition of each buffer propagates from the root of the tree. The switching activity for each inserted buffer is equal to clock frequency or from annotation (SAIF or `set_switching_activity`).

If there are clock gating cells in the original clock network, the command builds a separate tree based on the fanout of the clock gating cell using the same rule for the clock tree, and this separate tree is considered as a branch of the whole clock tree, and also balances with other branches. Then the clock gating effect is accounted for by the annotated or, if not annotated, propagated switching activity. In order for you to investigate savings from clock gating, the command has the `-ignore_clock_gating` option to estimate the clock tree power by ignoring the clock gating cells and compare with the power that considers the clock gating effect. The power consumed by the registers driven by the clock is also included in the report if the `-include_registers` option is specified.

EXAMPLES

In the following example, clock tree power is estimated for clock CLK by building the clock tree using the buffer `buf1a1` of library `ssc_core_typ` and wire load model in the design is used to calculate the wire capacitance. The clock input transition is 0.2.

```
pt_shell> create_clock -name CLK -period 10 clk
pt_shell> set_clock_transition 0.2 CLK
pt_shell> estimate_clock_network_power ssc_core_typ/buf1a1
```

SEE ALSO

`report_power(2)`

estimate_eco

Estimate delay changes for the **size_cell** and **insert_buffer** commands.

SYNTAX

```
int estimate_eco
[-max]
[-min]
[-rise]
[-fall]
[-type size_cell / insert_buffer]
[-
sort_by area / stage_delay / arrival / slack / transition / max_transition / min_tr
ansition / max_capacitance / min_capacitance]
[-verbose]
[-nosplit]
[-significant_digits]
[-inverter_pair]
[-current_library]
[-libraries lib_spec]
[-lib_cells lib_cell_list]
object_list
```

Data Types

<i>lib_spec</i>	string
<i>lib_cell_list</i>	list

ARGUMENTS

-min2	Indicates that minimum delay calculation is to be shown.
-max	Indicates that maximum delay calculation is to be shown.
-rise	Indicates that stage driver output pin is rising.
-fall	Indicates that stage driver output pin is falling.
-verbose	Indicates verbose message is to be shown.
-nosplit	Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The <i>-nosplit</i> option prevents line-splitting and facilitates writing software to extract information from the report output.

-type *estimation_type*
 Specifies either *size_cell* or *insert_buffer* ECO estimation type is to be performed. If not specified, the default type is *size_cell*.

-sort_by *option*
 Specifies *area*, *stage_delay*, *arrival*, *slack*, *transition*, *max_transition*, *min_transition*, *max_capacitance*, or *min_capacitance* to sort library cells to be reported. If not specified, the default type is *area*.

-significant_digits *digits*
 Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-inverter_pair
 Specifies inverter pair is to be inserted when *eco_type* is *insert_buffer*.

-current_library
 Only search the cell's current library to resolve the *lib_cell* name.

-libraries
 Resolves *lib_cell* from the libraries specified this option only.

-lib_cells *lib_cell_name*
 Specifies the list of library cells to be used for ECO.

object_name
 Specifies the name of object. For **size_cell**, the object should be a cell. For **insert_buffer**, the object should be a pin.

DESCRIPTION

The **estimate_eco** command provides a method to estimate the timing effects of certain ECO commands, without actually incurring a timing update to compute the slack. As its name implies, the results of this command are estimates only, and are expected to reasonably predict, but not necessarily match, the actual slack resulting from the ECO command.

ECO estimation is supported for two ECO commands: **size_cell** and **insert_buffer**. This is specified with the required **-type** argument, which takes a value of *size_cell* or *insert_buffer*, which matches the corresponding command name. Both buffer insertion and double-inverter insertion are supported for the **insert_buffer** estimation.

When performing **insert_buffer** estimation, the **-lib_cell** option is required and must supply a list of one or more buffer library cells as buffer insertion candidates. The buffer cells can be specified in the form of *lib_cell* collection objects, a list of *library_name/base_name* names, or simply a list of *base_name* library cell names. The library cell specifications follow the same rules as the **insert_buffer** command. For more information, see the **insert_buffer** man page.

When performing **size_cell** estimation, the **-lib_cell** option is optional and defaults to the list of alternative library cells for the specified instance cell as obtained by the **get_alternative_lib_cells** command. When the **-lib_cell** option is specified, the candidate library cells can be specified in the form of lib_cell collection objects, a list of library_name/base_name names, or simply a list of base_name library cell names. The library cell specifications follow the same rules as the **size_cell** command. For more information, see the **size_cell** man page.

For both commands, if you want to limit the candidate cells to a specific library, you can use the **-current_library** and **-libraries** options of the **get_alternative_lib_cells** command, and pass the resulting lib_cell collection to the **estimate_eco** command.

The **estimate_eco** command generates a report that predicts the timing and slack effects of one or more potential library cell candidates for the **size_cell** or **insert_buffer** command. By default, the estimates are shown in a compact form that lists one candidate library cell per line, with stage delay, arrival and slack information columns. A verbose mode is always available with the **-verbose** option. In verbose mode, a separate detailed estimation report is shown for each candidate library cell.

The **-min** and **-max** options are available to specify whether minimum-delay hold timing, maximum-delay setup timing, or both should be shown. By default, the maximum-delay timing is shown. The **-rise** and **-fall** options are available to specifically request rise/fall timing. By default, both rise and fall information is shown. If only the **-rise** or **-fall** option is specified, only that information will be shown.

The estimate computation takes into account the incoming transition times, the output loading, the fanout loading, the detailed parasitics (if present) and the driver characteristics of the candidate cells. The rise/fall timing printed in the report represents the worst timing through the cell output pins in the corresponding direction. In the verbose report, the arrival is the arrival time at the input pin of the load cell driven by the resized or inserted cell, and the stage delay is the delay of the inserted/resized cell and the driven net.

In the compact report without the verbose option, the reported arrival time, slack and transition time are the estimated values at the input pin of the load cell. Maximum transition, minimum transition, maximum capacitance, and minimum capacitance values are the worst value of the pins in the stage. Individual values can be seen in the verbose reporting.

To perform slack and arrival estimations, the **timing_save_pin_arrival_and_slack** variable should be set to *true* when this command is issued. If the **timing_save_pin_arrival_and_slack** variable has not been set to *true*, before the command executes it automatically sets it to *true* and updates the design timing. If you intend to use this command, it is recommended that you set this variable to *true* before the first timing update, thus preventing the cost of an additional timing update.

Estimated data can be selectively reported if the content of the **eco_estimation_output_columns** variable is changed. By default, only *area*, *stage_delay*, *arrival*, and *slack* are reported. Others, such as *max_transition* and *max_capacitance* can be reported by changing the variable. For more information, see the **eco_estimation_output_columns** man page.

EXAMPLES

The following example shows a minimum and maximum delay summary estimation of four potential buffer cell insertions at an instance pin:

```
pt_shell> estimate_eco \  
-type insert_buffer \  
-min -max \  
-lib_cell {BUFX2 BUFX4 DLY1X1 DLY2X1} \  
wishbone/TxDone_wb_reg/D
```

delay type : max_rise

lib cell	area	stage_delay	arrival	slack
slow/DLY1X1	19.96	0.857	2.076	14.859
slow/DLY2X1	19.96	1.218	2.437	14.498
slow/BUFX4	16.63	0.639	1.858	15.077
slow/BUFX2	13.31	0.647	1.867	15.069
*No buffer	0.00	0.491	1.711	15.225

delay type : max_fall

lib cell	area	stage_delay	arrival	slack
slow/DLY1X1	19.96	0.746	1.966	14.767
slow/DLY2X1	19.96	1.088	2.308	14.425
slow/BUFX4	16.63	0.514	1.734	14.999
slow/BUFX2	13.31	0.534	1.754	14.979
*No buffer	0.00	0.354	1.573	15.160

delay type : min_rise

lib cell	area	stage_delay	arrival	slack
slow/DLY1X1	19.96	0.366	2.076	0.296
slow/DLY2X1	19.96	0.727	2.437	0.657
slow/BUFX4	16.63	0.148	1.858	0.077
slow/BUFX2	13.31	0.156	1.867	0.086
*No buffer	0.00	0.000	1.711	-0.070

delay type : min_fall

lib cell	area	stage_delay	arrival	slack
slow/DLY1X1	19.96	0.679	1.966	0.098
slow/DLY2X1	19.96	1.021	2.308	0.440
slow/BUFX4	16.63	0.447	1.734	-0.134
slow/BUFX2	13.31	0.467	1.754	-0.114
*No buffer	0.00	0.286	1.573	-0.294

The following example shows a minimum delay verbose estimation of a single potential buffer cell insertion at an instance pin:

```
pt_shell> estimate_eco \  
-type insert_buffer \  
-min \  
-verbose \  

```

```
-lib_cell {DLY1X1} \
wishbone/TxDone_wb_reg/D
```

```
cell name      : wishbone/TxDoneSync1_reg
current lib cell: slow/DFFRHQX1
```

```
buffer lib cell: slow/DLY1X1
```

min_rise	current	estimate
input net delay	0.012	0.012
stage delay	0.000	0.366
cell delay	0.000	0.000
output net delay	0.000	0.000
buffer delay	0.000	0.366
delta delay	0.000	0.000
arrival time	1.711	2.076
slack	-0.070	0.296

```
buffer lib cell: slow/DLY1X1
```

min_fall	current	estimate
input net delay	0.011	0.011
stage delay	0.286	0.679
cell delay	0.286	0.286
output net delay	0.000	0.000
buffer delay	0.000	0.392
delta delay	0.000	0.000
arrival time	1.573	1.966
slack	-0.294	0.098

The following example shows a maximum delay summary estimation of potential cell instance resizes for a given inverter cell name pattern:

```
pt_shell> estimate_eco \
  -type size_cell \
  -lib_cell {INV*} \
  wishbone/bd_ram/U9/U6200
```

delay type : max_rise	lib cell	area	stage_delay	arrival	slack
slow/INVX20	63.20	1.062	9.340	0.019	
slow/INVX16	56.55	1.173	9.451	-0.092	
slow/INVX12	43.24	1.347	9.624	-0.266	
slow/INVX8	19.96	1.682	9.961	-0.602	
*slow/CLKINX4	13.31	2.727	11.005	-1.646	
slow/INVX3	13.31	3.423	11.701	-2.342	
slow/INVX4	13.31	2.770	11.048	-1.689	
slow/INVX2	9.98	4.943	13.220	-3.861	
slow/INVX1	6.65	9.269	17.546	-8.187	
slow/INVXL	6.65	13.239	21.516	-12.157	

```
delay type : max_fall
```

```
estimate_eco
```

lib cell	area	stage_delay	arrival	slack

slow/INVX20	63.20	0.855	7.542	1.843
slow/INVX16	56.55	0.940	7.627	1.758
slow/INVX12	43.24	1.074	7.761	1.624
slow/INVX8	19.96	1.279	7.967	1.418
*slow/CLKINX4	13.31	2.731	9.418	-0.033
slow/INVX3	13.31	2.278	8.965	0.420
slow/INVX4	13.31	1.901	8.588	0.797
slow/INVX2	9.98	3.192	9.879	-0.494
slow/INVX1	6.65	5.420	12.107	-2.722
slow/INVXL	6.65	7.474	14.161	-4.776

The following example shows a maximum delay verbose estimation of potential cell instance resize for specific user-specified alternative cells:

```
pt_shell> estimate_eco \
  -type size_cell \
  -verbose \
  wishbone/bd_ram/U9/U6200 \
  -lib_cell {INVX20}
```

```
cell name      : wishbone/bd_ram/U9/U6200
current lib cell: slow/CLKINX4
```

```
new lib cell: slow/INVX20
```

max_rise	current	estimate

area	13.310	63.200
input net delay	0.005	0.005
stage delay	2.727	1.062
cell delay	2.609	0.957
output net delay	0.118	0.105
delta delay	0.000	0.000
arrival time	11.005	9.340
slack	-1.646	0.019

```
new lib cell: slow/INVX20
```

max_fall	current	estimate

area	13.310	63.200
input net delay	0.005	0.005
stage delay	2.731	0.855
cell delay	2.612	0.751
output net delay	0.118	0.104
delta delay	0.000	0.000
arrival time	9.418	7.542
slack	-0.033	1.843

The following example shows transition and maximum transition only by changing the variable.

```
pt_shell> set eco_estimation_output_columns "transition max_transition"
pt_shell> estimate_eco \
    -type size_cell \
    -max -rise \
    ram/U9/U6200
```

```
delay type : max_rise
lib cell          trans      max_trans
-----
slow/INVSX20      9.340      0.019
slow/INVSX16      9.451     -0.092
slow/INVSX12      9.624     -0.266
slow/INVSX8       9.961     -0.602
*slow/CLKINVSX4   11.005     -1.646
slow/INVSX3       11.701     -2.342
slow/INVSX4       11.048     -1.689
slow/INVSX2       13.220     -3.861
slow/INVSX1       17.546     -8.187
slow/INVSXL       21.516    -12.157
```

The following example shows how the slack value can be parsed from any estimate_eco verbose report for a given min/max rise/fall condition:

```
# specify lib_cell and design instance
set lib_cell INVSX20
set instance wishbone/bd_ram/U9/U6200

# obtain all min/max rise/fall estimation info for a single ECO
redirect -variable rpt {estimate_eco -type size_cell -min -max -rise -fall -
verbose -lib_cell $cell $instance}

# specify min/max rise/fall conditions of interest
set mm max
set rf rise

# obtain min/max rise/fall subreport
regexp "({mm}_{rf}.+?)\n\n" $rpt dummy subrpt

# obtain slack from subreport
regexp {slack +[^\s]+\s+([^\s]+)} $subrpt dummy slack
echo $slack
```

This report parsing example can be used with both size_cell and insert_buffer estimation.

SEE ALSO

```
get_alternative_lib_cells(2)
report_timing(2)
report_delay_calculation(2)
report_constraints(2)
eco_estimation_output_columns(3)
```

extract_model

Generates a timing/power model for a design from its gate-level netlist.

SYNTAX

```
string extract_model
[-context_borrow]
[-latch_level levels]
-output file_name
[-format format_list]
[-parasitic_format format_list]
[-library_cell]
[-remove_internal_arcs]
[-ignore_boundary_parasitics]
[-test_design]
[-arc_types arc_list]
[-noise]
[-power]
[-block_scope]
[-block_scope_only]
[-scope_scenario scenario_name]
[-validate valid_list]
```

Data Types

<i>levels</i>	int
<i>file_name</i>	string
<i>format_list</i>	list
<i>arc_list</i>	list

ARGUMENTS

-context_borrow

Indicates that PrimeTime is to determine which latches on an interface borrow, based on the input port arrival times and clock definitions specified at the time of model extraction. The actual time borrowed by a latch is not used by model extraction; only whether a latch borrows or not. A borrowing latch is traced through during model extraction, while tracing stops at a non-borrowing latch. Use this option if your design contains latches on its interface. The *-context_borrow* and *-latch_level* options are mutually exclusive.

-latch_level *levels*

Specifies the number of levels of latch borrowing that are to occur at the interface of a design; the default is 0. The extracted model does not preserve all borrowing behavior up to the specified level, but only the specified borrowing behavior. Use this option only after you have verified that all latches at the interface of a design borrow if they are at a certain level. The *-context_borrow* and *-latch_level* options are mutually exclusive.

-output *file_name*

Specifies the root name to use in deriving the names of the generated output

files; the default is the current design name. The root name is appended with an appropriate extension (.db, .data, .mod, or .lib), depending on the value of the *-format* option.

-format format_list

Specifies a list of one or more output formats for the generated model. If the list contains more than one item, they must be enclosed in braces. Allowed values are as follows:

- **db** (the default), which specifies Synopsys DB format; the output file has the extension *_lib.db* and contains the DB library description of the core cell of the model.
- **lib**, which specifies Synopsys Library Compiler format; the output file has the extension *.lib*.

-parasitic_format format_list

Specifies a list of formats for the boundary parasitic file written for the model. The allowed values are *spef* (the default) for the SPEF format, and *sbpf* for the SBPF format.

-library_cell

Indicates that the model is to be generated as a library cell, named *current_design name*, rather than a design containing a core library cell. This option eliminates boundary nets, and the boundary parasitics become part of the model. You must use this option if you use the *-remove_internal_arcs* or the *-test_design* options; the *-test_design* option instantiates the library cell into a test design.

-remove_internal_arcs

Indicates that internal arcs are to be removed; only the pin-to-pin timing is to be preserved. Internal clocks are lost because internal arcs are needed to support them. The generated model contains only timing arcs that start or end at an input port or output port. If you use the *-remove_internal_arcs* option, you must also use the *-library_cell* option.

-ignore_boundary_parasitics

Indicates that detailed parasitic annotations present on the boundary nets of the design are not to be included in the model. If specified without the *-library_cell* option, the *-ignore_boundary_parasitics* command disables the writing of the SPEF file. If specified with the *-library_cell* option, the *-ignore_boundary_parasitics* option causes the effects of the boundary net parasitics to be excluded from the model.

-test_design

Indicates that a test design DB instantiating the model library cell is to be generated. The design name is *current_design_test* and the DB file is *current_design_test.db*. If you use the *-test_design* option, you must also use the *-library_cell* option.

-arc_types arc_list

Specifies a list of one or more arc types to be included when extracting a model; only the specified types are extracted. By default, all arc types are included. Allowed values are *min_seq_delay*, *max_seq_delay*, *min_combo_delay*, *max_combo_delay*, *setup*, *hold*, *recovery*, *removal*, and *pulse_width*. If the list

contains more than one item, they must be enclosed in braces.

-noise

Specifies that noise features should be added to the model. If the option is specified and no noise information is available in the cell's libraries and none has been specified by the user, then only steady-state resistance is written by using an estimation. This option will perform an implicit **update_noise** if necessary. This option is only support for the lib format.

-power

Specifies that power features should be added to the model. This option will perform an implicit **update_power** if necessary. So the variable `power_enable_analysis` has to be set to TRUE and you must have PrimeTime PX license. This option is only support for the lib format.

-block_scope

Indicates that the block scope information needs to be captured in addition to extracting the timing model. When specified, PrimeTime creates a separate file that stores the scope information based on the current constraint setup of the block for model extraction.

-block_scope_only

Indicates that only the scope information based on the current setup needs to be captured for the block, no need to actually extract the timing model. This option is useful for the later runs when an ETM was already generated but the scope of block-level validation has changed yet the original ETM is not impacted by the change and no need to be re-generated. When specified, PrimeTime creates a file contains scope information captured.

-scope_scenario *scenario_name*

Specifies the name of the scenario for which the scope data needs to be labeled and later checked for. The scope information captured and stored in the scope data file will be marked as corresponding to the given scenario. If not specified, a fixed default name is used internally.

-validate *valid_list*

Specifies what should be automatically validated after the model is created. The only allowed value is timing.

DESCRIPTION

The **extract_model** command generates a static timing model for the current design from its gate-level netlist. The generated model exhibits the same timing characteristics as the original design and therefore is used instead of the gate-level netlist for purposes of static timing analysis. Noise information and power information can also be included with the use to the **-noise** option and the **-power** option, respectively.

The model generated by the **extract_model** command is context-independent. That is, changes in clock waveforms, output capacitive loads, input drives, input delays, and output delays result in different timing exhibited by the model. The only exception occurs when the **-context_borrow** option is specified. In this case, the timing of the generated model, especially time borrowing information, is valid only for the clock waveforms, input delays and output delays specified when the model is extracted.

By default, the generated model is a design containing a single instance of the core cell. All boundary nets in the original design are preserved in the model. The name given to core lib_cell is *current_design_name_core* and the DB model design name is *current_design_name*. The design is written in db format, to the file *filename.db*, regardless of the values given to the *-format* option.

When scope of the block is captured, it is not impacted by the **hier_scope_check_defaults** environment variable. Instead, all applicable scoping information of the block based on its current constraint setup is captured and stored.

Also, the command can be used to generated scope information for the block that will be replaced with the ETM at top-level.

If the *-validate* option is given and the **hier_modeling_version** variable is set to 2.0, the model will be validated automatically after the model is created.

EXAMPLES

The following example generates a model in Synopsys DB format for the current design. The model is generated for the operating condition set on the design. In addition, the time borrowed by each level-sensitive latch is locked at its current value in the generated model. Two files are generated: *DMA_model.db* and *DMA_model_lib.db*.

```
pt_shell> extract_model -context_borrow -output DMA_model
```

The following example generates a model in Synopsys DB format for the current design. The generated model is a library cell contained in *model2_lib.db*. The file *model2_test.db* is also generated; this contains the test design that instantiates the model core cell, but it is not a part of the model. *model2_test.db* can be used to obtain timing and model reports in PrimeTime. Also, it creates scope information for the block.

```
pt_shell> extract_model -library_cell \  
-test_design -output model2 -format {db} -block_scope
```

SEE ALSO

```
extract_model_capacitance_limit(3)  
extract_model_clock_transition_limit(3)  
extract_model_data_transition_limit(3)  
extract_model_enable_report_delay_calculation(3)  
extract_model_gating_as_nochange(3)  
extract_model_lib_format_with_check_pins(3)  
extract_model_merge_clock_gating(3)  
extract_model_noise_iv_index_lower_factor(3)  
extract_model_noise_iv_index_upper_factor(3)
```

extract_model

extract_model_noise_width_points(3)
extract_model_num_capacitance_points(3)
extract_model_num_clock_transition_points(3)
extract_model_num_data_transition_points(3)
extract_model_num_noise_iv_points(3)
extract_model_num_noise_width_points(3)
extract_model_single_pin_cap(3)
extract_model_status_level(3)
extract_model_use_conservative_current_slew(3)
extract_model_with_3d_arcs(3)
extract_model_with_clock_latency_arcs(3)
timing_clock_gating_propagate_enable(3)
timing_disable_clock_gating_checks(3)
timing_enable_preset_clear_arcs(3)
hier_scope_check_defaults(3)
hier_modeling_version(3)
check_block_scope(2)

filter

The **filter** command, a synonym for the **filter_collection** command, is a Design Compiler emulation command provided for compatibility between PrimeTime and Design Compiler. You use the **filter_collection** command to filter an existing collection, resulting in a new collection. The base collection remains unchanged.

SEE ALSO

`filter_collection(2)`

filter_collection

Filters an existing collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection filter_collection  
base_collection expression  
[-regexp]  
[-nocase]
```

```
collectionbase_collection  
string      expression
```

ARGUMENTS

base_collection

Specifies the base collection to be filtered. This collection is copied to the result collection. Objects are removed from the result collection if they are evaluated as **false** by the conditional *expression* value. Substitute the collection you want for *base_collection*.

expression

Specifies an expression with which to filter *base_collection*. Substitute the string you want for *expression*.

-regexp

Specifies that the =~ and !~ filter operators will use real regular expressions. By default, the =~ and !~ filter operators use simple wildcard pattern matching with the * and ? wildcards.

-nocase

Makes the pattern match case-insensitive. When you specify this option, you must also specify the **-regexp** option.

DESCRIPTION

Filters an existing collection, resulting in a new collection. The base collection remains unchanged. In many cases, commands that create collections support a **-filter** option that filters as part of the collection process, rather than after the collection has been made. This type of filtering is almost always more efficient than using the **filter_collection** command after a collection has been formed. The **filter_collection** command is most useful if you plan to filter the same large collection many times using different criteria.

The **filter_collection** command results in either a new collection or an empty string. A resulting new collection contains the subset of the objects in the input *base_collection*. A resulting empty string (the empty collection) indicates that the *expression* filtered out all elements of the input *base_collection*.

The basic form of the conditional expression is a series of relations joined

together with AND and OR operators. Parentheses () are also supported. The basic relation contrasts an attribute name with a value through a relational operator. For example:

```
is_hierarchical == true and area <= 6
```

The relational operators are

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

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only **true** or **false**.

Existence relations determine if an attribute is defined or not defined for the object. For example:

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are

```
defined
undefined
```

These operators apply to any attribute as long as it is valid for the object class.

For a complete description of conditional expressions, see the *PrimeTime Reference Manual*.

This command matches a regular expression matching in the same way as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the filter *expression*. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding "." to the beginning or end of the expressions as needed. You can make the regular expression search case-insensitive by using the **-nocase** option.

EXAMPLES

The following example creates a collection of only hierarchical cells.

```
pt_shell> set a [filter_collection [get_cells *] \  
"is_hierarchical == true"]  
{"Adder1", "Adder2"}
```

The following example creates a collection of all nonmoded cell timing_arc objects in the current design.

```
pt_shell> set b [filter_collection \  
[get_timing_arcs -of_objects [get_cells *]] \  
"undefined(mode)"]
```

SEE ALSO

collections (2), **regexp** (2).

find

The **find** command, used to create a collection of design objects, is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
string find  
[-hierarchy]  
[-flat]  
type  
[object_list]
```

Data Types

type	string
object_list	list

ARGUMENTS

-hierarchy
Find objects throughout hierarchy.

-flat
Not supported by PrimeTime.

type
Object class (Values: design, port, net, cell, pin, clock, library, lib_cell, lib_pin).

object_list
Patterns to match.

DESCRIPTION

The **find** command creates a collection of objects. The command exists in PrimeTime for compatibility with Design Compiler. Complete information about the **find** command can be found in the Design Compiler documentation. The supported method for creating collections of objects is to use **get*** commands (for example, the **get_cells** command or the **get_pins** command).

SEE ALSO

```
get_cells(2)  
get_clocks(2)  
get_designs(2)  
get_lib_cells(2)  
get_lib_pins(2)  
get_libs(2)  
get_nets(2)  
get_pins(2)
```



```
get_ports(2)
```

fix_eco_timing

Improves or fixes timing violations through engineering change order (ECO) changes.

SYNTAX

```
string fix_eco_timing  
[-type fixing_type]  
[-slack_lesser_than slack_limit]  
[-slack_greater_than slack_limit]  
[-group group]  
[-pba_mode effort]  
[-from from_list]  
[-to to_list]  
[-setup_margin margin]  
[-hold_margin margin]  
[-buffer_list buffer_list]  
[-verbose]
```

Data Types

<i>fixing_type</i>	string
<i>slack_limit</i>	float
<i>group</i>	string
<i>effort</i>	string
<i>from_list</i>	list
<i>to_list</i>	list
<i>margin</i>	float
<i>buffer_list</i>	list

ARGUMENTS

- type *fixing_type***
Required option that specifies the desired fixing type. Valid values are *setup* and *hold*.
- slack_lesser_than *slack_limit***
Specifies that only those paths with a slack less than the *slack_limit* option are to be fixed. You can combine this option with the *-slack_greater_than* option to fix paths inside inside or outside a given slack range. If you do not specify this option, the default *-slack_lesser_than* option limit is zero.
- slack_greater_than *slack_limit***
Specifies that only those paths with a slack greater than the *slack_limit* option are to be fixed. This can be useful to avoid spending time on timing paths that are grossly violating design problems or incorrect constraints. If you do not specify this option, there is no *-slack_greater_than* option limit.
- group *group***
Restricts the fixing process to paths in this *group_name* type. Paths are grouped by using the **group_path** or **create_clock** command. If desired, you can supply multiple path groups to this option using a list of group names

(wildcards are allowed) or a collection of *path_group* objects.

-pba_mode effort

Controls the use of path-based analysis. The effort value can be either *none*, *path*, or *exhaustive*, and the meanings are the same as in the **report_timing** and **get_timing_path** commands. That is, when *none* is specified (the default), path-based analysis is not applied. When you set the *effort* value to the *path* value, path-based analysis is applied to paths after they are gathered. When you set the *effort* to The *exhaustive* value is an exhaustive path-based analysis path search algorithm, applied to determine the worst path-based analysis path set in the design. When specified, the fixing process uses the specified path-based analysis type to determine the timing fixes needed. This option pertains only to hold fixing; setup fixing always uses the *effort* of *none*.

-from from_list

Specifies a list of from pins, ports, nets, or clocks to be used for timing path fixing. This option is consistent with the **report_timing** and **get_timing_paths** commands. That is, if you do not specify the *from_list* option, the default behavior reports the longest path to an output port if the design has no timing constraints. Otherwise, the default behavior is to report the path with the worst slack within each path group if the design has timing constraints.

-to to_list

Specifies a list of to pins, ports, nets, or clocks to be used for timing path fixing. This option is consistent with the **report_timing** and **get_timing_paths** commands. That is, if you do not specify the *to_list* option, the default behavior reports the longest path to an output port if the design has no timing constraints. Otherwise, the default behavior is to report the path with the worst slack within each path group if the design has timing constraints.

-setup_margin

Specifies an additional fixing margin to be applied to setup slacks during hold fixing. By default, the setup margin is zero.

-hold_margin

Specifies an additional fixing margin to be applied to hold slacks during hold fixing. By default, the setup margin is zero.

-buffer_list

Specifies the list of allowable buffers to use for hold-fixing. Buffer names can either be in a library or ref_name form, or supplied as a ref_name. This option is required when performing hold fixing.

-verbose

Shows additional information during the fixing process.

DESCRIPTION

This command seeks to make ECO changes to the design that improves or resolves setup/hold setup. You can use the **fix_eco_timing** command in the single-core analysis, distributed multi-scenario analysis, and multicore analysis flows within

PrimeTime. It uses algorithms that attempt to improve the specified slack type, while minimizing the number of ECO changes as well as the area impact of the changes.

You must specify a *setup* or *hold* fixing type with the required *-type* option. The fixing strategies are different for setup and hold fixing, due to the different nature of the violations. Setup fixing relies on cell upsizing to reduce logic delay and improve setup slack. Hold fixing relies on buffer insertion to add delay and improve hold slack. During setup fixing, if a significant amount of cross-coupling is detected on the slack-critical nets, the procedure first attempts to perform upsizes to reduce the setup delta delays, followed by the normal upsizing process.

The fixing process honors any *dont_touch* attributes that are applied to the design. Just as with Design Compiler and IC Compiler, a *dont_touch* attribute can be applied to an upper-level hierarchical cell, and its implicit effect propagates downward into the hierarchy. If a different *true* or *false* *dont_touch* value is applied deeper in the hierarchy, it takes precedence over any higher-level *dont_touch* attributes. The *dont_touch* attributes can also be placed on specific cell and net objects. In setup-fixing, resizing of a cell is prevented if that cell is *dont_touched*, either directly or due to upper-level hierarchy. In hold-fixing, buffer insertion at a pin is prevented if the the net segment directly connected to the pin is *dont_touched*, either directly or due to upper-level hierarchy.

Neither fixing type makes changes to clock nets used as data where timing violations exist. Setup fixing seeks to avoid introducing design rule checking (DRC) violations, but it is permitted to introduce hold violations since setup is a harder problem to fix. Hold fixing seeks to avoid introducing both setup and DRC violations. If you want to fix both types, it is recommended that you first fix setup timing and then fix hold timing.

In distributed multi-scenario analysis fixing, a restriction in one scenario also restricts design changes in other scenarios. A *dont_touch* attribute in any scenario prevents design changes across all scenarios for the affected cells and nets. Likewise, a clock propagating through a cell or pin in any scenario prevents design changes across all scenarios for the affected cells or nets.

The *-group*, *-from*, and *-to* options allow targeted fixing of the design. The *-group* option allows fixing to be limited to one or more specified path groups. You can use lists and wildcards to specify the list of allowable path groups. The *-from* and *-to* options allow fixing only for paths linked to a certain set of startpoints or endpoints, respectively. You can specify both the *-from* and *-to* options to fix timing only along a certain path.

The **fix_eco_timing** command requires that pin slack information be available in the design. If you have not set the **timing_save_pin_arrival_and_slack** variable to *true* (the default is *false*), the **fix_eco_timing** command automatically sets it to *true* and updates the design with arrival and slack information.

Slack threshold controls are available to provide additional control over the fixing process. The *-slack_lesser_than* option indicates that only violations less than this value should be fixed. In addition, you can use the *-slack_greater_than* option to avoid spending time fixing gross timing violations that might be caused by incorrect constraints. The slack thresholds only pertain to the slack of the specified fixing type (setup or hold).

Setup Fixing

When cells are upsized during setup-fixing, there is no default area limit imposed on the alternative cells considered for the upsize. However, if there is a specific requirement on the maximum size increase of a resized cell, you can control the area increase by setting the `eco_alternative_area_ratio_threshold` variable that specifies the maximum upsize area increase as a ratio. By setting this variable to 2, upsize is limited to twice the area of the original cell. A value of zero (the default value) means that no limit is imposed on upsizing. Imposing an upsize area limit can increase the timing predictability after physical implementation of the ECO changes, as the smaller cell size increases are less likely to perturb layout during incremental placement and routing. However, more changes (and more runtime) might be required to achieve the needed slack improvement.

In setup fixing, using certain library cells can be restricted during the resizing process by applying the `pt_dont_use` user attribute on the restricted `lib_cell` objects. For an example of how to apply this attribute, see the `EXAMPLES` section.

Hold Fixing

For hold fixing, it is recommended that a good representative set of buffer and delay cells be provided that covers the desired delay range. Supplying too many buffers in the list might not improve results and can also lead to longer runtime. Hold-fixing shows a list of estimated buffer delays before starting the fixing process. In the distributed multi-scenario analysis flow, the estimated buffer delays are computed across all scenarios. To see the buffer delays without performing any fixing, use a large negative `-slack_lesser_than` value to avoid fixing. This is useful for choosing a buffer list that has the minimal set of cells needed to provide good coverage of the desired delay range.

In addition, fixing margins are also available to provide additional control over the hold-fixing process. A positive `-hold_margin` value indicates that a hold violation should be over-fixed by the specified amount. This extra margin can be useful to account for timing differences between the predicted timing and the actual layout timing in PrimeTime due to scenic routes, legalization effects, and so on. In hold fixing, you can use the `-setup_margin` option to control how setup slack is preserved. By setting a positive `-setup_margin` value during hold fixing, the hold fixing process attempts to preserve that much positive setup slack while fixing hold violations. By specifying a negative `-setup_margin` value, hold fixing limits its setup slack impact to that negative value. Note that the `-hold_margin` value does not affect which hold violations are fixed. The `-slack_lesser_than` is used for this purpose. However, the `-hold_margin` value does provide a way to specify how much "over-fixing" should be performed on those violations.

In the `-verbose` hold fixing mode, a quantity called TFS is shown for the pins selected for buffer insertion. TFS represents the total fixable slack quantity at that pin. TFS is similar to TNS (total negative slack) except that it also takes into account any setup criticality, which can limit the amount of delay that can be inserted at that pin. The `-verbose` hold fixing also shows more information about where the buffers are inserted and where resizes are being performed. It also shows a summary of each iteration at the end.

EXAMPLES

The following example shows how to fix setup violations less than zero slack, but avoiding some grossly violating setup paths with slack less than -2:

```
pt_shell> fix_eco_timing -type setup -slack_greater_than -2
```

The following example shows how to fix a small set of hold violations that have an exhaustive path-based slack less than zero:

```
pt_shell> fix_eco_timing -type hold -pba_mode exhaustive \
          -buffer_list {BUFX2 DLY1X2 DLY2X2}
```

The following example shows how to fix setup violations only in a selected set of path groups, with verbose output enabled:

```
pt_shell> fix_eco_timing -type setup -group {SYSCLK* IOCLK} -verbose
```

The following example shows how to apply the *pt_dont_use* user attributes to a set of library cells, and you must define the user attribute first:

```
pt_shell> define_user_attribute pt_dont_use \
          -quiet -type boolean -class lib_cell
```

You can use the attribute directly with the **set_user_attribute** command, or, for convenience, you can define the following procedure:

```
proc set_pt_dont_use {lib_cell} {
    set_user_attribute \
        -class lib_cell \
        [get_lib_cell -quiet $lib_cell] \
        pt_dont_use true
}
```

Now, you can use this procedure to apply the attribute:

```
set_pt_dont_use {lib/CLKBUF* lib/CLKMUX*}
```

In the distributed multi-scenario analysis flow, you must define and apply the attribute at the scenarios using the **remote_execute** command.

SEE ALSO

```
create_clock(2)
get_timing_paths(2)
group_path(2)
remote_execute(2)
report_timing(2)
set_user_attribute(2)
eco_alternative_area_ratio_threshold(3)
timing_save_pin_arrival_and_slack(3)
```

foreach_in_collection

Iterates over the elements of a collection.

SYNTAX

```
string foreach_in_collection  
itr_var  
collections  
body
```

Data Types

<i>itr_var</i>	string
<i>collections</i>	list
<i>body</i>	string

ARGUMENTS

<i>itr_var</i>	Specifies the name of the iterator variable.
<i>collections</i>	Specifies a list of collections over which to iterate.
<i>body</i>	Specifies a script to execute per iteration.

DESCRIPTION

The **foreach_in_collection** command is used to iterate over each element in a collection. You cannot use the Tcl-supplied **foreach** command to iterate over collections because the **foreach** command requires a list, and a collection is not a list. Also, using the **foreach** command on a collection causes the collection to be deleted.

The arguments to **foreach_in_collection** parallel those of **foreach**: an iterator variable, the collections over which to iterate, and the script to apply at each iteration. All arguments are required. Note that the **foreach_in_collection** command does not allow a list of iterator variables.

During each iteration, the *itr_var* option is set to a collection of exactly one object. Any command that accepts the *collections* option as an argument accepts the *itr_var* option because they are of the same data type, which is collection.

You can nest the **foreach_in_collection** command within other control structures, including another **foreach_in_collection** command.

Note that if the body of the iteration is modifying the netlist, it is possible that all or part of the collection involved in the iteration are deleted. The **foreach_in_collection** command is safe for such operations. If a command in the body causes the collection to be removed, at the next iteration, it ends with a message

indicating that the iteration ended prematurely.

An alternative to collection iteration is to use complex filtering to create a collection that includes only the desired elements, then apply one or more commands to that collection. If the order of operations does not matter, the following are equivalent. The first is an example without iterators.

```
set s [get_cells {U1/*}]
command1 $s
command2 $s
unset s
```

The following is the same example using the **foreach_in_collection** command:

```
foreach_in_collection itr [get_cells {U1/*}] {
    command1 $itr
    command2 $itr
}
```

For collections with large numbers of objects, the non-iterator version is more efficient, though both produce the same results if the commands are order-independent.

EXAMPLES

The following example removes the wire load model from all hierarchical cells in the current instance.

```
pt_shell> foreach_in_collection itr [get_cells *] {
?           if {[get_attribute $itr is_hierarchical] == "true"} {
?           remove_wire_load_model $itr
?           }
?           }
Removing wire load model from cell 'i1'.
Removing wire load model from cell 'i2'.
```

SEE ALSO

```
collections(2)
filter_collection(2)
query_objects(2)
```

get_alternative_lib_cells

Creates a collection of library cells that can be used to replace or "size" a specified cell in the current design.

SYNTAX

```
string get_alternative_lib_cells
```

```
[-current_library]  
[-libraries lib_spec]  
[-base_names]  
object
```

Data Types

<i>object</i>	string
<i>lib_spec</i>	list

ARGUMENTS

object

Specifies a cell or a library cell for which alternative library cells is required.

-current_library

If this option is specified, then PrimeTime searches for library cells in the cell's current library or library cell's current library only. You cannot specify this option with the **-libraries** option.

-libraries *lib_spec*

Specifies a list of libraries from which to select alternative library cells. The default is to select from all libraries in the link path. The *lib_spec* option can be a list of library names, or collections of libraries loaded into PrimeTime; the latter can be obtained using the **get_libs** command. You cannot specify this option with the **-current_library** option.

-base_names

Specifies that a list of library cell base names is to be returned. Multiple library cells may have the same base name, however the list is stripped of these duplicate names. This option is enabled by default if you invoke PrimeTime with the **-multi_scenario** option.

DESCRIPTION

The **get_alternative_lib_cells** command creates a collection of library cells that can be used to replace or "size" a specified cell or library cell. A library cell is included in the collection only if it satisfies the same criteria used by the **size_cell** command to determine whether a library cell can be used to replace a cell in the current design.

If the `-current_library` option is specified, then PrimeTime searches for library cells in the cell's current libraries or library cell's current library only. If the `-libraries` option is specified, then PrimeTime searches for library cells in the libraries contained within the `lib_spec` option only.

Alternatively, if neither of the above options are specified, PrimeTime searches for the library cell in the following sources in this order:

- 1) The object's current library
- 2) The `link_path_per_instance` variable
- 3) The `link_path` variable. If a library is found with a min library relationship (from `set_min_library`), the min library is automatically included.

To see a comparison of the slack and design cost within a set of alternative library cells, use the `report_alternative_lib_cells` command.

EQUIVALENT LIBRARY CELLS

Two library cells are compatible for sizing if all of the following restrictions are satisfied. The exact same set of restrictions applies to the `size_cell`, `get_alternative_lib_cells` and `report_alternative_lib_cells` commands.

- 1) The library cells must be functionally equivalent. At least one of the following must be true, and none can be false:

- If defined, the `function_id` attribute must match. This attribute is created by Library Compiler and is available on `lib_cell` objects.
 - If defined, the `mog_func_id` attribute must match. This attribute is created by Library Compiler for multiple output gate (MOG) cells and is available on `lib_cell` objects.
 - If defined, the `user_function_class` attribute must match. This is a user-defined functional equivalency attribute for cells that are too complex for Library Compiler to compute functional information for. This attribute is available on `lib_cell` objects.
- 2) The number of pins, ordering of pins, pin directions, and internal pin names must match.
 - 3) The number of enabled library cell arcs, arc types and sense must match, including any conditional or "when" expressions.

EXAMPLES

The following example shows how to find alternative library cells for a cell in the design. Here, `u_max` is an instance of `class_max/OR3` and `u_min` is an instance of `class_min/OR3`.

```
pt_shell> set link_path [list * class_max.db]
* class_max.db
pt_shell> set_min_library class_max.db -min_version class_min.db
Loading db file '/u/libs/class_max.db'
Loading db file '/u/libs/class_min.db'
Created max/min library relationship:
  Max: /u/libs/class_max.db:class_max
  Min: /u/libs/class_min.db:class_min
...
pt_shell> get_alternative_lib_cells -libraries class_min u_max
{"class_min/OR3", "class_min/OR3P"}
pt_shell> get_alternative_lib_cells -libraries class_max u_min
{"class_max/OR3", "class_max/OR3P"}
pt_shell> get_alternative_lib_cells u_max
{"class_max/OR3P", "class_min/OR3", "class_min/OR3P"}
pt_shell> get_alternative_lib_cells u_min
{"class_max/OR3", "class_max/OR3P", "class_min/OR3P"}
pt_shell> get_alternative_lib_cells u_min -current_library
{"class_min/OR3P"}
pt_shell> get_alternative_lib_cells -base_names u_min
{"OR3", "OR3P"}
```

The following example shows how to find alternative library cells for a `lib_cell`.

```
pt_shell> get_alternative_lib_cells class_max/OR3
{"class_max/OR3P", "class_min/OR3", "class_min/OR3P"}
pt_shell> get_alternative_lib_cells class_max/OR3 -current_library
{"class_max/OR3P"}
```

SEE ALSO

```
get_libs(2)
report_alternative_lib_cells(2)
set_min_library(2)
size_cell(2)
link_path_per_instance(3)
link_path(3)
```

get_attribute

Retrieves the value of an attribute on an object.

SYNTAX

```
string get_attribute [-class class_name] [-quiet] [-value_list] object_spec  
attr_name  
string class_name  
string object_spec or  
collection object_spec  
string attr_name
```

ARGUMENTS

-class *class_name*
Specifies the class name of *object_spec*, if *object_spec* is a name. Valid values for *object_spec* are *design*, *port*, *cell*, *pin*, *net*, *lib*, *lib_cell*, *lib_pin*, *clock*, *timing_path*, and *timing_point*. You must use this option if *object_spec* is a name.

-quiet
Indicates that any error and warning messages are not to be reported.

-value_list
Normally the return value of `get_attribute` command is a string if there is only a single object, and a list if multiple objects is specified. This option indicates that return value should be a list, even if there is only a single object specified to retrieve attribute from.

object_spec
Specifies a single object from which to get the attribute value. *object_spec* must be is either a collection of exactly one object, or a name which is combined with the *class_name* to find the object. If *object_spec* is a name, you must also use the **-class** option.

attr_name
Specifies the name of the attribute whose value is to be retrieved.

DESCRIPTION

Retrieves the value of an attribute on an object. The object is either a collection of exactly one object, or a name of an object. If it is a name, the **-class** option is required. The return value is a string.

EXAMPLES

In the following example, the first command defines an attribute 'X' for cells. The second command sets the attribute to a value on all cells in this level of the hierarchy. The third command retrieves the value from one cell, combines it with the application attribute **full_name**, and creates a simple report.

```

pt_shell> define_user_attribute -type int -class cell X
pt_shell> set_user_attribute [get_cells *] X 30
Set attribute 'X' on 'i1'
Set attribute 'X' on 'i2'
pt_shell> foreach_in_collection sel [get_cells *] {
?          echo -n "On '[get_attribute $sel full_name]', "
?          echo "X = [get_attribute $sel X]"
?          }
On 'i1', X = 30
On 'i2', X = 30

```

SEE ALSO

collections (2), **define_user_attribute** (2), **foreach_in_collection** (2), **list_attributes** (2), **remove_user_attribute** (2), **report_attribute** (2), **set_user_attribute** (2).

get_cells

Creates a collection of cells from the current design relative to the current instance. You can assign these cells to a variable or pass them into another command.

SYNTAX

```
collection get_cells
[-hierarchical]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-filter expression]
[patterns | -of_objects objects]
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-hierarchical

Searches for cells level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a cell block1/adder, a hierarchical search finds it using "adder".

-filter *expression*

Filters the collection with the *expression* value. For any cells that match *patterns* or *objects*, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.

-quiet

Suppresses warning and error messages if there are no objects that match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The *-regexp* and *-exact* options are mutually exclusive.

-nocase

When combined with the *-regexp* option, makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regexp* option.

-exact

Disables simple pattern matching. For use when searching for objects that

contain the * and ? wildcard characters. The `-exact` and `-regexp` options are mutually exclusive; you can specify only one.

`-of_objects objects`

Creates a collection of cells connected to the specified objects. In this case, each object is either a named pin, a pin collection, a named net or a net collection. The `-of_objects` and `patterns` options are mutually exclusive; you can specify only one. In addition, you cannot use the `-hierarchical` option with the `-of_objects` option.

`patterns`

Matches cell names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the `-regexp` option. Patterns can also include collections of type cell. The `patterns` and `-of_objects` options are mutually exclusive; you can specify only one.

DESCRIPTION

This command creates a collection of cells in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any cells match the `patterns` or `objects` option and passes the filter (if specified). If no objects match the criteria, an empty string is returned.

If any `patterns` or `objects` option fails to match any objects and the current design is not linked, the design automatically links.

Regular expression matching is the same as in the Tcl **regexp** command. When using the `-regexp` option, take care in the way you quote the `patterns` and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding "." to the beginning or end of the expressions as needed.

You can use the **get_cells** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_cells** result to a variable.

When issued from the command prompt, the **get_cells** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_cells** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_cells** command as an argument to the **query_objects** command.

For information about collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the cells that begin with "o" and reference an FD2 library cell. Although the output looks like a list, it is not. The output is just a display.

```
pt_shell> get_cells "o*" -filter "ref_name == FD2"  
{"o_reg1", "o_reg2", "o_reg3", "o_reg4"}
```

The following example shows that, given a collection of pins, you can query the cells connected to those pins.

```
pt_shell> set pinsel [get_pins o*/CP]  
{"o_reg1/CP", "o_reg2/CP"}  
  
pt_shell> query_objects [get_cells -of_objects $pinsel]  
{"o_reg1", "o_reg2"}
```

The following example removes the wire load model from cells i1 and i2.

```
pt_shell> remove_wire_load_model [get_cells {i1 i2}]  
Removing wire load model from cell 'i1'.  
Removing wire load model from cell 'i2'.  
1
```

SEE ALSO

```
collections(2)  
filter_collection(2)  
get_pins(2)  
link_design(2)  
query_objects(2)  
regex(2)  
collection_result_display_limit(3)
```

get_clock_network_objects

Returns a collection of objects that belong or relate to the direct clock network.

SYNTAX

```
collection get_clock_network_objects  
-type object_type  
[clock_list]  
[-include_clock_gating_network]
```

Data Types

<i>object_type</i>	string
<i>clock_list</i>	list

ARGUMENTS

`-type object_type`
Specifies the type of the clock network objects to be returned.

`clock_list`
Specifies the clock domains of which the clock network objects are returned.
By default, the command returns all clock network objects.

`-include_clock_gating_network`
Indicates that clock gating networks are included in the clock network.

DESCRIPTION

This command returns a collection of clock network objects of certain type (specified by the *object_type* option) that belong or relate to one or several clock domains (specified by the *clock_list* option), or all clock domains if the *clock_list* option is not specified.

A clock network is a special logic part of the design that propagates the clocks from the clock sources to the clock pins of latches, flip-flops (that function as anything but propagating clocks) or black-box IPs. The propagation also stops at design output ports, dangling pins or nets, or the sources of other clocks. The **get_clock_network_objects** command retrieves certain types of objects from the direct clock network (including the latches, flip-flops and black box IPs driven by the clock network). If the specified clock is a generated clock, the propagation does not go backward to the clock network of its master clock. If the specified clock is a master clock, the propagation does not go forward to the clock network of its generated clocks either.

The *object_type* option can be one of the following:

`cell`
Cells that belong to the clock network, including non-inverting or inverting buffers, combinational cells, library defined clock gating cells, etc.

register

Latches, flip-flops or black box IPs, such as memory cells, that are driven by the clock network.

net

Nets that connect the clock network cells to other clock network cells, to the driven registers, or to the I/O ports of design.

pin

Pins and ports that are connected with the clock network nets. Note that the clock pins of the driven registers are included.

clock_gating_output

Output pins of clock gating cells, either defined by the library, inserted by Power Compiler, automatically inferred by PrimeTime, or manually set by the user. The results are consistent with those of **report_clock_gating_checks**, and can be affected by other clock gating check related commands or variables.

The `-include_clock_gating_network` option indicates that discrete logic structure functioning as clock gating is taken as belonging to the clock network. Only the typical clock gating logic is considered as qualified clock gating network to be included in the clock network. The typical clock gating logic starts from the output of a level sensitive latch driven by the specified clock, possibly goes through a couple of buffers, and comes back to the specified clock network through one of the input pins of an AND or OR gate. The input pin must be a PrimeTime clock check enable pin, either inferred or manually set. Therefore, similarly, the results can be affected by clock gating check related commands or variables. When the clock gating network is included in the clock network, all objects in the clock gating network is considered as clock network objects. The latch is regarded as a clock network *cell*, but not a clock network *register*. The *clock_gating_output* object type is not affected by this option.

EXAMPLES

The following command returns a collection of clock network pins of all clock domains in the design, not including pins of the clock gating networks.

```
pt_shell> get_clock_network_objects -type pin
```

SEE ALSO

```
create_clock(2)  
create_generated_clock(2)  
get_clocks(2)  
get_generated_clocks(2)  
remove_clock(2)  
remove_generated_clock(2)  
report_clock(2)  
remove_clock_gating_check(2)  
remove_disable_clock_gating_check(2)
```

```
report_clock_gating_check(2)
set_clock_gating_check(2)
set_disable_clock_gating_check(2)
timing_disable_clock_gating_checks(2)
```

get_clocks

Creates a collection of clocks from the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection get_clocks
[-quiet]
[-regex]
[-nocase]
[-filter expression]
patterns
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regex

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with the -regex option, makes matches case-insensitive. You can use the -nocase option only when you also use the -regex option.

-filter *expression*

Filters the collection with the *expression* option. For any clocks that match the *patterns* argument, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result.

patterns

Matches clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_clocks** command creates a collection of clocks in the current design that match certain criteria. The command returns a collection if any clocks match the *patterns* argument and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_clocks** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_clocks** command result to a variable.

When issued from the command prompt, the **get_clocks** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_clocks** command provides a fast, simple way to display clocks in a collection. However, if you want the flexibility provided by the *query_objects* options (for example, if you want to display the object class), use the **get_clocks** command as an argument to the **query_objects** option.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man page for the **all_clocks** command, which also creates a collection of clocks.

EXAMPLES

The following example applies the **set_max_time_borrow** command to all clocks in the design matching "PHI*".

```
pt_shell> set_max_time_borrow 0 [get_clocks "PHI*"]
```

The following example sets the variable `clock_list` to a collection of clocks that have period of 15.

```
pt_shell> set clock_list [get_clocks * -filter "period==15"]
```

SEE ALSO

```
all_clocks(2)
collections(2)
create_clock(2)
query_objects(2)
report_clock(2)
collection_result_display_limit(3)
```

get_correlations

Creates a collection of correlations. You can assign these correlations to a variable or pass them into another command.

SYNTAX

```
collection get_correlations  
[-quiet]  
[-regex]  
[-nocase]  
[-filter expression]  
patterns
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-quiet
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regex
Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase
When combined with the *-regex* option, makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regex* option.

-filter *expression*
Filters the collection with the *expression* argument. For any correlations that match the *patterns* argument, the expression is evaluated based on the correlation's attributes. If the expression evaluates to true, the correlation is included in the result.

patterns
Matches correlation names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_correlations** command creates a collection of correlations in the current design that match certain criteria. The command returns a collection if any correlations match the *patterns* argument and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_correlations** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_correlations** command result to a variable.

When issued from the command prompt, the **get_correlations** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_correlations** command provides a fast, simple way to display correlations in a collection. However, if you want the flexibility provided by the *query_objects* options (for example, if you want to display the object class), use the **get_correlations** command as an argument to the *query_objects* option.

For information about collections and the querying of objects, see the **collections** command man page. In addition, see the man page for the **all_correlations** command, which also creates a collection of correlations.

EXAMPLES

The following example sets the **corr_list** variable to a collection of correlations that have a constant of 1.

```
pt_shell> set corr_list [get_correlations * -filter "constant==1"]
```

SEE ALSO

```
all_correlations(2)
collections(2)
create_correlation(2)
query_objects(2)
collection_result_display_limit(3)
```


get_current_power_domain

Gets the power domains that are included in the power analysis.

This command works only in UPF mode.

SYNTAX

string **get_current_power_domain**

DESCRIPTION

The *set_current_power_domain* command provides the control of calculating power consumption for the domains of interest. Domain-based power consumption data can be generated for a multi power domain design. Only the power dissipated in the blocks covered by the current power domain list will be calculated and reported. The *get_current_power_domain* command returns the power domains being included in the power analysis.

EXAMPLES for UPF mode

The following example shows generating domain-based power analysis results. The design has two sub-blocks "InstA" and "InstB", which are covered by power domain "PD1" and "PD2" respectively. The first *report_power* command generates the power analysis results for the whole design, which is the default behavior. The second *report_power* command generates the power analysis results for the power consumed in power domain "PD1". The third *report_power* command generates the results for the power consumed in power domain "PD2".

```
pt_shell> ...
pt_shell> create_power_domain PD1 -elements {InstA}
pt_shell> create_power_domain PD2 -elements {InstB}
pt_shell> ...
pt_shell> report_power
pt_shell> set_current_power_domain PD1
pt_shell> report_power
pt_shell> get_current_power_domain
pt_shell> set_current_power_domain PD2
pt_shell> report_power
pt_shell> get_current_power_domain
```

SEE ALSO

set_current_power_domain(2).

get_current_power_net

Gets the power nets that are included in the power analysis.

This command works in UPF mode only.

SYNTAX

string **get_current_power_net**

DESCRIPTION

The *set_current_power_net* command provides control of calculating power consumption for the power nets of interest. Power net-based power consumption data can be generated for a multi-supply design. Only the power dissipated on the cell or part of cell that is supplied by the current power netlist is calculated and reported. The **get_current_power_net** command returns the power nets being included in the power analysis.

EXAMPLES for UPF mode

The following example shows generating power net-based power analysis results. The design has two subblocks "InstA" and "InstB", which is covered by power domain PD1 and PD2, respectively. There are total 6 supply nets defined. Among them, four are power nets and two are ground nets. The primary power net for power domain "PD1" is "VDDIS", and the primary power net for power domain "PD2" is "VDDGS". The first **report_power** command generates the power analysis results for the whole design, which is the default behavior. The second **report_power** command generates the power consumption associated with power net "VDDI". The third **report_power** command generates the power consumption associated with power net "VDDIS", which is the output of power switch "top_header". The fourth and fifth reports are for "VDDG" and "VDDGS", respectively.

```
pt_shell> ...
pt_shell> create_power_domain PD1 -elements {InstA}
pt_shell> create_power_domain PD2 -elements {InstB}
pt_shell> create_supply_net VDDI -domain PD1
pt_shell> create_supply_net VDDIS -domain PD1
pt_shell> create_supply_net VSS -domain PD1
pt_shell> set_domain_supply_net PD1 -primary_power_net VDDIS -
primary_ground_net VSS
pt_shell> connect_supply_net -ports InstA/LS_u1/VDDL VDDI
pt_shell> connect_supply_net -ports InstA/LS_u1/VDD VDDIS
pt_shell> connect_supply_net -ports InstA/LS_u1/VSS VSS
pt_shell> create_power_switch top_header \
    -domain PD1 \
    -output_supply_port {NSLEEP_OUT VDDIS} \
    -input_supply_port {NSLEEP_IN VDDI} \
    -on_state {state1 NSLEEP_IN {CNTL}} \
    -control_port { CNTL ctrl }
pt_shell> ...
pt_shell> create_supply_net VDDG -domain PD2
```

```

pt_shell> create_supply_net VDDGS -domain PD2
pt_shell> create_supply_net VSS -domain PD2 -reuse
pt_shell> set_domain_supply_net PD2 -primary_power_net VDDGS -
primary_ground_net VSS
pt_shell> create_power_switch gprs_header \
        -domain PD2 \
        -output_supply_port {NSLEEPOUT VDDGS} \
        -input_supply_port {NSLEEPIN VDDG} \
        -on_state {state1 NSLEEPIN {CNTL}} \
        -control_port { CNTL ctrl }

pt_shell> ...
pt_shell> report_power
pt_shell> set_current_power_net { VDDI }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDIS }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDG }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDGS }
pt_shell> report_power
pt_shell> get_current_power_net

```

SEE ALSO

set_current_power_net(2)

get_designs

Creates a collection of one or more designs loaded into PrimeTime. You can assign these designs to a variable or pass them into another command.

SYNTAX

```
collection get_designs
[-hierarchical]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-filter expression]
patterns
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

- hierarchical
Searches for designs inferred by the design hierarchy relative to the current instance. The full name of the object at a particular level must match the *patterns*. Using this option does not force an autolink.
- filter *expression*
Filters the collection with the *expression* value. For any designs that match *patterns*, the expression is evaluated based on the design's attributes. If the expression evaluates to true, the design is included in the result.
- quiet
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.
- regexp
Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The -regexp and -exact options are mutually exclusive.
- nocase
When combined with the -regexp option, makes matches case-insensitive. You can use the -nocase option only when you also use the -regexp option.
- exact
Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The -exact and -regexp options are mutually exclusive; you can specify only one.

patterns

Matches design names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the `-regexp` option. Patterns can also include collections of type design.

DESCRIPTION

This command creates a collection of designs from those currently loaded into PrimeTime that match certain criteria. The command returns a collection if any designs match the `patterns` option and pass the filter, if specified. If no objects matched your criteria, an empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the `-regexp` option, take care in the way you quote the `patterns` and filter `expression`; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding `".*"` to the beginning or end of the expressions as needed.

You can use the **get_designs** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_designs** result to a variable.

When issued from the command prompt, the **get_designs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable .

The "implicit query" property of the **get_designs** command provides a fast, simple way to display designs in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_designs** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the designs that begin with 'mpu.' Although the output looks like a list, it is just a display. A complete listing of designs is available using the **list_designs** command.

```
pt_shell> get_designs mpu*
{"mpu_0_0", "mpu_0_1", "mpu_1_0", "mpu_1_1"}
```

The following example shows that, given a collection of designs, you can remove those designs.

```
pt_shell> remove_design [get_designs mpu*]
Removing design mpu_0_0...
Removing design mpu_0_1...
```

```
Removing design mpu_1_0...
Removing design mpu_1_1...
```

SEE ALSO

```
collections(2)
filter_collection(2)
list_designs(2)
query_objects(2)
regexp(2)
remove_design(2)
collection_result_display_limit(3)
```

get_distributed_variables

Gets Tcl variables from slaves and creates an array at the master indexed by the scenario name.

SYNTAX

```
Boolean get_distributed_variables  
[-pre_commands pre_command_string]  
[-post_commands post_command_string]  
[-attributes attribute_list]  
[-merge_type type]  
[-null_merge_method method]  
variable_list
```

Data Types

<i>pre_command_string</i>	string
<i>post_command_string</i>	string
<i>attribute_list</i>	list
<i>type</i>	string
<i>method</i>	string
<i>variable_list</i>	list

ARGUMENTS

-pre_commands *pre_command_string*

A string of commands to be executed in the slave context before the retrieval of variables. Commands are grouped using the ";" character. The maximum size of a command is 1000 characters.

-post_commands *post_command_string*

A string of commands to be executed in the slave context after the retrieval of variables. Commands are grouped using the ";" character. The maximum size of a command is 1000 characters.

-attributes *attribute_list*

A list of attributes to be retrieved from a slave collection. If this option is not specified, only the **full_name**, **scenario_name**, and **object_class** attributes are retrieved. This option should be used in conjunction with the **set_distributed_parameter -collection_levels** commands to control the amount of data retrieved from the slave.

-merge_type *type*

This option can be used to merge variable values that come from the scenarios as they are retrieved. By default the **get_distributed_variables** command brings back a scenario variable as an array. With the **-merge_type** option values can be merged back into a variable during retrieval. In addition to simple variables, arrays of one or more dimensions can also be merged. The merging of collection objects is not supported. The allowed values for the **-merge_type** option are *min*, *max*, *average*, *total*, *list*, and *unique_list*.

-null_merge_method

This option allows the user to specify what behavior should be used when a null (empty) value of {} is encountered during the merging operations. With the default null merging method of ignore, the null is simply ignored. The value override allows a null value to win during a merging operation. The value error gives an error if a null value is found during a merging operation.

DESCRIPTION

The **get_distributed_variables** command retrieves data from the slaves and makes it available for further processing at the master. The variables included in the *variable_list* option must be slave variables. Each slave variable may have a different value depending on the scenario context. The command creates an array at the master for each variable. The array will have the same name as the variable. The array will be indexed by the scenario name. The data in the array for a given scenario index will have the same value as the slave variable in that scenario's context. The variable types currently supported are TCL Arrays, TCL lists and all types of collections. You can use the *-merge_type* and *-null_merge_method* options of the **get_distributed_variables** command to merge variable values that come from the scenarios as they are retrieved. This is especially useful when you are writing customized distributed multi-scenario analysis scripts. By default, the **get_distributed_variables** command brings back a scenario variable as an array. With the *-merge_type* option, the values can be merged back into a variable during retrieval. For more information on the merging process, refer to the *Distributed Multi-Scenario Analysis User Guide*. You need to be aware that when number of scenarios > number of processors variables may be destroyed during the save restore process. When number of scenarios < number of processors it is advisable to use the **pre_commands** variable to generate the variables to be transmitted. When retrieving a collection, use the *-attributes* option to specify what attributes are to be retrieved from the slaves for the given collection.

EXAMPLES

In the following examples, it is assumed that the number of scenarios < number of processors,

```
pt_shell> remote_execute {set my_paths [get_timing_paths -nworst 10]}
pt_shell> get_distributed_variables my_paths -attributes {slack}
pt_shell> foreach_in_collection path $my_paths(scen1) {
    echo [get_attribute $path slack]
}
pt_shell> remote_execute {set pslack [get_attribute get_pins U1/A] slack}
pt_shell> get_distributed_variables pslack
pt_shell> echo [array get pslack]
{scen1 0.1231 scen2 0.1232}
```

In the following example, shows access to an attribute which is itself a collection.

```
pt_shell> remote_execute {set my_pins [get_pins *]}
pt_shell> get_distributed_variable my_pins -attributes {direction clocks}
pt_shell> foreach_in_collection pin $my_pins(scen1) {
    echo "direction:[get_attribute $pin direction]"
}
```



```

        set my_clocks [get_attribute $pin clocks]
        foreach_in_collection my_clock $my_clocks {
            echo "full clock name : get_attribute $my_clock full_name" }
    }

```

In the following example, the variable slack is set at each scenario and merges the smallest value of the variable.

```

scenario bc:
set slack 0.3084

```

```

scenario tc:
set slack 0.0901

```

```

scenario wc:
set slack -0.7081

```

For example, to keep the numerically smallest value of the slack variable from all scenarios:

```

pt_shell> get_distributed_variables {slack} -merge_type min
1
pt_shell> echo $slack
-0.7081

```

In the following example, you get a unique list of all clocks at all scenarios

```

pt_shell> remote_execute {set clock_names
                        [get_object_name [all_clocks]]}
pt_shell> get_distributed_variables clock_names -merge_type list
1
pt_shell> echo $clock_names
CLK33 CLK66 CLK50 CLK100 ATPGCLOCK JTAGCLK

```

In the following examples, assume that the number of scenarios > number of processors.

```

pt_shell> get_distributed_variables my_paths -
pre_commands {set my_paths [get_timing_paths -nworst 10]} -attributes slack

```

SEE ALSO

```

set_distributed_variables(2)
set_distributed_parameters(2)

```

get_generated_clocks

Creates a collection of generated clocks.

SYNTAX

```
collection get_generated_clocks
[-quiet]
[-regexp]
[-nocase]
[-filter expression]
patterns
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with the -regexp option, makes matches case-insensitive. You can use the -nocase option only when you also use the -regexp option.

-filter *expression*

Filters the collection with the *expression* option. For any generated clocks that match the *patterns* option, the expression is evaluated based on the generated clock's attributes. If the expression evaluates to true, the generated clock is included in the result.

patterns

Matches generated clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_generated_clocks** command creates a collection of generated clocks from the current design that match certain criteria. The command returns a collection if any generated clocks match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

To create a generated clock in the design, use the **create_generated_clock** command.

To remove a generated clock from the design, use the **remove_generated_clock** command. To show information about clocks and generated clocks in the design, use the **report_clock** command.

You can use the **get_generated_clocks** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_generated_clocks** command result to a variable.

When issued from the command prompt, the **get_generated_clocks** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_generated_clocks** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the the **query_objects** command, use the **get_generated_clocks** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following command applies the **set_clock_latency** command on all generated clocks in the design matching the "GEN*".

```
pt_shell> set_clock_latency 2.0 [get_generated_clocks "GEN*"]
```

The following command removes all the generated clocks in the design matching "GEN1*".

```
pt_shell> remove_generated_clock [get_generated_clocks "GEN1*"]
```

SEE ALSO

```
collections(2)
create_generated_clock(2)
query_objects(2)
remove_generated_clock(2)
report_clock(2)
collection_result_display_limit(3)
```

get_ilm_objects

Returns a collection of nets, cells, or pins that are part of the interface logic for the current design.

SYNTAX

```
collection get_ilm_objects  
[-type {net | pin | cell}]
```

ARGUMENTS

```
-type {net | pin | cell}
```

Specifies the type of object to be returned as a collection of objects. Allowed values are *net*, *pin*, or *cell*. A collection is returned of objects of the specified type that belong to the interface logic on the current design.

DESCRIPTION

Returns a collection of nets, cells, or pins that have been identified as belonging to interface logic by using the **identify_interface_logic** command. The **is_interface_logic_pin** attribute identifies pins that are part of the interface logic. The command takes into account that the design corresponding to the interface logic model (ILM) is flattened, hierarchical nets are reduced to a single flattened net, and hierarchical pins and cells on the original design are not contained in the ILM. The objects returned are those that will be referred to in the interface logic netlist, script, and back-annotation files written using the **write_ilm_*** commands. Use the **get_ilm_objects** command to review the objects that have been identified as belonging to the interface logic on the current design.

For a discussion of ILM creation and the associated commands, see the **identify_interface_logic** man page.

EXAMPLES

The following example returns a collection of all interface logic cells.

```
pt_shell> get_ilm_objects -type cell
```

The following command returns a collection of all nets in the flattened ILM netlist.

```
pt_shell> get_ilm_objects -type net
```

SEE ALSO

```
identify_interface_logic(2)  
write_ilm_netlist(2)  
write_ilm_parasitics(2)  
write_ilm_script(2)  
write_ilm_sdf(2)
```

get_lib_cells

Creates a collection of library cells from libraries loaded into PrimeTime. You can assign these library cells to a variable or pass them into another command.

SYNTAX

```
collection get_lib_cells
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[-exact]
patterns | -of_objects objects
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* value. For any library cells that match the *patterns* or *objects* argument, the expression is evaluated based on the library cell's attributes. If the expression evaluates to true, the library cell is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** options are mutually exclusive.

-of_objects *objects*

Creates a collection of library cells that are referenced by the specified cells or own the specified library pins. In this case, each object is either a named library pin or netlist cell, or a library pin collection or a netlist cell collection. The **-of_objects** and *patterns* options are mutually exclusive;

you must specify one, but not both.

patterns

Matches library cell names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the *-regexp* option. Patterns can also include collections of type *lib_cell*. The *patterns* and *-of_objects* options are mutually exclusive; you must specify one, but not both.

DESCRIPTION

This command creates a collection of library cells from libraries currently loaded into PrimeTime that match certain criteria. The command returns a collection if any library cells match the *patterns* or *objects* option and pass the filter, if specified. If no objects match the criteria, an empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the *-regexp* option, take care in the way you quote the *patterns* and filter *expression*. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding "."*" to the beginning or end of the expressions as needed.

You can use the **get_lib_cells** command at the command prompt, or you can nest it as an argument to another command. For example, you can use it in the **query_objects** command. In addition, you can assign the **get_lib_cells** command result to a variable.

When issued from the command prompt, the **get_lib_cells** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_cells** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_lib_cells** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all library cells that are in the *misc_cmos* library and begin with AN2. Although the output looks like a list, it is just a display.

```
pt_shell> get_lib_cells misc_cmos/AN2*
{"misc_cmos/AN2", "misc_cmos/AN2P"}
```

The following example shows one way to find out the library cell used by a particular cell.

```
pt_shell> get_lib_cells -of_objects [get_cells o_reg1]  
{ "misc_cmos/FD2" }
```

SEE ALSO

```
collections(2)  
filter_collection(2)  
get_cells(2)  
query_objects(2)  
regexp(2)  
collection_result_display_limit(3)
```


get_lib_pins

Creates a collection of library cell pins from libraries loaded into PrimeTime. You can assign these library cell pins to a variable or pass them into another command.

SYNTAX

```
collection get_lib_pins
[-filter expression]
[-quiet]
[-regex]
[-nocase]
[-exact]
patterns | -of_objects objects
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* option. For any library cell pins that match the *patterns* option (or the *objects* option), the expression is evaluated based on the library cell pin's attributes. If the expression evaluates to true, the library pin is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regex

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The -regex option and the -exact option are mutually exclusive.

-nocase

When combined with the -regex option, makes matches case-insensitive. You can use the -nocase option only when you also use the -regex option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the "*" and "?" wildcard characters. The -exact and -regex options are mutually exclusive.

-of_objects *objects*

Creates a collection of library cell pins referenced by the specified netlist pins or owned by the specified library cells. In this case, each object is either a named library cell or netlist pin, or a library cell collection or

netlist pin collection. The *-of_objects* and *patterns* options are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell pin names against patterns. The *patterns* option can include the wildcard characters "*" and "?" or regular expressions, based on the *-regexp* option. The *patterns* option can also include collections of type **lib_pin**. The *patterns* and *-of_objects* options are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **get_lib_pins** command creates a collection of library cell pins from libraries currently loaded into PrimeTime that match certain criteria. The command returns a collection if any library cell pins match the *patterns* or *objects* options and pass the filter (if specified). If no objects matched the criteria, the empty string is returned.

Regular expression matching is the same as in the **regexp** Tcl command. When using the *-regexp* option, take care in the way you quote the *patterns* option and filter the *expression* option. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding "." to the beginning or end of the expressions as needed.

You can use the **get_lib_pins** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_lib_pins** command result to a variable.

When issued from the command prompt, the **get_lib_pins** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_pins** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_lib_pins** command as an argument to the **query_objects** command. For example, use this if you want to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all pins of the AN2 library cell in the misc_cmos library. Although the output looks like a list, it is just a display.

```
pt_shell> [get_lib_pins misc_cmos/AN2/*  
{"misc_cmos/AN2/A", "misc_cmos/AN2/B", "misc_cmos/AN2/Z"}
```

The following example shows one way to find out how the library pin is used by a

particular pin in the netlist.

```
pt_shell> get_lib_pins -of_objects o_reg1/Q  
{ "misc_cmos/FD2/Q" }
```

SEE ALSO

```
collections(2)  
filter_collection(2)  
get_libs(2)  
get_lib_cells(2)  
query_objects(2)  
regexp(2)  
collection_result_display_limit(3)
```

get_lib_timing_arcs

Creates a collection of library arcs for custom reporting and other processing. You can assign these library arcs to a variable and get the desired attribute for further processing.

SYNTAX

```
string get_lib_timing_arcs  
[-to to_list]  
[-from from_list]  
[-of_objects object_list]  
[-filter expression]  
[-quiet]
```

Data Types

<i>to_list</i>	list
<i>from_list</i>	list
<i>object_list</i>	list
<i>expression</i>	string

ARGUMENTS

- to *to_list***
Specifies the "to" library pins, or ports. All backward library arcs from the specified library pins or ports are considered.
- from *from_list***
Specifies the "from" library pins, or ports. All forward library arcs from the specified library pins or ports are considered.
- of_objects *object_list***
Specifies library cells or timing arcs. If a library cell is specified, all library cell arcs of that cell are considered. If a timing arc collection is given in the object list, the corresponding library timing arc is considered.
- filter *expression***
Specifies the filter expression. A filter expression is a string that comprises a series of logical expressions describing a set of constraints you want to place on the collection of library arcs. Each subexpression of a filter expression is a relation contrasting an attribute name with a value, by means of an operator.
- quiet**
Specifies that all messages are to be suppressed.

DESCRIPTION

Creates a collection of library arcs for custom reporting and other processing. You can assign these library arcs to a variable and get the desired attribute for further processing. Use the **foreach_in_collection** command to iterate among the

library arcs in the collection. You can use the **get_attribute** command to obtain information about the paths. You can also use the filter expression to filter the library arcs that satisfy the specified conditions. The following attributes are supported on library timing arcs:

```
from_lib_pin
is_disabled
is_user_disabled
mode
object_class
sdf_cond
sense
to_lib_pin
```

One attribute of a library timing arc is the **from_lib_pin**, which is the library pin from which the timing arc begins. In the same way, **to_lib_pin** is the library pin to which the timing arc ends. To get more information on **from_lib_pin** and **to_lib_pin**, use the **get_attributes** command. The **object_class** attribute holds the class name of library timing arcs: **lib_timing_arc**.

See the **collections** and **foreach_in_collection** man pages for more information.

EXAMPLES

The following procedure is an example of how the collection returned from an invocation of **get_lib_timing_arcs** can be utilized to provide useful and readable cell level arc information.

```
proc show_lib_arcs {args} {
    set lib_arcs [eval [concat get_lib_timing_arcs $args]]
    echo [format "%15s    %-15s  %18s" "from_lib_pin" "to_lib_pin" \
        "sense"]
    echo [format "%15s    %-15s  %18s" "-----" "-----" \
        "-----"]

    foreach_in_collection lib_arc $lib_arcs {
        set fpin [get_attribute $lib_arc from_lib_pin]
        set tpin [get_attribute $lib_arc to_lib_pin]
        set sense [get_attribute $lib_arc sense]
        set from_lib_pin_name [get_attribute $fpin base_name]
        set to_lib_pin_name [get_attribute $tpin base_name]
        echo [format "%15s -> %-15s  %18s" \
            $from_lib_pin_name $to_lib_pin_name \
            $sense]
    }
}
```

pt_shell> **show_lib_arc -of_objects [get_timing_arcs -of_objects ffa]**

from_lib_pin	to_lib_pin	sense
-----	-----	-----
CP -> D		setup_clk_rise
CP -> D		hold_clk_rise

CP -> Q	rising_edge
CP -> QN	rising_edge
CD -> Q	clear_low
CD -> QN	preset_low

```
pt_shell> show_lib_arc -of_objects mylib/AN2
```

from_lib_pin	to_lib_pin	sense
-----	-----	-----
A -> Z		positive_unate
B -> Z		positive_unate

```
pt_shell> show_lib_arc -from mylib/AN2/A
```

from_lib_pin	to_lib_pin	sense
-----	-----	-----
A -> Z		positive_unate

SEE ALSO

```
collections(2)
filter_collection(2)
foreach_in_collection(2)
get_attribute(2)
get_timing_arcs(2)
```

get_libs

Creates a collection of libraries loaded into PrimeTime. You can assign these libraries to a variable or pass them into another command.

SYNTAX

```
collection get_libs
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[patterns | -of_objects objects]
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

- filter *expression*
Filters the collection with the *expression* option. For any libraries that match *patterns* (or *objects*), the expression is evaluated based on the library's attributes. If the expression evaluates to true, the library is included in the result.
- quiet
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.
- regexp
Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modify the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The -regexp and -exact options are mutually exclusive.
- nocase
When combined with the -regexp option, makes matches case-insensitive. You can use the -nocase option only when you also use the -regexp option.
- exact
Disables simple pattern matching. This is used when searching for objects that contain the "*" and "?" wildcard characters. The -exact and -regexp options are mutually exclusive.
- of_objects *objects*
Creates a collection of libraries that contain the specified objects. In this case, each object is either a named library cell or a library cell collection. the -of_objects and patterns options are mutually exclusive; you can specify only one.

patterns

Matches library names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the `-regex` option. Patterns can also include collections of type lib. The `patterns` and `-of_objects` options are mutually exclusive; you can specify only one.

DESCRIPTION

The **get_libs** command creates a collection of libraries from those currently loaded into PrimeTime that match certain criteria. The command returns a collection if any libraries match the `patterns` option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the **regex** Tcl command. When using the `-regex` option, take care in the way you quote the `patterns` option and filter the `expression` option. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding "." to the beginning or end of the expressions as needed.

You can use the **get_libs** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_libs** command result to a variable.

When issued from the command prompt, the **get_libs** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_libs** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_libs** command as an argument to the **query_objects** command. For example, use this if you want to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries all loaded libraries. Although the output looks like a list, it is just a display. Note that a complete listing of libraries is available using the **list_libraries** command.

```
pt_shell> get_libs *
{"misc_cmos", "misc_cmos_io"}
```

The following example shows that given a library collection, you can remove those libraries. Note that you cannot remove libraries if they are referenced by a design.

```
pt_shell> remove_lib [get_libs misc*]
Removing library misc_cmos...
Removing library misc_cmos_io...
```


SEE ALSO

`collections(2)`
`filter_collection(2)`
`list_libraries(2)`
`query_objects(2)`
`regexp(2)`
`remove_lib(2)`
`collection_result_display_limit(3)`

get_license

Obtains a license for a feature.

SYNTAX

```
int get_license feature_list
```

Data Types

```
feature_list list
```

ARGUMENTS

feature_list

A list of features to be obtained. Refer to the *Synopsys System Installation and Configuration Guide* for a list of features supported by the current release, or determine from the key file all the features licensed at your site.

DESCRIPTION

Obtains a license for the named features. These features are checked out by the current user until the **remove_license** command is used or until the program is exited. The **get_license** command is valid only when Network Licensing is enabled.

The **list_licenses** command provides a list of the features that you currently have checked out.

For more information on the **get_license** command, refer to the *Synopsys System Installation and Configuration Guide*.

EXAMPLES

This example obtains the STAMP Compiler feature:

```
pt_shell> get_license STAMP-Compiler  
Checked out license 'STAMP-Compiler'  
1
```

SEE ALSO

```
license_users(2)  
list_licenses(2)  
remove_license(2)
```

get_nets

Creates a collection of nets from the netlist. You can assign these nets to a variable or pass them into another command.

SYNTAX

```
collection get_nets
[-hierarchical]
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-top_net_of_hierarchical_group]
[-segments]
[-boundary_type btype]
[patterns | -of_objects objects]
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-hierarchical

Searches for nets level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a net block1/muxsel, a hierarchical search would find it using "muxsel." You cannot use the *-hierarchical* option with the *-of_objects* option.

-filter *expression*

Filters the collection with *expression*. For any nets that match *patterns* (or *objects*), the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the net is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The *-regexp* and *-exact* options are mutually exclusive.

-nocase

When combined with the *-exact* option, makes matches case-insensitive. You can use *-nocase* only when you also use the *-exact* option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The -exact and -regexp options are mutually exclusive.

-top_net_of_hierarchical_group

Keep only the top net of a hierarchical group. When more than one hierarchical net of the same group is specified (local nets at various hierarchical levels of the same physical net), only the net closest to the top of the hierarchy is saved with the collection. In the case of multiple nets at the same level, the first net specified is kept. Although this option can be used when there are multiple nets specified, it is best used in combination with the -segments option and with a single net.

-segments

Return all global segments for given nets. This modifies the initial search which matches *patterns* or *objects*. It is applied after filtering, but before the -top_net_of_hierarchical_group option is determined. For each net, all global segments which are part of that net will be added to the result collection. Global net segments are all those physically connected across all hierarchical boundaries. Although this option can be used with other options and when there are multiple nets specified, it is best used with a single net. When combined with the -top_net_of_hierarchical_group option, you can isolate the highest net segment of a physical net.

-boundary_type btype

Specifies what to do when getting nets of boundary pins. This option requires the -of_objects option. Allowed values are lower, upper, and both, meaning the net inside the hierarchical block, outside the hierarchical block, or both nets, respectively. The option has no meaning for non-hierarchical pins. Note that The "upper" value is less useful, as getting pins of a hierarchical net will return the pin outside the hierarchical block, and a subsequent get_nets would find the upper hierarchical net. Using upper on a hierarchical pin is the same as omitting the option.

-of_objects *objects*

Creates a collection of nets connected to the specified objects. Each object is either a named pin, a pin collection, a named cell or cell collection. The -of_objects and *patterns* options are mutually exclusive; you can specify only one. In addition, you cannot use the -hierarchical option with the -of_objects option.

patterns

Matches net names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the -regexp option. Patterns can also include collections of type net. The *patterns* and -of_objects options are mutually exclusive; you can specify only one.

DESCRIPTION

The **get_nets** command creates a collection of nets in the current design, relative to the current instance that match certain criteria. The command returns a collection if any nets match the *patterns* or *objects* and pass the filter (if specified). If no objects matched the criteria, the empty collection is returned.

If any *patterns* (or *objects*) fail to match any objects and the current design is not linked, the design automatically links.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and *filter expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding **.*** to the beginning or end of the expressions as needed.

You can use the **get_nets** command at the command prompt, or you can nest it as an argument to another command (for example, **query_objects**). In addition, you can assign the **get_nets** result to a variable.

When issued from the command prompt, **get_nets** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of **get_nets** provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_nets** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

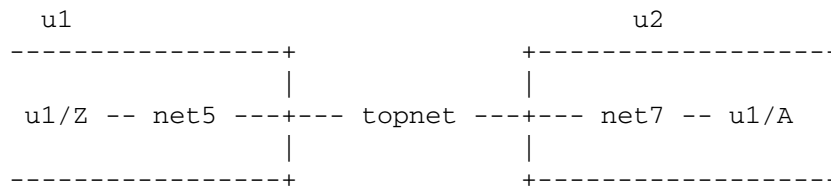
The following example queries the nets that begin with 'NET' in block 'block1'. Although the output looks like a list, it is just a display.

```
pt_shell> get_nets block1/NET
{"block1/NET1QNX", "block1/NET2QNX"}
```

The following example shows that with a collection of pins, you can query the nets connected to those pins.

```
pt_shell> current_instance block1
block1
pt_shell> set pinsel [get_pins {o_reg1/QN o_reg2/QN}]
{"o_reg1/QN", "o_reg2/QN"}
pt_shell> query_objects [get_nets -of_objects $pinsel]
{"NET1QNX", "NET2QNX"}
```

This example shows how to use the **-top_net_of_hierarchical_group** and **-boundary_type** options. Given the following circuit:



there are hierarchical cells *u1* and *u2* at this level, connected by a net *topnet*. Within *u1* is a pin *u1/Z* driving *net5*, and within *u2* is a pin *u1/A* being driven by *net7*. These three nets are physically the same net. Assume these are the only nets in the design. Notice the difference in the results of the following two **get_nets** commands:

```

pt_shell> get_nets * -hierarchical
{"topnet", "u1/net5", "u2/net7"}
pt_shell> get_nets * -hierarchical -top_net_of_hierarchical_group
{"topnet"}
  
```

Assume that at the top level, *topnet* is connected to pins *u2/in* and *u1/out*. Here are some examples of the *-boundary_type* option. Notice now in this case, the "upper" type does not add value.

```

pt_shell> get_nets -of_objects u2/in
{"topnet"}
pt_shell> get_nets -of_objects u2/in -boundary_type upper
{"topnet"}
pt_shell> get_nets -of_objects u2/in -boundary_type lower
{"u2/net7"}
pt_shell> get_nets -of_objects u2/in -boundary_type both
{"u2/net7", "topnet"}
pt_shell> get_nets -boundary lower -of_objects \
               [get_pins -of_objects [get_nets topnet]]
{"u1/net5", "u2/net7"}
  
```

SEE ALSO

```

collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
regexp(2)
collection_result_display_limit(3)
  
```

get_noise_violation_sources

Creates a collection of noise violation sources for custom reporting and other processing. You can assign these timing arcs to a variable and get the desired attribute for further processing.

SYNTAX

```
int get_noise_violation_sources  
[-above]  
[-below]  
[-low]  
[-high]  
[-nworst_endpoints pin_count]  
[-max_sources_per_endpoint pin_count]  
[-slack_type slack_type]  
[object_list]
```

Data Types

<i>pin_count</i>	integer
<i>slack_type</i>	string
<i>object_list</i>	list

ARGUMENTS

-above

Performs the reporting only above the rails. If this option is combined with the *-low* option, it reports for the noise bumps above the low rail. If it is combined with the *-high* option, it reports the noise bumps above the high rail. Otherwise, it reports the noise bumps above the high rail and above the low rail.

-below

Performs the reporting only below the rails. If this option is combined with the *-low* option, it reports for the noise bumps below the low rail. If it is combined with the *-high* option, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps below the high rail and below the low rail.

-low

Performs the reporting only for the low rail. If this option is combined with the *-above* option, it reports the noise bumps above the low rail. If it is combined with the *-below* option, it reports the noise bumps below the low rail. Otherwise, it reports the noise bumps for both below and above the low rail.

-high

Performs the reporting only for the high rail. If this option is combined with the *-above* option, it reports the noise bumps above the high rail. If it is combined with the *-below* option, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps for both below and above the high rail.

-nworst_endpoints *pin_count*
 Specifies the number of endpoint pins to be reported. Any number greater than 1 is accepted; the default value is 1.

-max_sources_per_endpoint *pin_count*
 Specifies the number of violation source pins per endpoint to be reported. Any number greater than 1 is accepted; the default value is 1.

-slack_type *slack_type*
 Specifies the type of slack to be used. Valid values are *area*, *height*, and *area_percent*. A *slack_type* of *area* reports slack as the voltage margin multiplied by the noise bump width. The voltage margin is defined by the noise bump height and noise immunity curves or DC noise margin. This setting is the default. A *slack_type* of *height* reports noise slack as the voltage margin. A *slack_type* of *area_percent* reports noise slack as the percentage of the noise constraint area. The noise constraint area is computed by multiplying the noise height constraint by the noise bump width. The default is *area*.

object_list
 Specifies the load pins for which the noise reporting is performed. If no pin is specified, reporting is performed on the entire design.

DESCRIPTION

Provides a collection of noise violation source pins for failing noise endpoints of the current design.

EXAMPLES

The following example creates a collection of the worst violation source for the worst endpoint in the current design and assign it to a variable, *viol_pins*.

```
pt_shell> set viol_pins [get_noise_violation_sources]
```

The following example creates a collection of maximum 10 violation sources for the endpoint *ffa/Q* in the above the low rail.

```
pt_shell> get_noise_violation_sources -above -low
-max_sources_per_endpoint 10 <HardSpace[get_pin ffa/Q]
```

The following example creates a collection of maximum 10 violation source for the first two *nworst* endpoints.

```
pt_shell> get_noise_violation_sources -max_sources_per_endpoint 10
-nworst_endpoint 2
```


SEE ALSO

`update_noise(2)`
`set_noise_parameters(2)`
`report_noise(2)`
`report_noise_violation_sources(2)`

get_object_name

Gets the name of objects in the given collection.

SYNTAX

```
string get_object_name  
collection
```

Data Types

```
collection          string
```

ARGUMENTS

```
collection  
    Specifies the collection.
```

DESCRIPTION

The **get_object_name** command is a convenient way to get the *full_name* attribute of objects. When multiple objects are passed to the **get_attribute** command, the values are returned as a Tcl list, containing one entry for each object.

EXAMPLES

The following example shows how to use the **get_object_name** command:

```
pt_shell> get_object_name [get_cells i1]  
i1  
pt_shell> get_attribute [get_cells {i1 i2}] full_name  
{i2 i3} i1  
pt_shell> get_object_name [get_cells i*]  
{i1 i2 i3}
```

SEE ALSO

```
collections (2)  
get_attribute (2)
```

get_path_groups

Creates a collection of path groups from the current design. You can assign these path groups to a variable or pass them into another command.

SYNTAX

```
collection get_path_groups
[-quiet]
[-regex]
[-nocase]
[-filter expression]
patterns
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

- quiet
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.
- regex
Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, it modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.
- nocase
When combined with the -regex option, it makes matches case-insensitive. You can use the -nocase option only when you also use the -regex option.
- filter *expression*
Filters the collection with the *expression* value. For any path groups that match the *patterns* values, the expression is evaluated based on the path group's attributes. If the expression evaluates to true, the path group is included in the result.
- patterns*
Matches path group names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_path_groups** command creates a collection of path groups from the current design that match certain criteria. The command returns a collection if any path groups match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Path groups control the optimization cost function and also affect timing reports.

You can use the **get_path_groups** command at the command prompt, or you can nest it as an argument to another command (for example, **query_objects**). In addition, you can assign the **get_path_groups** result to a variable.

When issued from the command prompt, the **get_path_groups** command behaves as though the **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_path_groups** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_path_groups** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following generates a timing report for each path group matching "Z*".

```
pt_shell> foreach_in_collection grp [get_path_groups Z*] { \  
?          report_timing -group $grp \  
?          }
```

SEE ALSO

```
collections(2)  
foreach_in_collection(2)  
group_path(2)  
query_objects(2)  
report_path_group(2)  
collection_result_display_limit(3)
```

get_pins

Creates a collection of pins from the netlist. You can assign these pins to a variable or pass them into another command.

SYNTAX

```
collection get_pins
[-hierarchical]
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[-leaf]
[patterns | -of_objects objects]
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-hierarchical

Searches for pins level-by-level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the **find** UNIX command. For example, if there is a pin block1/adder/D[0], a hierarchical search finds it using "adder/D[0]". You cannot use the *-hierarchical* option with the *-of_objects* option.

-filter *expression*

Filters the collection with the *expression*. For any pins that match *patterns* (or *objects*), the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The *-regexp* and *-exact* options are mutually exclusive.

-nocase

When combined with the *-regexp* option, this makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regexp* option.

-exact

Disables simple pattern matching. This is used when searching for objects

that contain the "*" and "?" wildcard characters. The *-exact* and *-regexp* options are mutually exclusive.

-leaf

You can use this option only with the *-of_objects* option. For any nets in the *objects* argument to the *-of_objects* option, only pins on leaf cells connected to those nets will be included in the collection. In addition, hierarchical boundaries are crossed in order to find pins on leaf cells.

-of_objects *objects*

Creates a collection of pins connected to the specified objects. Each object is a named cell or net, or cell collection or pin collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one. In addition, you cannot use the *-hierarchical* option with the *-of_objects* option.

patterns

Matches pin names against patterns. The *patterns* option can include the wildcard characters "*" and "?" or regular expressions, based on the *-regexp* option. The *patterns* option can also include collections of type pin. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

DESCRIPTION

The **get_pins** command creates a collection of pins in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any pins match the *patterns* or *objects* options and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

When used with the *-of_objects* option, the **get_pins** command searches for pins connected to any cells or nets specified in the *objects*. For net objects, there are two variations of pins that will be considered. By default, only pins connected to the net at the same hierarchical level are considered. When combined with the *-leaf* option, only pins connected to the net that are on leaf cells are considered. In this case, hierarchical boundaries are crossed in order to find pins on leaf cells. Note that the *-leaf* option has no effect on the pins of cells.

If no *patterns* (or *objects*) match any objects, and the current design is not linked, the design automatically links.

When a cell has bus pins, the **get_pins** command can find them in several ways. For example, if cell u1 has a bus "A" with indices 2 to 0, and the *bus_naming_style* for your design is "%s[%d]", then to find these pins, you can use "u1/A[*]" as the pattern. You can also find the same three pins with "u1/A" as the pattern.

Regular expression matching is the same as in the **regexp** Tcl command. When using the *-regexp* option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_pins** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_pins** command result to a variable.

When issued from the command prompt, the **get_pins** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_pins** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_pins** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the 'CP' pins of cells that begin with 'o'. Although the output looks like a list, it is just a display.

```
pt_shell> get_pins o*/CP
{"o_reg1/CP", "o_reg2/CP", "o_reg3/CP", "o_reg4/CP"}
```

The following example shows that given a collection of cells, you can query the pins connected to those cells.

```
pt_shell> set csel [get_cells o_reg1]
{"o_reg1"}
pt_shell> query_objects [get_pins -of_objects $csel]
{"o_reg1/D", "o_reg1/CP", "o_reg1/CD", "o_reg1/Q", "o_reg1/QN"}
```

The following example shows the difference between getting local pins of a net and leaf pins of net. In this example, NET1 is connected to i2/a and reg1/QN. Cell i2 is hierarchical. Within i2, port a is connected to the U1/A and U2/A.

```
pt_shell> get_pins -of_objects [get_nets NET1]
{"i2/a", "reg1/QN"}
pt_shell> get_pins -leaf -of_objects [get_nets NET1]
{"i2/U1/A", "i2/U2/A", "reg1/QN"}
```

The following example shows how to create a clock using a collection of pins.

```
pt_shell> create_clock -period 8 -name CLK [get_pins o_reg*/CP]
1
```

SEE ALSO

collections(2)
create_clock(2)
filter_collection(2)
get_cells(2)
link_design(2)

```
query_objects(2)
regexp(2)
collection_result_display_limit(3)
```


get_ports

Creates a collection of ports from the current design or instance. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection get_ports
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[-exact]
[patterns | -of_objects objects]
```

Data Types

<i>expression</i>	string
<i>objects</i>	list
<i>patterns</i>	list

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* option. For any ports that match *patterns*, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the cell is included in the result.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The *-regexp* and *-exact* options are mutually exclusive.

-nocase

When combined with the *-regexp* option, makes matches case-insensitive. You can use the *-nocase* option only when you also use the *-regexp* option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the "*" and "?" wildcard characters. The *-exact* and *-regexp* options are mutually exclusive.

-of_objects *objects*

Creates a collection of ports connected to the specified objects. Each object is a named net collection. The *-of_objects* and *patterns* options are mutually exclusive; you can specify only one.

patterns2

Matches port names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the *-regexp* option. Patterns can also include collections of type port. The *patterns* and *-of_objects* options are mutually exclusive; you can specify only one.

DESCRIPTION

The **get_ports** command creates a collection of ports in the current design or instance that match certain criteria. The command returns a collection if any ports match the *patterns* option and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

If the **port_search_in_current_instance** variable is true, then the **get_ports** command gets the ports of current instance. If the **port_search_in_current_instance** variable is false (the default), then the **get_ports** command gets the ports of the current design.

Regular expression matching is the same as in the **regexp** Tcl command. When using the *-regexp* option, take care in the way you quote the *patterns* and filter the *expression* option; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_ports** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_ports** command result to a variable.

When issued from the command prompt, the **get_ports** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_ports** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command, use the **get_ports** command as an argument to the **query_objects** command. For example, use this to display the object class.

For information about collections and the querying of objects, see the **collections** man page. In addition, refer to the **all_inputs** and **all_outputs** man pages, which also create collections of ports.

EXAMPLES

The following example queries all input ports beginning with 'mode'. Although the output looks like a list, it is just a display.

```
pt_shell> get_ports "mode*" -filter {direction == in}
{"mode[0]", "mode[1]", "mode[2]"}
```

The following example sets the driving cell for ports beginning with "in" to an FD2.

```
pt_shell> set_driving_cell -cell FD2 -library my_lib [get_ports in*]
```

The following example reports ports connected to nets matching the pattern "bidir*".

```
pt_shell> report_port [get_ports -of_objects [get_nets "bidir*"]]
```

SEE ALSO

```
all_inputs(2)
all_outputs(2)
collections(2)
filter_collection(2)
query_objects(2)
regexp(2)
port_search_in_current_instance(3)
collection_result_display_limit(3)
```

get_power_domains

Create a collection of power domains in the current design.

SYNTAX

```
string get_power_domains
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[patterns]
[-of_objects cells]
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list
<i>cells</i>	string

ARGUMENTS

- filter *expression***
Filters the collection with *expression*. For any power domains that match *patterns*, the expression is evaluated based on the power domain's attributes. If the expression evaluates to true, the power domain is included in the result. This option works the same as the **filter** command in dc_shell.
- quiet**
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.
- regexp**
Uses the *patterns* argument as real regular expressions rather than simple wildcard patterns.
- nocase**
Makes matches case-insensitive.
- patterns***
Matches power domain names against patterns. Patterns can include the wildcard characters * (asterisk) and ? (question mark). For more details about using and escaping wildcards, refer to the **wildcards** man page. The *patterns* and **-of_objects** options are mutually exclusive.
- of_objects *cells***
Returns a collection of power domain that owns the cells. The *patterns* and **-of_objects** options are mutually exclusive.

DESCRIPTION

The **get_power_domains** command creates a collection of power domains in the current

design, that match certain criteria. The command returns a collection handle (identifier) if any power domains match the *patterns* or *-of_objects* value and pass the filter, if specified. If no objects match the criteria, the empty string is returned.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example queries the power domain and creates collection of it.

```
pt_shell> create_power_domain TOP_DOMAIN
1
pt_shell> create_power_domain SUB_DOMAIN -power_down \
        -power_down_ctrl [get_nets pd_ctrl] \
        -object_list [get_cells mid1]
1
pt_shell> get_power_domain {TOP_DOMAIN SUB_DOMAIN}
{"", ""}

prompt> get_power_domain TOP*
{"", ""}
```

SEE ALSO

report_power_domain(2)

get_power_group_objects

Return a collection of cells in a power group.

SYNTAX

```
collection get_power_group_objects  
group_names
```

Data Types

```
group_names      list
```

ARGUMENTS

```
group_names  
    Specifies the power groups of which the collection of cells will be returned.
```

DESCRIPTION

The **get_power_group_objects** command returns a collection of cells contained by the specified power groups.

EXAMPLES

In the following example, the cells included in the newly created power group are displayed.

```
pt_shell> create_power_group -name my_group [get_cells "u1 u2"]  
1  
pt_shell> get_power_group_objects my_group  
{"u1", "u2"}
```

SEE ALSO

```
create_power_group(2)  
report_power_groups(2)  
remove_power_groups(2)
```

get_power_switches

Create a collection of UPF power switches in the current design.

SYNTAX

```
string get_power_switches
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[patterns]
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-filter *expression*
Filters the collection with the *expression* argument. For any UPF power switches that match the *patterns* argument, the expression is evaluated based on the power switch's attributes. If the expression evaluates to true, the power switch is included in the result. This option works the same as the **filter** command in pt_shell.

-quiet
Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp
Uses the *patterns* argument as real regular expressions rather than simple wildcard patterns.

-nocase
Makes matches case-insensitive.

patterns
Matches supply net names against patterns. Patterns can include the wildcard characters "*" (asterisk) and "?" (question mark). For more details about using and escaping wildcards, refer to the **wildcards** man page. The *patterns* and *-of_objects* arguments are mutually exclusive.

DESCRIPTION

The **get_power_switches** command creates a collection of UPF power switch objects in the current design, that match certain criteria. The command returns a collection handle (identifier) if any supply nets match the *patterns* or *-of_objects* arguments and pass the filter (if specified). If no objects match the criteria, an empty string is returned.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example creates a power switch and creates collection of it.

```
pt_shell> create_power_domain TOP_DOMAIN
1
pt_shell> create_supply_net SN1 -domain TOP_DOMAIN
1
pt_shell> create_supply_net SN2 -domain TOP_DOMAIN
1
pt_shell> create_power_switch my_switch -domain TOP_DOMAIN \
    -output_supply_port {OUT SN2} \
    -input_supply_port {IN SN2} \
    -on_state {state1 IN {CNTL} } \
    -control_port {ctrl CNTL}
```

```
prompt> get_power_switches *
{"my_switch"}
```

SEE ALSO

```
create_power_switch(2)
report_power_switches(2)
```


get_qtm_ports

Creates a collection of Quick Timing Model (QTM) ports. You can assign these QTM ports to a variable or pass them into another command.

SYNTAX

```
collection get_qtm_ports
[-filter expression]
pattern
```

Data Types

<i>expression</i>	string
<i>pattern</i>	list

ARGUMENTS

-filter *expression*
Filters the collection with the *expression* option. For any QTM ports that match the *patterns* option, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the QTM port is included in the result.

patterns
Matches QTM port names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_qtm_ports** command creates a collection of QTM ports from the current QTM model that match certain criteria. The command returns a collection if any QTM ports match the *patterns* option and pass the filter (if specified). If no objects match the criteria, an empty string is returned.

To create a QTM port, use the **create_qtm_port** command.

See the **collections** man page for information about collections and the querying of objects.

For a basic description about QTM, refer to the **create_qtm_model** man page. For a more detailed description about QTM, refer to the *PrimeTime Modeling User Guide*.

EXAMPLES

The following command applies the **set_qtm_port_load** command to all QTM ports in the model matching "IN*".

```
pt_shell> set_qtm_port_load -value 2.0 [get_qtm_ports "IN*"]
```

The following command creates a delay arc from QTM port A to all QTM ports in the

```
model matching "OUT*".
```

```
pt_shell> create_qtm_delay_arc -from A -to [get_qtm_ports "OUT*"] -value 3.0
```

SEE ALSO

```
create_qtm_constraint_arc(2)  
create_qtm_delay_arc(2)  
create_qtm_model(2)  
create_qtm_port(2)  
report_qtm_model(2)  
save_qtm_model(2)  
set_qtm_port_drive(2)  
set_qtm_port_load(2)
```

get_random_numbers

Generates a list of random numbers for a specified variation.

SYNTAX

```
list get_random_numbers  
-sample_size sample_size  
-seed seed  
variation_object Variation object
```

Data Types

<i>sample_size</i>	int
<i>seed</i>	int
<i>variation_object</i>	collection

ARGUMENTS

```
-sample_size sample_size  
    Specifies the sample size.
```

```
-seed seed  
    Specifies the seed for this particular stream. This number must be a positive  
    integer. A different number will generate a different sample.
```

```
variation_object  
    Specifies the variation to generate random samples from. There can only be  
    one object.
```

DESCRIPTION

Enables the generation of random numbers in a list.

EXAMPLES

In the following example, the returned list contains a sample of size 500 of random numbers generated from a variation with a normal distribution with mean 15.0 and standard deviation 2.0. The seed for this random number stream is 17777.

```
pt_shell> create_variation -name test -type normal -values { 15.0 2.0 }  
_sel21  
pt_shell> set rnd_list [get_random_numbers -sample_size 500 -  
seed 17777 [get_variations test]]  
10.613445 14.553537 13.051448 16.382267 ...
```

SEE ALSO

`create_variation(2)`

get_selection

Returns the list of objects currently selected in the GUI or information about the selected objects.

SYNTAX

```
collection get_selection
[-type object_type]
[-design design]
[-more_than more]
[-fewer_than fewer]
[-count]
[-num num]
[-name slct_bus]
[-slct_targets target_slct_bus]
[-slct_targets_operation operation]
[-create_slct_buses]
[-type_list]
```

Data Types

<i>object_type</i>	string
<i>design</i>	string
<i>more</i>	int
<i>fewer</i>	int
<i>num</i>	int
<i>slct_bus</i>	string
<i>target_slct_bus</i>	string
<i>operation</i>	string

ARGUMENTS

-type *object_type*
Specifies the type of selected object. If specified, the selection will be filtered with this type. The *object_type* option can be one of the following:

Format	Description
design	design object
port	port object
net	net object
cell	cell object
pin	pin object
path	timing path object

-design *design*
Returns objects that only belong to the *design* option.

-more_than *more*
Returns true if the selection bus contains more than the *more* option.

-fewer_than fewer
Returns true if the selection bus contains more than the *fewer* option.

-count
Returns the number of elements contained in the selection bus.

-num num
Does not return more than the *num* option.

-name slct_bus
Specifies the selection bus from which the selection is taken. The default is "global", which is the default global selection bus.

-slct_targets target_slct_bus
Specifies the name of a selection bus in which to store the result. By default, the result of this command is returned in a collection. If this option is specified, the *target_slct_bus* selection bus is used instead. The *target_slct_bus* option is also returned as a result of the command.

-slct_targets_operation operation
Specifies which operation should be used when storing the result in the *target_slct_bus* selection bus. This is only allowed if you specify the *-slct_targets* option. Legal operations are clear, add, and remove. If the add operation is specified, the result of the **get_selection** command is added to the target selection bus. This is the default behavior. If the clear operation is specified, the target selection bus is first cleared before the result of the **get_selection** command is added. If the remove operation is specified, the result of the **get_selection** command is removed from the target selection bus.

-create_slct_buses
Specifies that a new selection bus is created in which the result is stored. The name of the newly created selection bus is returned.

-type_list
Returns a Tcl list of the types of elements contained in the selection bus.

DESCRIPTION

The **get_selection** command constructs a collection from the objects that are currently selected in the GUI and returns the collection. If no objects are selected, an empty string is returned. If no *-type* option is specified, the command returns a heterogeneous collection of all objects currently selected. If the *-type* option is used, the collection is filtered to a homogeneous collection containing only objects of the type specified.

You can use the **get_selection** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_selection** command result to a variable.

When issued from the command prompt, the **get_selection** command behaves as though the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_selection** command provides a fast way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command (for example, if you want to display the object class), use the **get_selection** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example returns the collection of the selected cell objects.

```
pt_shell> get_selection -type cell
```

```
{"I_PRGRM_CNT_TOP", "I_DATA_PATH", "I_REG_FILE", "I_STACK_TOP"}
```

The next example assigns the collection to the **collect_a** variable, which is passed to another **query_objects** command.

```
pt_shell> set collect_a [get_selection -type cell]
```

```
{"I_PRGRM_CNT_TOP", "I_DATA_PATH", "I_REG_FILE", "I_STACK_TOP"}
```

```
pt_shell> query_objects $collect_a
```

```
{"I_PRGRM_CNT_TOP", "I_DATA_PATH", "I_REG_FILE", "I_STACK_TOP"}
```

The next example assigns the collection to the **pins** variable, which is passed to another **report_timing** command to generate a report on the ten worst paths to the selected pin. The output from the **report_timing** command is not shown here for brevity.

```
pt_shell> set pins [get_selection -type pin]
```

```
{"OUT[0]"}
```

```
pt_shell> report_timing -to $pins -nworst 10 -maxpaths 10
```

SEE ALSO

```
collections(2)  
filter(2)  
filter_collection(2)  
query_objects(2)  
change_selection(2)
```

get_si_bottleneck_nets

Identify the crosstalk bottlenecks in the design. This is useful when the major sources of violations come from crosstalk effects.

SYNTAX

```
int get_si_bottleneck_nets
-cost_type          type
[-slack_lesser_than slack_limit]
[-max_nets          count]
[-significant_digits digits]
[-nosplit]
[-include_clock_nets]
[-minimum_active_aggressors active_aggressor_count]
[-min]
[-max]
```

Data Types

<i>type</i>	list
<i>count</i>	int
<i>slack_limit</i>	float
<i>active_aggressor_count</i>	int

ARGUMENTS

-cost_type *type*
Sorts the nets based on the cost. The *delta_delay*, *delta_delay_ratio*, *total_victim_delay_bump* are the costs of the victim net. And the *delay_bump_per_aggressor* for aggressor nets of the selected victim nets.

-slack_lesser_than *slack_limit*
Applies the cost function only to the victim nets whose slack is less than the slack limit. The default is 0.0.

-max_nets *count*
Sets the maximum number of nets to be reported. The default is 20.

-significant_digits *digits*
Sets the number of digits to display: Range: 0 to 13. The default is the same as the **report_default_significant_digits** global variable.

-nosplit
Use to not split lines if column overflows.

include_clock_nets
Includes the clock nets for bottleneck. By default, clock nets are excluded from the bottleneck analysis.

-minimum_active_aggressors *active_aggressor_count*
Sets the minimum number of active aggressors for the net to be selected. The default is 1.

-min

Use for the minimum analysis. For example, the *slack_limit* applies to the minimum slack of the net in the net selection.

-max

Use for the maximum analysis. For example, the *slack_limit* applies to the maximum slack of the net in the net selection.

DESCRIPTION

This command has the same behavior as the **report_si_bottleneck** command, but returns nets instead of reporting on them. For more information, see the **report_si_bottleneck** command.

EXAMPLES

The following example gets PrimeTime SI bottleneck. It has an arc with specified start and end points.

```
pt_shell> get_si_bottleneck_nets -cost_type delta_delay -significant_digits 6
```

SEE ALSO

report_si_bottleneck (2)
report_default_significant_digits (3)
set_coupling_separation (2)
size_cell (2)

get_supply_nets

Create a collection of UPF supply nets in the current design.

SYNTAX

```
string get_supply_nets  
[-filter expression]  
[-quiet]  
[-regexp]  
[-nocase]  
[patterns]
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* option. For any UPF supply nets that match the *patterns* option, the expression is evaluated based on the supply net's attributes. If the expression evaluates to true, the supply net is included in the result. This option works the same as the **filter** command in dc_shell.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Uses the *patterns* argument as real regular expressions rather than simple wildcard patterns.

-nocase

Makes matches case-insensitive.

patterns

Matches supply net names against patterns. Patterns can include the wildcard characters "*" (asterisk) and "?" (question mark). For details about using and escaping wildcards, refer to the **wildcards** man page. The *patterns* and *-of_objects* arguments are mutually exclusive.

DESCRIPTION

The **get_supply_nets** command creates a collection of UPF supply nets in the current design that match certain criteria. The command returns a collection handle (identifier) if any supply nets match the *patterns* or *-of_objects* arguments and pass the filter (if specified). If no objects match the criteria, an empty string is returned.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example creates a supply net and creates collection of it.

```
pt_shell> create_power_domain TOP_DOMAIN
1
pt_shell> create_supply_net SN1 -domain TOP_DOMAIN
1
pt_shell> get_supply_nets {SN1}
{"SN1"}

prompt> get_supply_nets SN*
{"SN1"}
```

SEE ALSO

```
create_supply_net(2)
report_supply_net(2)
```

get_supply_ports

Create a collection of UPF supply ports in the current design.

SYNTAX

```
string get_supply_ports
[-filter expression]
[-quiet]
[-regexp]
[-nocase]
[patterns]
```

Data Types

<i>expression</i>	string
<i>patterns</i>	list

ARGUMENTS

-filter *expression*

Filters the collection with the *expression* option. For any UPF supply ports that match the *patterns* option, the expression is evaluated based on the supply port's attributes. If the expression evaluates to true, the supply port is included in the result. This option works the same as the **filter** command in dc_shell.

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Uses the *patterns* option as real regular expressions rather than simple wildcard patterns.

-nocase

Makes matches case-insensitive.

patterns

Matches supply port names against patterns. Patterns can include the wildcard characters "*" (asterisk) and "?" (question mark). For details about using and escaping wildcards, refer to the **wildcards** man page. The *patterns* and *-of_objects* options are mutually exclusive.

DESCRIPTION

The **get_supply_ports** command creates a collection of UPF supply ports in the current design that match certain criteria. The command returns a collection handle (identifier) if any supply ports match the *patterns* or *-of_objects* options and pass the filter (if specified). If no objects match the criteria, an empty string is returned.

For information about collections and the querying of objects, see the **collections** man page.

EXAMPLES

The following example creates a supply port and creates a collection of it.

```
pt_shell> create_power_domain TOP_DOMAIN
1
pt_shell> create_supply_port SN1 -domain TOP_DOMAIN
1
pt_shell> get_supply_ports {SN1}
{"SN1"}

prompt> get_supply_ports SN*
{"SN1"}
```

SEE ALSO

`create_supply_port(2)`

get_switching_activity

Gets switching activity annotation on nets, pins, ports and cells of the current design.

SYNTAX

```
int get_switching_activity
    [-toggle_rate]
    [-glitch_rate]
    [-static_probability]
    [-state_condition state]
    [-path_sources name_list]
    [-average_activity]
    [-exclude exclusion_group]
    [-include_only inclusion_group]
    [-toggle_limit limit]
    [-only_related_clock clock_name]
    [-rise]
    [-fall]
    object_list

string state
string name_list
list object_list
string exclusion_group
string inclusion_group
int limit
string clock_name
```

ARGUMENTS

-toggle_rate
Specifies the toggle rate (rise and fall if applicable) to be reported for the given design object. Given a design object, the command returns the list comprised of object name, simple toggle rate value (without glitches) and annotation source indicator. The unit of returned toggle rate value is number of toggle counts per target technology library time unit. If the object has no toggle rate value annotation, then this command returns the value -1. If the given object is not found in the design then the value of -2 is returned. annotation source indicator can have possible values, of: 1) file : implies that toggle_rate_value was annotated via a SAIF or vcd file 2) propagated : implies that toggle_rate_value is propagated using random vectors from annotated design points 3) default : implies that toggle_rate_value is default 4) UNINITIALIZED : implies that toggle_rate_value is uninitialized 5) create_clock : implies that the toggle_rate_value was annotated by a create_clock command 6) set_switching_activity : implies that the toggle_rate_value was annotated by a set_switching_activity command 7) set_case_activity : implies that the toggle_rate_value was annotated by a set_case_analysis command 8) implied : implies that the activity was implied without random vector propagation from annotated design points
The object name is the full name relative to the current instance.

-glitch_rate

Specifies the glitch rate (rise and fall if applicable) to be reported for the given design object. Given a design object, the command returns the list comprised of object name, glitch rate value and annotation source indicator. The unit of returned glitch rate value is number of glitch counts per target technology library time unit. If the object has no glitch rate value annotation, then this command returns the value -1. If the given object is not found in the design then the value of -2 is returned.

-static_probability

Specifies the static probability value to be reported for the given design object. Given a design object, the command returns the list comprised of object name, simple static probability and annotation source indicator. The static probability is defined as the percentage of time the signal is at logic state 1. If the object has no static probability value annotation, then this command returns the value -1. If the given object is not found in the design then the value of -2 is returned.

If none of the options **-toggle_rate**, **-glitch_rate** and **-static_probability** are set, then the command assumes all three options to be true, and returns **toggle_rate**, **glitch_rate** and **static_probability** values.

-state_condition *state*

Specifies the state condition when getting state-dependent toggle rate and glitch rate on pins or state-dependent static probabilities on cells. State dependent toggle rate and glitch rate can be retrieved for a pin when the internal power of the library cell pin is characterized with state-dependent power tables. State-dependent static probabilities for a cell can be annotated when the cell leakage power is characterized with state-dependent power tables. The state condition specified with this argument must be logically equivalent to a state condition in the internal / leakage power characterization. The state condition should be enclosed between " ". Moreover, if the user wants to get switching activity information for the default state condition then the argument of **-state** option should be "default". If **-state_condition** is applied to a cell, then only static probability can be reported. Only leakage power is associated with cells. If **-state** is applied to a pin, then the tool will look for the condition in the internal power tables and report the toggle rate/glitch rate/static probability for that pin.

Given a design object, the command returns the list comprised of object name, *state*, static probability for the *state* and the annotation source indicator. If *state* does not represent a valid state for the cell, then the value of -2 is returned. If the cell has no state-dependent static probability annotation for the *state*, then the value of -1 is returned.

-path_sources *name_list*

Specifies the path sources when getting path-dependent toggle rate and glitch rate on pins. This is used when the library cell pin has path-dependent internal power. The path source(s) specified with this argument must be the same as those in the internal power characterization. The **-path** argument takes a path condition (a list of input pins). Again the path condition after **-path** options should be enclosed between " ". If there are multiple input pins in path condition then each input pin should be separated by a space, e.g. "A B".

Please note that if *name_list* is given using **-path_sources** argument, then *state* should also be provided using **-state_condition**.

Given a design object, the command returns list comprised of object name, state_condition, source pin name, toggle rate value, toggle rate annotation source indicator, glitch rate value, and glitch rate annotation source indicator. If the combination of *state* and *name_list* does not represent a valid state-dependent-path-dependent (SDPD) condition for the design object, then the value of -2 is returned. If the design object does not have toggle rate or glitch rate annotation for the given SDPD conditions, then the value of -1 is returned.

-rise|-fall

Used with **-state_condition** option. This argument tells the command to report either rise or fall transitions. If **-state_condition** *state* argument is given and neither **-rise** or **-fall** arguments are given then by default command assumes both the arguments to be true. If **-rise** or **-fall** arguments are given along with **-static_probability** argument, then **-rise** and **-fall** arguments will be ignored as static probability does not depend on rise or fall transitions type.

This option filters out objects in the *object_list*, so that switching activity is not reported for objects that meet the specified criteria.

When the **-average_activity** option is used, the **-exclude** option serves to filter out nets before the average is taken.

For more information about how to use the **-exclude** and **-include_only** options, please read the appropriate section in the man page for

report_switching_activity.

This option filters out objects in the *object_list*, so that switching activity is reported only on objects that meet the specified criteria.

When the **-average_activity** option is used, the **-include_only** option serves to filter out nets before the average is taken.

For more information about how to use the **-exclude** and **-include_only** options, please read the appropriate section in the man page for

report_switching_activity.

Changes the definition of what is considered to be low activity. This definition can be used by the **-exclude** and **-include_only**, depending on the criteria given to those options. The value of the argument represents the maximum number of toggles considered to be low activity. The default value is 1. The number of toggles on a net is defined to be the toggle rate times the simulation duration. For vector free power analysis (where no SAIF file or VCD has been used) the default simulation duration is 10000 ns. If a SAIF has been used, the duration is taken from the SAIF file. If a VCD file has been used to annotate activity, the duration is the duration of the VCD simulation.

The **-toggle_limit** option has no effect unless the **-exclude** or **-include_only** options are used. The **-toggle_limit** option only affects the behavior of these other options.

When this option is used, the **-static_probability**, **-rise**, **-fall**, and **-path_sources** options cannot be used

When the **-average_activity** option is used, the *object_list* should consist only of hierarchical cells. For each hierarchical cell given, the average toggle rate and glitch rate of nets within the hierarchical cell will be computed and reported.

In this case the **-include_only** and **-exclude** options play the role of filtering out nets in the hierarchical cell before the average toggle rate is computed. Static probabilities are never averaged. If the **-average_activity** option is used, only average toggle rate and average glitch rate will be reported. Since the nets in the hierarchical cell may have different sources for the activity

information, the activity source for the toggle and glitch rate is reported as "averaged".

`only_related_clock` *clock_name*

Only nets and objects that are in the clock domain *clock_name* are included. Objects not in the domain are filtered out. When `-average_activity` is used, objects are filtered out before averaging takes place.

`object_list`

Specifies a list of nets, pins, ports or cells in the current design for which the `static_probability`, `toggle_rate`, and `glitch_rate` switching activity values are to be reported.

When the `-average_activity` option is used, the `object_list` should consist of hierarchical cells whose average switching activity on nets in the cell is to be computed.

DESCRIPTION

This command can be used to report different kind of switching activity information on design nets, ports, pins and cells. These include simple toggle rate, glitch rate and static probability on nets, ports and pins; state and path dependent toggle rate and glitch rate on cell pins, and state dependent static probabilities on cells.

Toggle rates, glitch rates and static probabilities are reported using the `-toggle_rate`, `-glitch_rate` and `-static_probability` arguments respectively. The toggle rate, glitch rate and static probability can be made state dependent by specifying the state condition with the `-state_condition` argument. The toggle rate and glitch rate can be made path dependent by specifying the path source(s) with the `-path_sources` argument.

The design objects that will be annotated with switching activity can only be specified explicitly as a list of objects. Note that *object_list* should always be provided to the command.

Average activity statistics for a hierarchical block, or portions of a hierarchical block can be obtained using the `-average_activity` option.

For more statistics on the switching activity annotation on the current design use the `report_switching_activity` command.

If the `power_analysis_mode` has been set to `averaged` or `leakage_variation`, and if a vcd file has been read with the `read_vcdP` command (without the `-pipe` option), then the tool will annotate the activity from the vcd prior to reporting the switching activity. In addition, unannotated objects where the activity can be derived accurately without random vector simulation (e.g. clock nets, Qbar pins where the Q pin is annotated, constant nets) will be reported by `report_switching_activity` as having *impliedP* activity. Activity propagation for other unannotated nodes will not happen until `update_power` is run.

If the `power_analysis_mode` has been set to `time_based`, then `get_switching_activity` will not have access to toggle rate information prior to running `update_power`. The command will report whether a net is uninitialized or whether it will be annotated from the VCD file, but since the VCD file has not been read yet, the toggle rate (and other values) will be reported as -1.

`get_switching_activity`

Use of the filtering options, **-exclude** and **-include_only**, allow the user to filter out objects before reporting switching activity. This can be useful, for instance, to report only nets with no switching activity. In this case the *object_list* could be all nets in the design (obtained by **[get_net * -hier -top]**), while the option **-include_only {no_switching_activity}** filters the large set to return a list of only those nets with no switching activity.

EXAMPLES

The following **get_switching_activity** command reports simple toggle rate value, glitch rate value and static probability value on all design input ports.

```
pt_shell> get_switching_activity -toggle_rate -glitch_rate -
static_probability [all_inputs]
pt_shell> get_switching_activity [all_inputs]
```

The following example reports different switching activity values on design output ports.

```
pt_shell> get_switching_activity -toggle_rate [all_outputs]
pt_shell> get_switching_activity -glitch_rate [all_outputs]
pt_shell> get_switching_activity -static_probability [all_outputs]
pt_shell> get_switching_activity -toggle_rate -static_probability
[all_outputs]
pt_shell> get_switching_activity -glitch_rate -static_probability
[all_outputs]
pt_shell> get_switching_activity -glitch_rate -toggle_rate [all_outputs]
pt_shell> get_switching_activity -glitch_rate -toggle_rate -
static_probability [all_outputs]
```

The following example reports state dependent static probabilities on the cell orl1:

```
pt_shell> get_switching_activity -static_probability -state_condition "A &
B" [get_cell orl1]
pt_shell> get_switching_activity -state_condition "A & B" [get_cell orl1]
pt_shell> get_switching_activity -static_probability -state_condition "A & !
B" [get_cell orl1]
pt_shell> get_switching_activity -static_probability -state_condition "! A &
B" [get_cell orl1]
pt_shell> get_switching_activity -static_probability -state_condition "! A &
! B" [get_cell orl1]
pt_shell> get_switching_activity -static_probability -state_condition
"default" [get_cell orl1]
```

The following example reports simple and path dependent toggle rates and glitch rates on the output pin Y of the cell xor1:

```
pt_shell> get_switching_activity -toggle_rate -glitch_rate [get_pin xor1/Y]
pt_shell> get_switching_activity -toggle_rate -glitch_rate -path_sources "A"
-state_condition "B" [get_pin xor1/Y]
pt_shell> get_switching_activity -rise -fall -toggle_rate -glitch_rate -
path_sources "A" -state_condition "B" [get_pin xor1/Y]
pt_shell> get_switching_activity -rise -toggle_rate -glitch_rate -
path_sources "A" -state_condition "B" [get_pin xor1/Y]
```

```

pt_shell> get_switching_activity -fall -toggle_rate -glitch_rate -
path_sources "A" -state_condition "B" [get_pin xor1/Y]
pt_shell> get_switching_activity -toggle_rate -glitch_rate -path_sources "B"
-state_condition "A" [get_pin xor1/Y]
pt_shell> get_switching_activity -toggle_rate -glitch_rate -path_sources "A
B" -state_condition "default" [get_pin xor1/Y]

```

The following example reports the average switching activity on certain nets within the hierarchical cell "state_machine1". The nets used are those that are driven by sequential cells and have annotated switching activity.

```

pt_shell> get_switching_activity -toggle_rate -average_activity -
include_only {sequential & annotated} [get_cell state_machine1]

```

The following example reports the switching activity on every pin that is an output of a sequential cell. This might be useful for checking to see that switching activity annotation to sequential cells happened properly when the user invoked a command such as **read_saif**.

```

pt_shell> get_switching_activity -include_only {sequential} [get_pin * -
hier]

```

The following example reports the switching activity on every pin that is an output of a sequential cell or a black box and does not have annotated activity. This could be useful for generating a list of pins that the user might want to annotated before propagation of switching activity.

```

pt_shell> get_switching_activity -include_only {sequential,black_box &
!annotated} [get_pin * -hier]

```

The following example reports the switching activity on every net that has fewer than 10 toggles during simulation. Note that `get_switching_activity` reports toggle rates, not toggle counts. to obtain the toggle count of a net, multiply the rate by the simulation time.

```

pt_shell> get_switching_activity -toggle_limit 10 -include_only
{low_activity} [get_net * -hier -top]

```

The following example reports on a net in the **averaged** power analysis mode. The net is in the VCD file, and the **read_vcd** command had previously been given. This example runs **get_switching_activity** before and after `update_power`. The example shows that prior to power analysis, on the first invocation of a activity reporting command, the VCD activity is applied, then the toggle rate and source are reported. After power analysis, the toggle rate is also reported.

```

pt_shell> get_switching_activity "z[31]" Information: Reading file
vcd.dump.gz to annotate toggle rates on the design...
=====
Summary: Total number of nets = 1629 Number of annotated nets = 1629 (100.00%)
Total number of leaf cells = 1339 Number of fully annotated leaf cells = 1339
(100.00%)
=====
{"z[31]" 0.00330065 file 0 file 0.525616 file} pt_shell>
get_switching_activity "z[31]" {"z[31]" 0.00330065 file 0 file 0.525616 file}
pt_shell> update_power Information: Running average power analysis... 1

```

```
pt_shell> get_switching_activity "z[31]" {"z[31]" 0.00330065 file 0 file
0.525616 file}
```

The following example reports on a net in the **time_based** power analysis mode. The net is in the VCD file. This example runs **get_switching_activity** before and after **update_power**. The example shows that prior to power analysis, the activity source is known (file), but the toggle rate is unknown and is reported as -1. After **update_power**, the toggle rate is known and is reported.

```
pt_shell> get_switching_activity "z[31]" {"z[31]" -1 file -1 file -1 file}
pt_shell> update_power Information: The waveform options are: File name:
primetime_px.fsdb File format: fsdb Time interval: 0.01ns Hierarchical level:
1
Information: Power analysis is running, please wait ...
Information: analysis is done for time window (0ns - 9998.03ns)
1 pt_shell> get_switching_activity "z[31]" {"z[31]" 0.00330065 file 0 file
0.526482 file}
```

SEE ALSO

report_switching_activity (2), read_saif(2), reset_switching_activity(2),
set_switching_activity(2), report_power (2);

get_timing_arcs

Creates a collection of timing arcs for custom reporting and other processing. You can assign these timing arcs to a variable and get the desired attribute for further processing.

SYNTAX

```
string get_timing_arcs [-to to_list] [-from from_list] [-of_objects object_list] [-  
filter expression] [-quiet]  
list to_list  
list from_list  
list object_list  
string expression
```

ARGUMENTS

- to *to_list*
Specifies a list of to pins or to ports to get timing arcs for. All backward arcs from the specified pins or ports are considered.
- from *from_list*
Specifies a list of from pins or from ports to get timing arcs for. All forward arcs from the specified pins or ports are considered.
- of_objects *object_list*
Specifies cells or nets to get timing arcs for. If a cell is specified, all cell_arcs of that cell are considered. If a net is specified, all net_arcs of that net are considered.
- filter *filter_str*
Specifies a string that comprises a series of logical expressions describing a set of constraints you want to place on the collection of arcs. Each subexpression of a filter expression is a relation contrasting an attribute name with a value by means of an operator.
- quiet
Specifies that all messages are to be suppressed.

DESCRIPTION

The **get_timing_arcs** command creates a collection of arcs for custom reporting or other operations. Use the **foreach_in_collection** command to iterate among the arcs in the collection. You can use the **get_attribute** command to obtain information about the arcs. You can also use the filter expression to filter the arcs that satisfy the specified conditions. The following attributes are supported on timing arcs:

```
delay_max_fall  
delay_max_rise  
delay_min_fall  
delay_min_rise
```

```

from_pin
is_annotated_fall_max
is_annotated_fall_min
is_annotated_rise_max
is_annotated_rise_min
is_cellarc
is_disabled
is_user_disabled
mode
object_class
sdf_cond
sense
to_pin

```

One attribute of a timing arc is `from_pin` which is the pin or port from which the timing arc begins. In the same way, the `to_pin` is the pin or port at which the timing arc ends. In order to get more information about the `from_pin` and the `to_pin`, use the **get_attributes** command. The **object_class** attribute holds the name of the timing arc class **timing_arc**.

EXAMPLES

The following procedure gets the same arguments as the **get_timing_arcs** command and prints out some of the attributes of the timing arcs.

```

proc format_float {number {format_str "%.2f"}} {
    switch -exact -- $number {
        UNINIT { }
        INFINITY { }
        default {set number [format $format_str $number]}
    }
    return $number;
}

proc show_arcs {args} {
    set arcs [eval [concat get_timing_arcs $args]]
    echo [format "%15s    %-15s  %8s %8s  %s" "from_pin" "to_pin" \
        "rise" "fall" "is_cellarc"]
    echo [format "%15s    %-15s  %8s %8s  %s" "-----" "-----" \
        "----" "----" "-----"]

    foreach_in_collection arc $arcs {
        set is_cellarc [get_attribute $arc is_cellarc]
        set fpin [get_attribute $arc from_pin]
        set tpin [get_attribute $arc to_pin]
    }
}

```

```

set rise [get_attribute $arc delay_max_rise]
set fall [get_attribute $arc delay_max_fall]

set from_pin_name [get_attribute $fpin full_name]

set to_pin_name [get_attribute $tpin full_name]
echo [format "%15s -> %-15s  %8s %8s  %s" \
    $from_pin_name $to_pin_name \
    [format_float $rise] [format_float $fall] \
    $is_cellarc]
}
}

```

pt_shell> **show_arcs -of_objects ffa**

from_pin	to_pin	rise	fall	is_cellarc
-----	-----	----	----	-----
ffa/CP -> ffa/D		0.85	0.85	true
ffa/CP -> ffa/D		0.40	0.40	true
ffa/CP -> ffa/Q		1.34	1.42	true
ffa/CP -> ffa/QN		2.38	1.98	true
ffa/CD -> ffa/Q		1.00	0.82	true
ffa/CD -> ffa/QN		1.78	1.00	true

pt_shell> **show_arcs -from ffa/CP**

from_pin	to_pin	rise	fall	is_cellarc
-----	-----	----	----	-----
ffa/CP -> ffa/D		0.85	0.85	true
ffa/CP -> ffa/D		0.40	0.40	true
ffa/CP -> ffa/Q		1.34	1.42	true
ffa/CP -> ffa/QN		2.38	1.98	true

pt_shell> **show_arcs -from ffa/Q**

from_pin	to_pin	rise	fall	is_cellarc
-----	-----	----	----	-----
ffa/Q -> QA/A		0.00	0.00	false

pt_shell> **show_arcs -to ffa/***

from_pin	to_pin	rise	fall	is_cellarc
-----	-----	----	----	-----
ffa/CP -> ffa/D		0.85	0.85	true
ffa/CP -> ffa/D		0.40	0.40	true
a/Z -> ffa/D		0.00	0.00	false
CLK -> ffa/CP		0.00	0.00	false
RESET -> ffa/CD		0.00	0.00	false
ffa/CP -> ffa/Q		1.34	1.42	true
ffa/CD -> ffa/Q		1.00	0.82	true
ffa/CP -> ffa/QN		2.38	1.98	true
ffa/CD -> ffa/QN		1.78	1.00	true

SEE ALSO

collections (2), **foreach_in_collection** (2), **get_attribute** (2).

get_timing_paths

Creates a collection of timing paths for custom reporting and other processing. You can assign these timing paths to a variable or pass them into another command.

SYNTAX

```
string get_timing_paths
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
[-exclude exclude_list
  | -rise_exclude rise_exclude_list
  | -fall_exclude fall_exclude_list]
[-through through_list]
[-rise_through rise_through_list]
[-fall_through fall_through_list]
[-delay_type delay_type]
[-nworst paths_per_endpoint]
[-max_paths max_path_count]
[-group group_name]
[-true]
[-unique_pins]
[-true_threshold path_delay]
[-slack_greater_than greater_slack_limit]
[-slack_lesser_than lesser_slack_limit]
[-ignore_register_feedback feedback_slack_cutoff]
[-include_hierarchical_pins]
[-justify]
[-trace_latch_borrow]
[-start_end_pair]
[-dont_merge_duplicates]
[-pre_commands pre_command_string]
[-post_commands post_command_string]
[-path_type format]
[-pba_mode effort]
[path_collection]

list from_list
list rise_from_list
list fall_from_list
list to_list
list rise_to_list
list fall_to_list
list exclude_list
list rise_exclude_list
list fall_exclude_list
list through_list
list rise_through_list
list fall_through_list
string delay_type
```

```

int    paths_per_endpoint
int    max_path_count
list   group_name
float  path_delay
float  greater_slack_limit
float  lesser_slack_limit
float  feedback_slack_cutoff
string pre_command_string
string post_command_string
string format
string effort

```

ARGUMENTS

-from *from_list*

Specifies a list of from pins, ports, nets, or clocks to be reported. Path startpoints are typically the input ports or clock pins of registers. If you specify a clock, all startpoints are considered if they are clocked by the clock.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-to *to_list*

Specifies a list of to pins, ports, nets, or clocks to be reported. Path endpoints are typically the output ports or data pins of registers. If you specify a clock, all endpoints are considered if they are constrained by the clock.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-exclude *exclude_list*
 Specifies that only paths not including the named pins, ports, nets, cell instances are to be reported. Reporting will exclude all data paths from/through/to the named pins, ports, nets and cell instances. If a cell instance is specified, all pins of the cell are excluded. **-exclude** has higher precedence than **-from/-through/-to**. **-exclude** does not work with **-true** option. **-exclude** is exclusive with **-rise_exclude** or **-fall_exclude**. **-exclude** does not apply to borrowing path from **-trace_latch_borrow** option or clock path from **-path full_clock/full_clock_expanded** options.

-rise_exclude *rise_exclude_list*
 Same as the **-exclude** option, but applies only to paths rising at the named pins, ports, nets, cell instances.

-fall_exclude *fall_exclude_list*
 Same as the **-exclude** option, but applies only to paths falling at the named pins, ports, nets, cell instances.

-through *through_list*
 Specifies a list of through pins, ports, or nets to be reported. Only paths through the named pins are considered.
 You can specify many *through_list* groups by using multiple **-through** options. The objects specified within one **-through** option are assumed to be in OR mode. The group of objects specified with multiple **-through** options is assumed to be in AND mode.
 If you specify **-through** only once, PrimeTime reports only the paths that travel through one or more of the objects in the list.
 If you specify multiple **-through** options, PrimeTime reports only the paths that travel through one or more of the objects in each list. PrimeTime uses the exact order in which the **-through** options are listed; so to obtain correct results, you must ensure that this order is the same as that followed by the actual paths in the circuit.

-rise_through *rise_through_list*
 Specifies the same as the **-through** option, except that the path must rise through the objects specified.

-fall_through *fall_through_list*
 Specifies the same as the **-through** option, except that the path must fall through the objects specified.

-delay_type *delay_type*
 Specifies the type of path delay. Valid values are **max** (the default), **min**, **min_max**, **max_rise**, **max_fall**, **min_rise**, or **min_fall**. The "rise" or "fall" in the *delay_type* refers to a rising or falling transition at the path endpoint.

-nworst *paths_per_endpoint*
 Gets *n* worst paths to endpoint, where *paths_per_endpoint* is ≥ 1 . The default is 1, meaning that only the worst path to an endpoint is considered. Specifying larger values of *paths_per_endpoint* increases run time.

-max_paths *max_path_count*
 Specifies the maximum number of paths to get per path group, where *max_path_count* ≥ 1 . The default is 1.

-path_type *format*

Specifies the format of the path report and how the timing path is displayed. The allowed value is **full_clock_expanded**, which displays full clock paths between a primary clock and a related generated clock in addition to the **full_clock** timing path.

-group *group_name*

Restricts the collection to paths in the groups specified in *group_name*. Paths are grouped by using the **group_path** or **create_clock** command.

-true

Specifies that the longest (least-slack) true paths in the design are to be reported. This option can require long runtimes for certain designs that have many false paths. The **true_delay_prove_true_backtrack_limit** and **true_delay_prove_false_backtrack_limit** variables are used to limit the amount of backtracking during the operation of the **report_timing** command with the **-true** option. The **set_case_analysis** command is used to specify a partial input vector to be considered for **-true** analysis. **-true** cannot be combined with **-max_paths(>1)**, **-nworst(>1)**, **-delay_type** (path type other than **max**), **-unique**, **-rise_through**, **fall_through** and **-rise_from** and **fall_from** options: **-true** is mutually exclusive with them.

-unique_pins

Specifies that only paths through a unique set of pins are to be reported. This option can require longer runtimes when used in combination with the **-nworst** option with a large number of paths targeted for reporting.

-start_end_pair

Indicates that paths are reported for each pair of startpoint and endpoint based on connectivity. This option can lead to long runtime with large memory usage and can lead to generating a huge number of paths depending on the design. By default this option will only search for paths which are violating. This default value can be changed by having an explicit **-slack_lesser_than** option. The options that do not work with this option are **-nworst**, **-max_paths**, **-unique_pins**, **-true**, **-justify**, **-slack_greater_than**, **-ignore_register_feedback**. Unlike with other options of **get_timing_paths**, this option causes the paths returned to no longer be sorted based on slack, instead, paths are arranged based on the endpoint with those sharing the same endpoint appearing next to one another. The maximum number of paths returned is limited to 2000000. In order to avoid the potential of returning duplicate paths, this option works as though the variable **timing_report_always_use_valid_start_end_points** was set to true.

-true_threshold *path_delay*

Used with the **-true** option. Specifies a threshold path delay value, in library time units, to be used by the **-true** option to speed up searching. If this option is specified, the **get_timing_paths** command with the **-true** option returns the first path it finds greater than or equal to *path_delay*, rather than continuing to search for a longer one.

-slack_greater_than *greater_slack_limit*

Specifies that only those paths with a slack greater than *greater_slack_limit* are to be reported. This option can be combined with **-slack_lesser_than** to report only those paths inside or outside a given slack range.

`-slack_lesser_than lesser_slack_limit`

Specifies that only those paths with a slack less than *lesser_slack_limit* are to be reported. This option can be combined with **-slack_greater_than** to report only those paths inside or outside a given slack range.

`-ignore_register_feedback feedback_slack_cutoff`

Specifies that timing paths are to be ignored if they start and end at the same register that holds a value. This option applies to min delay as well as max delay reports. Paths are ignored only if the slack is less than the specified *feedback_slack_cutoff* value. This option is applied as a filter to the paths after they are generated. Therefore, the number of paths generated may be less than the number specified with the `-nworst` and `-max_paths` options.

`-include_hierarchical_pins`

Specifies that the returned timing paths contain points for each hierarchical pin crossed, as well as all leaf pins in the path.

`-justify`

Specifies to find and report an input vector that sensitizes the reported paths or to report the path as false if no input vector is found. Use the **set_case_analysis** command to specify a partial input vector to be considered for **-justify** analysis.

`-trace_latch_borrow`

This option controls the type of report generated for a path that starts at a transparent latch. If the path startpoint borrows from the previous stage, using this option causes the report to show the entire set of borrowing paths that lead up to the borrowing latch, starting with a nonborrowing path or a noninverting sequential loop.

`-pre_commands pre_command_string`

This option is available only if the user invokes PrimeTime with the `-multi_scenario` option. This option allows users to specify a string of commands to be executed in the slave context before the execution of the `merged_reporting` command. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

`-post_commands post_command_string`

This option is available only if the user invokes PrimeTime with the `-multi_scenario` option. This option allows users to specify a string of commands to be executed in the slave context after the execution of the `merged_reporting` commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

`-dont_merge_duplicates`

This option is available only if the user invokes PrimeTime with the `-multi_scenario` option. It turns OFF a main capability in merged reporting that is ON by default. The option affects the manner in which paths from multiple scenarios are merged. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. By using this option, PrimeTime will not merge duplicate instances of the same path into a single instance, but instead shows all critical instances of the path from all scenarios. Since the number of paths reported is limited by the `-nworst`, `-max_paths` and other options of this command, the resulting

merged report, when this option is used, may not be evenly spread out across the design, but instead may be focussed on the portion of the design that is critical in each scenario.

-attributes *attribute_list*

This option is available only if the user invokes PrimeTime with the **-multi_scenario** option. A list of attributes to be retrieved from a slave collection. If this option is not specified then only the **full_name**, **scenario_name** and **object_class** attributes are retrieved. This option should be used in conjunction with **set_distributed_parameter -collection_levels** commands to control the amount of data retrieved from the slave.

-pba_mode *effort*

This option controls path-based analysis. The *effort* value can be one of {*none*, *path*, *exhaustive*}. When *effort* is set to *none* (the default) PBA is not applied. When *effort* is set to *path*, PBA is applied to paths after they have been gathered. When *effort* is set to *exhaustive* an exhaustive PBA path search algorithm is applied to determine the worst PBA path set in the design. This option cannot be specified with **-true** when *effort* is set to *path*. This option cannot be specified with **-justify**, **-start_end_pair** or *path_collection* when *effort* is set to *exhaustive*.

path_collection

This option can only be specified with "**-pba_mode path**". Path-based analysis is performed on the paths in the *path_collection* and a new path collection is returned. This option is mutually exclusive with options that control the selection of paths to gather.

DESCRIPTION

The **get_timing_paths** command creates a collection of paths for custom reporting or other operations. Use the **foreach_in_collection** command to iterate among the paths in the collection. You can use the **get_attribute** and **collection** commands to obtain information about paths. The following attributes are supported on timing paths:

clock_uncertainty
endpoint
endpoint_clock
endpoint_clock_close_edge_type
endpoint_clock_close_edge_value
endpoint_clock_is_inverted
endpoint_clock_is_propagated
endpoint_clock_latency
endpoint_clock_open_edge_type
endpoint_clock_open_edge_value
endpoint_clock_pin
endpoint_hold_time_value
endpoint_is_level_sensitive
endpoint_output_delay_value
endpoint_recovery_time_value
endpoint_removal_time_value
endpoint_setup_time_value
object_class

get_timing_paths

```

path_group
path_type
points
slack
startpoint
startpoint_clock
startpoint_clock_is_inverted
startpoint_clock_is_propagated
startpoint_clock_latency
startpoint_clock_open_edge_type
startpoint_clock_open_edge_value
startpoint_input_delay_value
startpoint_is_level_sensitive
time_borrowed_from_endpoint
time_lent_to_startpoint

```

One attribute of a timing path is the *points* collection. A point corresponds to a pin or port along the path. Iterate through these points with **foreach_in_collection** and get attributes on them by using the **get_attribute** command. The following attributes are available for points of a timing path:

```

arrival
object
object_class
rise_fall
slack
transition
voltage

```

For information about creating collections and iterating over the elements of a collection, see the **collections** and **foreach_in_collection** man pages.

EXAMPLES

You can find some detailed examples of custom timing reports in:

```
$SYNOPSYS_ROOT/auxx/pt/examples/tcl
```

where \$SYNOPSYS_ROOT is the installation directory for PrimeTime.

The following procedure prints out the startpoint name, endpoint name, and the slack of the worst path in each path group.

```

proc custom_report_worst_path_per_group {} {
    echo [format "%-20s %-20s %7s" "From" "To" "Slack"]
    echo "-----"
    foreach_in_collection path [get_timing_paths] {
    set slack [get_attribute $path slack]
    set startpoint [get_attribute $path startpoint]
    set endpoint [get_attribute $path endpoint]
    echo [format "%-20s %-20s %s" [get_attribute $startpoint full_name] \
        [get_attribute $endpoint full_name] $slack]
    }
}

```

```
pt_shell> custom_report_worst_path_per_group
```

>From	To	Slack
ffa/CP	QA	0.1977
ffb/CP	ffd/D	3.8834

The following example shows Total Negative Slack, Total Positive Slack, and Worst Negative Slack for the current design.

```
proc report_design_slack_information {} {
  set design_tns 0
  set design_wns 100000
  set design_tps 0
  foreach_in_collection group [get_path_groups *] {
    set group_tns 0
    set group_wns 100000
    set group_tps 0
    foreach_in_collection path [get_timing_paths -nworst 10000 -group $group] {
      set slack [get_attribute $path slack]
      if {$slack < $group_wns} {
set group_wns $slack
if {$slack < $design_wns} {
  set design_wns $slack
}
      }
      if {$slack < 0.0} {
set group_tns [expr $group_tns + $slack]
      } else {
set group_tps [expr $group_tps + $slack]
      }
    }
    set design_tns [expr $design_tns + $group_tns]
    set design_tps [expr $design_tps + $group_tps]
    set group_name [get_attribute $group full_name]
    echo [format "Group '%s' Worst Negative Slack : %g" $group_name $group_wns]
    echo [format "Group '%s' Total Negative Slack : %g" $group_name $group_tns]
    echo [format "Group '%s' Total Positive Slack : %g" $group_name $group_tps]
    echo ""
  }
  echo "-----"
  echo [format "Design Worst Negative Slack : %g" $design_wns]
  echo [format "Design Total Negative Slack : %g" $design_tns]
  echo [format "Design Total Positive Slack : %g" $design_tps]
}
```

```
pt_shell> report_design_slack_information
```

```
Group 'CLK' Worst Negative Slack : -3.1166
Group 'CLK' Total Negative Slack : -232.986
Group 'CLK' Total Positive Slack : 4.5656

Group 'vclk' Worst Negative Slack : -4.0213
Group 'vclk' Total Negative Slack : -46.1982
```

Group 'vclk' Total Positive Slack : 0

Design Worst Negative Slack : -4.0213
Design Total Negative Slack : -279.184
Design Total Positive Slack : 4.5656

If the **-pba_mode** option is specified, path recalculation (path-based analysis) is used during the path search, and the worst recalculated paths meeting the user's criteria are returned. This option requires that a path search be performed to ensure that the paths being returned truly are the worst paths. The additional runtime required for this option depends on the amount of timing improvement resulting from path-based analysis. As the timing improvement from path-based analysis increases, the runtime for the path search will increase. Large **-nworst** values can significantly increase the time needed for the search.

To see the worst path in the design with path-based analysis applied:

```
pt_shell>report_timing [get_timing_paths -pba_mode exhaustive]
```

To report all endpoints in the design which fail after considering path-based analysis:

```
pt_shell> set paths [get_timing_paths -pba_mode exhaustive -slack_lesser_than 0 -  
max_paths 1000]  
pt_shell> report_timing $paths
```

SEE ALSO

collections (2), **create_clock** (2), **foreach_in_collection** (2), **get_attribute** (2), **group_path** (2), **read_sdf** (2), **report_timing** (2), **set_case_analysis**, **set_operating_condition** (2), **true_delay_prove_false_backtrack_limit** (3), **true_delay_prove_true_backtrack_limit** (3), **timing_report_always_use_valid_start_end_points** (3).

get_variation_attribute

Returns a collection of one or more values associated with a variation's attribute.

SYNTAX

```
list get_variation_attribute
[-class class_name]
[-quiet]
variation
attribute
value
collection variation
string attribute
int or float value
```

ARGUMENTS

variation
The variation collection.

attribute
The attribute of the variation. Valid attributes are *pdf*, *cdf*, *quantile*, *cdf_values*, and *pdf_values*.

value
The parameterized value associated with the attribute of the variation.

DESCRIPTION

Returns the value (or a list of values) associated with the variation's attribute, parameterized at value.

EXAMPLES

This example returns the 0.98-quantile of the variation \$arrival_var.

```
pt_shell> get_variation_attribute $arrival_var quantile 0.98
30.000000
```

This example returns the cdf value at x-value 4.6.

```
pt_shell> get_variation_attribute $arrival_var cdf 4.6
0.000000
```

This example returns the pdf value at x-value 1.0.

```
pt_shell> get_variation_attribute $arrival_var pdf 1.0
```

```
get_variation_attribute
312
```


0.000000

This example returns a list of 100 x-pdf value pairs in the format of x1 pdf1 x2 pdf2 ... x100 pdf100.

```
pt_shell> get_variation_attribute $arrival_var pdf_values 100  
{-3.500000 0.000873} {-2.722222 0.009812} ...
```

This example returns a list of 100 x-cdf value pairs in the format of x1 cdf1 x2 cdf2 ... x100 cdf100.

```
pt_shell> get_variation_attribute $arrival_var cdf_values 100  
{-3.500000 0.000234} {-2.722222 0.003250} ...
```

SEE ALSO

`create_variation(2)`.

get_variations

Creates a collection of variations from the current design. You can assign these variations to a variable or pass them into another command.

SYNTAX

```
collection get_variations [-quiet] [-regex] [-nocase] [-filter expression] patterns  
string expression  
list patterns
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regex

Views the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with **-regex**, makes matches case-insensitive. You can use **-nocase** only when you also use **-regex**.

-filter *expression*

Filters the collection with *expression*. For any variations that match *patterns*, the expression is evaluated based on the variation's attributes. If the expression evaluates to true, the variation is included in the result.

patterns

Matches variation names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_variations** command creates a collection of variations in the current design that match certain criteria. The command returns a collection if any variations match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_variations** command at the command prompt, or you can nest it as an argument to another command (for example, **query_objects**). In addition, you can assign the **get_variations** result to a variable.

When issued from the command prompt, **get_variations** behaves as though **query_objects** had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the variable **collection_result_display_limit**.

The "implicit query" property of **get_variations** provides a fast, simple way to display variations in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use **get_variations** as an argument to **query_objects**.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man page for **all_variations**, which also creates a collection of variations.

EXAMPLES

The following example sets the variable `variation_list` to a collection of variations that have a mean of 15.

```
pt_shell> set variation_list [get_variations * -filter "mean==15"]
```

SEE ALSO

all_variations(2), **collections(2)**, **create_variation(2)**, **query_objects(2)**, **report_variation(2)**; **collection_result_display_limit(3)**.

group_path

Groups paths for cost function calculations and reporting.

SYNTAX

```
Boolean group_path
-name group_name | -default
[-weight weight_value]
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
```

```
string group_name
float weight_value
float range_value
list   from_list
list   rise_from_list
list   fall_from_list
list   through_list
list   rise_through_list
list   fall_through_list
list   to_list
list   rise_to_list
list   fall_to_list
```

ARGUMENTS

- name *group_name***
Specifies a name for the group. If a group with this name already exists, PrimeTime adds the paths or endpoints to that group. If a group with this name does not exist, PrimeTime creates a new group. You must specify **-name** unless you use **-default**; the **-name** and **-default** options are mutually exclusive.
- default**
Specifies that endpoints or paths are moved to the default group and removed from the current group. You must specify **-default** unless you use **-name**; the **-default** and **-name** options are mutually exclusive.
- weight *weight_value***
Specifies a cost function weight for this group. The *weight_value* must be a number between 0.0 and 100.0, the default is 1.0. A weight of 0.0 eliminates the paths in this group from cost function calculations. Do not use very small values (for example, 0.0001). Smaller values can prevent PrimeTime from implementing small improvements to the design. If you specify **-weight** when you add members to an existing group, the new weight for the group is used.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, one path endpoint on that cell is affected.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-through *through_list*

Specifies a list of path throughpoints (ports, pins, cells, or nets). You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

`-from A1 -through B1 -through C1 -to D1`

If more than one object is specified within one *-through*, *-rise_through*, or *-fall_through* option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

`-from A1 -through {B1 B2} -through {C1 C2} -to D1`

`-rise_through rise_through_list`

It is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects.

`-fall_through fall_through_list`

It is similar to the **-through** option, but applies only to paths with a falling transition at the specified objects.

DESCRIPTION

Groups a set of paths or endpoints for cost function calculations in optimization and analysis. Design Compiler uses the cost function to direct optimization. Path groups also affect the output of **report_timing** and **report_constraint**.

The delay cost function is the sum of all groups (weight * violation), where violation is the cost of the worst path in the path group. If no violation occurs within a group, the group cost is zero. Groups enable you to specify a set of paths to optimize even though there might be larger violations in another group. When you specify endpoints, all paths leading to those endpoints are grouped. However, for clock gating checks and asynchronous preset/clear checks PrimeTime creates default groups named *****clock_gating_default***** and *****async_default***** respectively. Any paths belonging to these groups cannot be overridden by a **group_path** command.

If you specify a clock in **object_list**, all endpoints related to that clock are included in the group. The **create_clock** command automatically creates a group for a new clock with a weight of 1.0 and the same name as the clock name.

The weight of the default group is 1.0. If you do not specify *weight_value* for a new group, the default is 1.0.

To undo **group_path**, use **remove_path_group**. To report path group information for a design, use **report_path_group**.

EXAMPLES

The following example groups all endpoints clocked by CLK1A or CLK1B into a new group 'group1' with weight = 2.0.

```
pt_shell> group_path -name group1 \  
-weight 2.0 -to {CLK1A CLK1B}
```

The following example adds OUT1 and ff34/D to the existing path group named 'ADDR'.

```
pt_shell> group_path -name ADDR \  
-to {OUT1 ff34/D}
```

The following example removes OUT1 and CLK2 from existing groups and places them in the default group.

```
pt_shell> group_path -default \  
-to {OUT1 CLK2}
```

The following example groups all paths from inputs I1 and I2 to outputs O5 and O7 into a group named 'serious' with weight 10.0.

```
pt_shell> group_path -name serious \  
-weight 10.0 -from {I1 I2} -to {O5 O7}
```

SEE ALSO

```
create_clock (2), current_design (2), remove_path_group (3), report_constraint (2),  
report_path_group (2), reset_design (2), set_input_delay (2), set_output_delay (2).
```

gui_start

Starts the PrimeTime GUI.

SYNTAX

```
string gui_start  
    [-file name_of_script_file]  
    [-no_windows]
```

Data Types

name_of_script_file string

ARGUMENTS

-file *name_of_script_file*
 The given script file is sourced before the GUI starts.

-no_windows
 The GUI starts without showing the default window.

DESCRIPTION

This command starts the PrimeTime GUI from the pt_shell prompt. It is ignored if the PrimeTime GUI has already been started.

EXAMPLES

The following example starts the PrimeTime GUI.

```
pt_shell> gui_start
```

SEE ALSO

gui_stop(2)

gui_stop

Stops the PrimeTime GUI.

SYNTAX

string **gui_stop**

ARGUMENTS

None.

DESCRIPTION

This command stops the PrimeTime GUI and returns to the pt_shell prompt. It is ignored if the PrimeTime GUI has not been started or has been stopped.

EXAMPLES

The following example stops the PrimeTime GUI and returns to the pt_shell prompt.

```
pt_shell> gui_stop
```

SEE ALSO

gui_start(2)

gui_write_window_image

Write a specified window to an image file.

SYNTAX

```
status gui_write_window_image
{ -file file_name }
[ -format png | xpm | jpg | bmp ]
[ -window window_id ]
```

Data Types

```
file_name      string
window_id     string
```

ARGUMENTS

-file *file_name*
Specifies the name of the file where the image will be saved.
The file name's postfix will determine the image format, unless *-format* has been used.

-format png | xpm | jpg | bmp
Specifies the image format to be written to the file.
Default value is png.
Note: If the format value contradicts the *file_name* postfix the format will take precedence and be appended to *file_name*.

-window *window_id*
Specifies the window name to write to the image file.
Open window ids can be queried using **gui_get_window_ids** command.
Default value is most recently used window.

DESCRIPTION

The **gui_write_window_image** command writes to a specified file an image of a window.

EXAMPLES

The following example will create a toplevel window and a child schematic view window, then will write out the schematic window's image to specified file *my_image.xpm*.

```
shell> set top [gui_create_window -type TopLevel]
shell> set schm [gui_create_schematic]
shell> gui_write_window_image -file my_image.xpm -window $schm
```

SEE ALSO

gui_create_schematic(2)

gui_write_window_image
322

```
gui_create_window(2)
gui_get_window_ids(2)
gui_set_active_window(2)
gui_get_current_window(2)
```

identify_interface_logic

Sets the **is_interface_logic_pin** attribute on pins of the current design that are part of its interface logic.

SYNTAX

```
int identify_interface_logic  
[-ignore_ports port_list | -auto_ignore]  
[-latch_level levels | -context_borrow]  
[-keep_ignored_fanout]  
[-include_pins pin_list]  
[-critical_pins]  
[-include_all_net_pins]  
[-traverse_disabled_arcs]
```

```
string port_list  
int level
```

ARGUMENTS

-ignore_ports *port_list*

Specifies a list of input or output ports whose fanout or fanin is to be ignored when placing the **is_interface_logic_pin** attribute. Substitute the list of ports you want for *port_list*. Clock ports are automatically ignored; you do not have to specify them in this list.

You can use this option to exclude the fanout of ports connected to chip-level nets (for example, scan enable and reset) from impacting the contents of interface logic. If these nets are not explicitly ignored, they cause unnecessary logic (for example, internal registers) to be part of the interface logic on the current design. You can also use this option to selectively generate the interface logic for a subset of the ports on a block; for example, an interface logic model (ILM) can model only the timing behavior on a subset of the ports on a block.

By default all nonclock input and all output ports are used in identifying the contents of interface logic. The **-ignore_ports** and **-auto_ignore** options are mutually exclusive.

-auto_ignore

Enables automatic determination of ports whose fanout should be ignored when placing the **is_interface_logic_pin** attribute. A port is ignored if the percentage of the total registers in the design in the transitive fanout of the port exceeds a specified threshold percentage contained in the **ilm_ignore_percentage** variable (default 25).

You can use this option to help you identify the test enable and reset ports of your design. Before using it, examine the current value of the **ilm_ignore_percentage** variable and reset it if necessary. Note that the **-auto_ignore** option might potentially ignore ports you do not want to ignore, or fail to ignore ports you want to ignore. Carefully read the messages issued by this command when you use this option to see which ports have been ignored and what percentage of registers to which they fanned out.

The **-ignore_ports** and **-auto_ignore** options are mutually exclusive.

-latch_level *levels*

Specifies the number of logic levels over which time borrowing can occur for latch chains that are a part of the interface logic. Substitute the number of levels you want for *levels*. By default, all latches found in interface logic are assumed to be potential borrowers. Thus, all logic from I/O ports to flip-flops or output ports are identified as belonging to interface logic. Note that this is the opposite of the default behavior of the **extract_model** command.

When you use this option, note that the value of *level* must account for the borrowing behavior of latches only on interface timing paths. If n represents a latch level, then $n + 1$ represents the number of latches maintained in latch chains that originate at input ports. The $n + 1$ th latch functions as an edge-triggered register and decouples I/O paths from internal paths. Use this option only when there are latches in the interface logic for a block. Specifying this option might potentially result in less accurate models, because not all latches are allowed to borrow. The **-context_borrow** and **-latch_level** options are mutually exclusive.

-context_borrow

Specifies that latch borrowing at the interface should be established based on the current context defined for the design. So, latches that borrow based on arrival times defined on input ports and clocks defined on the design are traced through, but path tracing stops at nonborrowing latches. The **-context_borrow** and **-latch_level** options are mutually exclusive.

-keep_ignored_fanout

Specifies that the fanout from ignored input ports to interface logic is to be maintained in the model. Thus, ignored ports are not used to identify interface logic, but connections between ignored ports and interface logic are preserved. Using this option results in larger models, particularly in conjunction with the **write_ilm_netlist** command **-include_all_net_pins** option. Timing slacks for ignored ports might differ from those reported on the original netlist because connections from ignored ports to internal registers are not preserved in the model.

-include_pins *pin_list*

Specifies a list of pins that must be included in the interface logic. Substitute the list of pins you want for *pin_list*. This option can be used to optionally add internal pins to the interface logic. After the interface logic is determined, this list of pins is added to the interface logic. You can use this option to define additional interface logic pins that will otherwise be removed in the model. There will not be any affect on pins that are already in the interface logic. Use this option in conjunction with the **all_fanin** and **all_fanout** commands to include a particular cone of logic in the model.

-critical_pins

Specifies that critical pins interface logic must be identified. This option allows you to extract a model that keeps only the pins in the critical paths of the design.

You can use this option to extract a critical pins interface logic. The model generated contains the interface logic pins in the critical paths of the design. The model is compact compared to normal interface logic models, but might not be as accurate outside of the current context. This model can be used only to do critical path analysis. It might take much longer to generate

this model than the normal model.

-include_all_net_pins

Specifies that all pins on non-clock interface logic nets and propagated clock interface logic nets are to be included in the netlist. Use this option if the interface logic model (ILM) will be used in non-SDF flows and delay calculation will be performed using the information contained in the ILM. Preserving all pins on a net maintains correct pin capacitance information for the net.

The **-include_all_net_pins** option does not affect non-propagated clock nets; that is, nets in the clock network that have user-specified source latency. This option is similar to the same option of the **write_ilm_netlist** command except that here it adds the extra pins to the interface logic. So, if you query the interface logic pins by using the **get_ilm_objects** command), you will notice these extra pins have been added.

The **write_ilm_netlist** command writes these pins only to the verilog netlist, but does not add them to the interface logic. It is recommended that you use this option while extracting a critical pins interface logic model.

-traverse_disabled_arcs

Specifies that the interface logic should not be affected by disabled arcs/pins on the design. If specified, this option forces the traversal of disabled arcs while determining interface logic.

DESCRIPTION

Sets the **is_interface_logic_pin** attribute on pins of the current design that are part of its interface logic. This is the first step in the generation of an interface logic model (ILM).

The interface logic on a block contains all cells whose timing is impacted by, or impacts, the external environment of a block. The following list describes such parts of interface logic:

- All cells in timing paths that lead from input ports to registers or output ports that terminate the paths.
- All cells in timing paths that lead to output ports from registers or input ports that originate the paths.
- Clock trees that drive interface registers; including any registers in the clock tree. Clock-gating circuitry is part of interface logic if it is driven by external ports, but not if it is driven by registered outputs on a block.

Notice that interface logic does not include internal register-to-register paths and logic on a block associated only with these paths.

This command implicitly performs an **update_timing** on the design if required.

You can review the objects you have identified as interface logic by using the **get_ilm_objects** command. When you are satisfied with the interface logic

designations, you can nondestructively generate a flattened Verilog netlist for the interface logic model (ILM) by using the **write_ilm_netlist** command. To generate script and back-annotation files for the ILM, use the **write_ilm_script**, **write_ilm_parasitics**, and **write_ilm_sdf** commands.

You can reset the `is_interface_logic_pin` attribute by reexecuting the **identify_interface_logic** command. You can remove all user-defined attributes, including the `is_interface_logic` attribute by using the **reset_design** command

EXAMPLES

The following example places the **is_interface_logic_pin** attribute on all nonclock pins in the current design, except for pins in the fanin/fanout of ports *port1*, *port2*, and *port3*.

```
pt_shell> identify_interface_logic \  
-ignore [get_ports {port1 port2 port3}]
```

The following example additionally includes three levels of logic from internal pin *ff/Q*.

```
pt_shell> set ilm_pins [all_fanout -level 3 \  
-flat [get_pin ff/Q]]  
pt_shell> identify_interface_logic \  
-include_pins $ilm_pins
```

The following example extracts a critical pins interface logic model.

```
pt_shell> identify_interface_logic -critical_pins \  
-include_all_net_pins
```

The following example places the `is_interface_logic_pin` attribute on all nonclock pins in the current design, except for pins identified by the **-auto_ignore** option. Because the value of the **ilm_ignore_percentage** variable is currently 35, pins are ignored if the percentage of the total design registers in their transitive fanout is greater than 35. The **-latch_level** option with a value of 1 states that the first latch encountered in IO paths might potentially borrow, but the second latch can be treated as an edge-triggered device. Thus, two levels of latches are maintained in the interface logic.

```
pt_shell> printvar ilm_ignore_percentage  
ilm_ignore_percentage = "35"  
pt_shell> identify_interface_logic -auto_ignore \  
-latch_level 1
```

SEE ALSO

get_ilm_objects (2), **write_ilm_netlist** (2), **write_ilm_parasitics** (2), **write_ilm_script** (2), **write_ilm_sdf** (2); **ilm_ignore_percentage** (3).

index_collection

Creates a single element collection. I.e. Given a collection and an index into it, if the index is in range, extracts the object at that index and creates a new collection containing only that object. The base collection remains unchanged.

SYNTAX

```
collection index_collection collection1 index  
collection collection1  
int index
```

ARGUMENTS

collection1
Specifies the collection to be searched.

index
Specifies the index into the collection. Allowed values are integers from 0 to **sizeof_collection** - 1.

DESCRIPTION

You can use the **index_collection** command to extract a single object from a collection. The result is a new collection containing that object. The range of indices is from 0 to one less than the size of the collection. If the specified index is outside that range, an error message is generated. Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support the sorting of collections allow you to impose a specific order on a collection. You can use the empty string for the *collection* argument. However, by definition, any index into the empty collection is invalid. So using **index_collection** with the empty collection always generates the empty collection as a result and generates an error message. Note that not all collections can be indexed.

EXAMPLES

The following example shows the first object in a collection being extracted.

```
pt_shell> set c1 [get_cells {u1 u2}]  
{"u1", "u2"}  
pt_shell> query_objects [index_collection $c1 0]  
{"u1"}
```

SEE ALSO

collections (2), **query_objects** (2), **sizeof_collection** (2).

insert_buffer

Inserts a buffer at one or more pins.

SYNTAX

```
string insert_buffer [-libraries lib_spec] [-inverter_pair] [-new_net_names  
new_net_names] [-new_cell_names new_cell_names] pin_or_port_list lib_cell
```

```
list new_net_names  
list new_cell_names  
list pin_or_port_list  
string lib_cell
```

ARGUMENTS

-libraries *lib_spec*

If this option is specified, then PrimeTime resolves *lib_cell* from the libraries contained in the *lib_spec* only. Libraries are searched in the order in which they appear in *lib_spec*. *lib_spec* can be a list of library names, or collections of libraries loaded into PrimeTime; the latter can be obtained using the **get_libs** command. You cannot specify this option if a full library cell name has been specified.

-inverter_pair

Indicates that a pair of inverting library cells is to be inserted instead of a single non-inverting library cell.

-new_net_names *new_net_names*

Specifies the net name to be given to the new net that PrimeTime inserts. This option can only be used if only one buffer or an inverter pair is being inserted. If one buffer is being inserted, you have to pass only one net name. If an inverter pair is being inserted, you have to pass two net names. These names can be any valid net names, but must be the leaf names i.e. not the hierarchical names. The new names must not contain embedded hierarchical separators. The new names must be unique in the current context (as specified by *current_instance*). If you use this option, you have to also use the **-new_cell_names** option.

-new_cell_names *new_cell_names*

Specifies the cell name to be given to the new cell that PrimeTime inserts. This option can only be used if only one buffer or an inverter pair is being inserted. If one buffer is being inserted, you have to pass only one cell name. If an inverter pair is being inserted, you have to pass two cell names. These names can be any valid cell names, but must be the leaf names i.e. not the hierarchical names. The new names must not contain embedded hierarchical separators. The new names must be unique in the current context (as specified by *current_instance*). If you use this option, you have to also use the **-new_net_names** option.

pin_or_port_list

Specifies a list of pins or ports to buffer.

`lib_cell`

Specifies the name of the library cell to use as a buffer (or inverter if **-inverter_pair** is specified). `lib_cell` can be a library cell object, or the name of a library cell. The former can be obtained using the **get_lib_cells** command. The latter can either be the full library cell name, i.e. '`lib_name/lcell_name`', or the just the base name of the library cell, i.e. '`lcell_name`'. You cannot specify the **-libraries** option and explicitly specify the full library cell name. If the user invokes PrimeTime with the **-multi_scenario** option, then the library cell base name only must be used. For more information, see the section entitled "RESOLVING LIBRARY CELLS".

DESCRIPTION

The **insert_buffer** command adds a buffer at one or more specified pins or ports. Like all other netlist editing commands, for **insert_buffer** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. The result of **insert_buffer** is a collection of the newly inserted cells on success, or the empty collection (empty string) on failure.

A library cell is a buffer if it has a single input and any number of outputs, as long as for each output, the logic function is either input or !input.

Newly created cells are given a unique name beginning with "U" and ending with a number; newly created nets are given a unique name beginning with "net" and ending with a number.

If you do not want PrimeTime to generate automatic names, you can use **-new_net_names** and **-new_cell_names** options to override.

insert_buffer uses the following basic rules to check its arguments:

- Each pin/port must be in scope (at or below the current instance). For a description of some special scoping rules, see the section entitled "BUFFERING INSIDE BOUNDARY PINS".
- Each pin/port must be connected to a net.
- Bidirectional pins cannot be buffered.
- The `lib_cell` cannot be sequential.
- The `lib_cell` must be a buffer, as previously defined.
- Without **-inverter_pair**, the library cell must have a noninverting output. The first noninverting output is used.
- When **-inverter_pair** is specified, the library cell must have an inverting output.

`insert_buffer`

The first inverting output is used. In this case, two cells are inserted, preserving the logic of the path.

- This command is a rare case where the objects argument (in this case, *pin_or_port_list*) does not necessarily stand alone. Pins on the same net can be grouped.

For details about grouping pins on the same net, and for additional connection rules, see the section entitled "CONNECTING THE NEW BUFFER".

Although you can mimic buffer insertion using other commands (for example, **create_cell**, **create_net**, **disconnect_net**, and **connect_net**), it is much more efficient to use **insert_buffer**.

Connecting the New Buffer

The new buffer is connected according to the following rules:

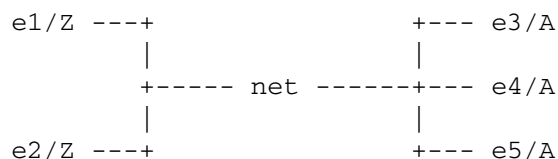
- The output of the new cell is always the new net, and the input of the new cell is always the old net. For example, buffering e1/Z or e3/A as follows

```
e1/Z --- old_net --- e3/A
```

creates the following:

```
e1/Z --- old_net --- U1 --- net1 --- e3/A
```

- If the pin is a load, it is disconnected from its net, and connected to the new net.
- If the pin is a driver, all loads on that net are disconnected from the old net and connected to the new net.
- When pins on the same net are specified, they are grouped, and one buffer is inserted. Loads and drivers are grouped separately.
- If the net is multi-driven, all driving pins must be specified for the command to succeed. In the following example, you cannot specify only e1/Z; you must specify both e1/Z and e2/Z. However, for the loads in Circuit 1, any combination can be buffered.



Circuit 1

- Certain parallel buffer cases are also examined more closely. In Circuit 2, you cannot subdivide the inputs of the parallel drivers. You must buffer the inputs of both cell1 and cell2.

+--- cell1 ---+ net1 ---+ +--- net2 --- +--- cell2 ---+

Circuit 2 -----

BUFFERING INSIDE BOUNDARY PINS

When you specify insertion of a buffer at a pin on the boundary of a hierarchical block, **insert_buffer** inserts the buffer either inside or outside the hierarchical block, depending on the current hierarchical scope. To insert the buffer inside the hierarchical block, set the scope to that block using the **current_instance** command, then execute **insert_buffer**. The buffer is inserted within the block to which **current_instance** is set. For an example of buffering inside boundary pins, see the EXAMPLES section.

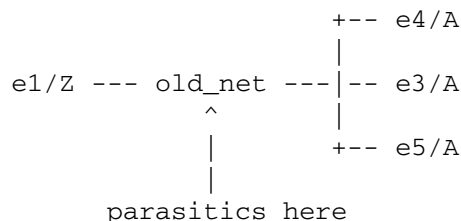
Buffering and its Effects on Parasitics

When parasitics are present and buffer insertion is performed the parasitics will only be preserved under the following conditions. (Note: what applies for a single buffer also applies for an inverter pair)

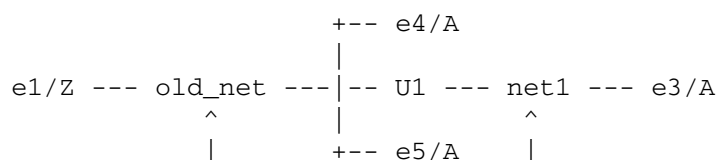
- Buffering load pins in the presence of parasitics.

When a single load pin connected to a net with parasitics is buffered, the parasitics will be preserved on the net connected to the input of the inserted buffer. There will be no parasitics on the net connected to the output of the inserted buffer.

Buffer load pin e3/A as follows



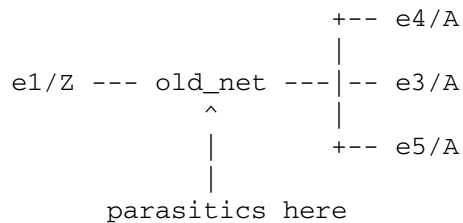
results in the parasitics being kept on the old net



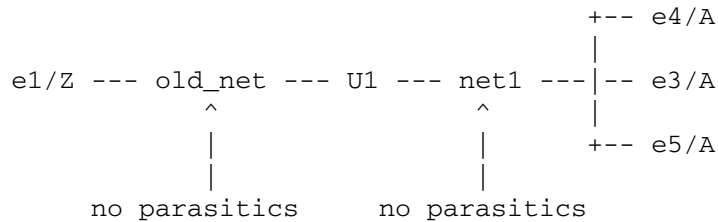
|
parasitics here
|
no parasitics

Buffering more than a single load pin will result in the parasitics on the net connected to the input of the inserted buffer being removed and a warning being issued. (See PARA-060)

Buffer load pins e3/A, e4/A and e5/A



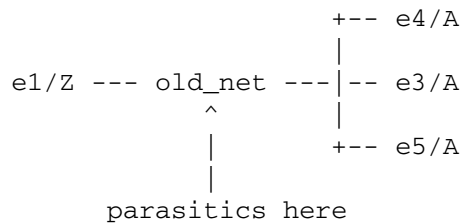
results in the parasitics being removed



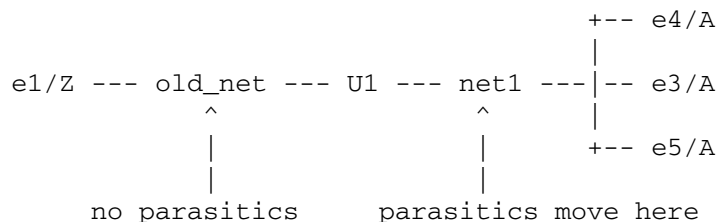
- Buffering driver pins in the presence of parasitics.

When a single driver pin is buffered, on a net with only one driver, the parasitics will be moved to the net connected to the output of the inserted buffer. There will be no parasitics on the net connected to the input of the inserted buffer.

Buffer driver pin e1/Z as follows

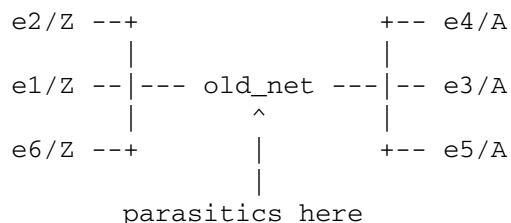


results in the parasitics being moved to net1

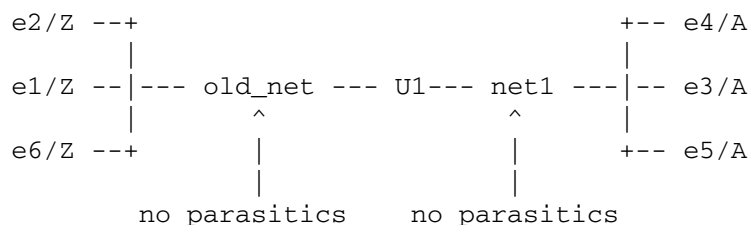


Buffering more than a single driver pin on a multiply driven net will result in the parasitics being removed from the net connected to the output of the inserted buffer. (See PARA-060)

Buffer driver pin e1/Z, e2/Z and e6/Z as follows



results in the parasitics being removed



Resolving Library Cells

If the `lib_cell` has been specified in base name only format, i.e. without a library from which to resolve it from, then PrimeTime resolves the library cell according to the following methodology.

If the **-libraries** option is specified, then PrimeTime searches for library cells in the libraries contained within the *lib_spec* only.

Alternatively, if the above option is not specified, PrimeTime searches for the library cell in the libraries contained within the **link path** variable.

The first library cell that is found is used.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with `swap_cell`. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

Refer to Circuit 1 for the following example; e1/Z and e2/Z both drive *net*. The first command specifies a *lib_cell* that is not a buffer; the command fails, and an error message is generated. The second command specifies a buffer for a *lib_cell*, but does not specify all driver pins on a multi-driven net. Again the command fails with an error message. The third command specifies a buffer for *lib_cell* and specifies all pins for the multi-driven net. This time the command succeeds, and creates the new cell U1 and the new net net1.

```
pt_shell> insert_buffer e1/Z class/AN2P
Error: Could not insert 'class/AN2P' -
      lib_cell is not a buffer. (NED-010)
```

Error: No changes made. (NED-040)

```
pt_shell> insert_buffer e1/Z class/B1I
```

Error: Could not insert a buffer on object 'e1/Z':

Multi-driver net and not all driver pins specified. (NED-012)

Error: No changes made. (NED-040)

```
pt_shell> insert_buffer {e1/Z e2/Z} class/B1I
```

Information: Inserted 'U1' at 'e1/Z', 'e2/Z' with 'class/B1I'. (NED-046)

```
{"U1"}
```

Refer to Circuit 1 for the following example. In the first command, an inverting buffer is inserted on two of the loads of "net". Two new cells are created, U2 and U3, as well as two new nets, net2 and net3. The second and third commands show the use of **report_cell** and **report_net**, to display the connections. Excerpts from the reports are shown.

```
pt_shell> insert_buffer {e3/A e4/A} class/IV -inverter_pair
```

Information: Inserted 'U2' and 'U3' at 'e3/A', 'e4/A' with 'class/IV'. (NED-046)

```
{"U2", "U3"}
```

```
pt_shell> report_cell -connections e4
```

```
*****
```

Report : cell

-connections

```
*****
```

Connections for cell 'e4':

Reference:	B1I
Library:	class

Input Pins	Net
-----	-----
A	net3

Output Pins	Net
-----	-----
Z	out2

1

```
pt_shell> report_net -connections net3
```

```
*****
```

Report : net

-connections

```
*****
```

Connections for net 'net3':

Driver Pins	Type
-----	-----
U3/Z	Output Pin (IV)

Load Pins	Type
-----	-----
e4/A	Input Pin (B1I)
e3/A	Input Pin (B1I)

insert_buffer

The following example demonstrates buffer insertion inside the boundary of a hierarchical block. To buffer a boundary pin on the inside of the hierarchical block, you must set the current instance to that block, as shown. Otherwise, **insert_buffer** would insert the buffer at the top level, outside instance e1.

Note the method used to get the boundary pin from within the block.

```
pt_shell> current_instance e1
u1
pt_shell> insert_buffer [get_pins ./z1] class/B1I
Uniquifying 'e1' (UBLOCK) as 'UBLOCK_0'.
Information: Inserted 'U1' at 'e1/z1' with 'class/B1I'. (NED-046)
{"e1/U1"}
```

SEE ALSO

current_instance (2), **get_pins** (2), **remove_buffer** (2), **report_cell** (2), **report_net** (2), **size_cell** (2), **swap_cell** (2), **write_changes** (2), **link_path** (3), **PARA-060**.

license_users

Lists the current users of the Synopsys licensed features.

SYNTAX

```
license_users [feature_list]  
list feature_list
```

ARGUMENTS

feature_list
Optional list of licensed features for which to obtain the information. If none is specified, then all features are shown. Refer to the *Synopsys System Installation and Configuration Guide* for a list of features supported by the current release, or determine from the key file all the features that are licensed at your site.

DESCRIPTION

Displays information about all of the licenses, related users, and host names currently in use. If a feature list is specified, only information about those features is displayed.

license_users is valid only when Network Licensing is enabled.

EXAMPLES

In this example, all of the users of the PrimeTime feature are displayed.

```
pt_shell> license_users PrimeTime  
  
john@node2      PrimeTime  
paul@node1      PrimeTime  
george@node3    PrimeTime, PrimeTime  
rstarr@node3    PrimeTime  
  
4 users listed.
```

SEE ALSO

get_license (2), **list_licenses** (2), **remove_license** (2).

link

The **link** command, a synonym for the **link_design** command, exists in PrimeTime for compatibility with Design Compiler.

SEE ALSO

link_design (2).

link_design

Resolves references in a design.

SYNTAX

```
string link_design [-verbose] [-remove_sub_designs] [-keep_sub_designs] [-force]  
[design_name]  
string design_name
```

ARGUMENTS

-verbose

Indicates that the linker is to display verbose messages.

-force

By default, if the target design is already fully linked, it is not relinked. This option forces relink of the design.

-remove_sub_designs

Indicates that subdesigns are to be removed after linking. By default, subdesigns are removed. Use this option to free up memory and improve performance. For more information, see the "Performance Considerations" section.

-keep_sub_designs

Indicates that subdesigns are to be kept after linking. By default, subdesigns are removed. Use this option to keep the sub-designs around so that `current_design` can be changed to other designs later.

design_name

Specifies the name of the design to be linked; the default is the current design.

DESCRIPTION

Performs a name-based resolution of design references for the specified *design_name* or the current design. In addition to resolving references in the design, **link_design** builds the internal representation of the design for analysis.

A complete, fully functional design must be connected to all referenced library components and designs; the references must be located and linked to the current design. Thus, the purpose of this command is to locate all designs and library components referenced by the current design and link them to the current design. During the link, all files specified in the **link_path** are loaded if they are not already in memory. The goal is a fully instantiated design on which analysis can be performed.

By default, the case sensitivity of the link is determined by the source of the objects being linked. Although it is not recommended, you can change this behavior using the **link_force_case** variable.

Automatic Loading of Designs and Libraries

You can set the **link_path** and **search_path** variables so that you need to read only your top design, then link. PrimeTime automatically finds and loads other designs and libraries that are needed.

In the following example, assume that your main design is in newcpu.db and uses the cmos.db library. The **link_path** variable specifies cmos.db, so the linker loads cmos.db when it starts. As the link proceeds, if a design is required and is not in memory, the linker searches the **search_path** for a file named *reference_name.db*. For example, the referenced BOX1 design is not in memory, so the linker searches for BOX1.db in the **search_path** and loads it.

```
pt_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"
/designs/newcpu/v1.6/dbs /libs/cmos
pt_shell> set link_path "* cmos.db"
* cmos.db
pt_shell> read_db newcpu.db
Loading db file '/designs/newcpu/v1.6/dbs/newcpu.db'
1
pt_shell> link_design newcpu
Loading db file '/libs/cmos/cmos.db'
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Design 'newcpu' was successfully linked.
1
```

Automatic Linking

Many PrimeTime commands attempt to link the current design for you. For example, **report_timing** invokes the linker if the current design is not linked. Automatic linking occurs only if the design is completely unlinked, and not if the current design is partially linked and has unresolved references. If the current design is totally linked, there is no need for an auto-link, so it is not attempted.

You can disable the auto-link process by setting the **auto_link_disable** variable to *true*. Normally this is not necessary, because the determination of whether the design is linked is very fast. The variable also provides compatibility with Design Compiler.

Unresolved References

If the linker fails to resolve one or more references, first determine and correct the source of the problem, then relink the design using the **link_design** command. Failures are typically caused by missing libraries or designs; incorrectly specified **link_path** or **search_path** variables; or a file that is in the path but is not accessible by you. After making the necessary corrections to your environment, you can decide whether to do an incremental relink or an initial link, or whether to allow the linker to create black boxes for the unresolved references.

If you can resolve the references by changing the **link_path** or **search_path** variables, you must relink the design.

The creation of black boxes is controlled by the variable **link_create_black_boxes**, whose default value is *true*. Unless you set this variable to *false*, the linker automatically converts each unresolved reference to a black box (an empty cell with no timing arcs). The result is a completely linked design on which analysis can be performed (assuming there are no other unrecoverable link errors). You can prevent unresolved references by ensuring that the **link_create_black_boxes** variable is set to *true*.

Black box creation can fail when there are multiple conflicting references, usually with generic logic. For example, if SELECT_OP has two references, one with five pins and the other with 20 pins, the second black box might not be created, and the design might not link. PrimeTime does not support generic logic, except for GTECH; if a design contains such generic logic, remove the design or replace it with an empty design.

Sometimes, black box creation fails. This will occur when there are multiple conflicting references. This happens most often with generic logic. For example, if there are two references to SELECT_OP, one with five pins, and the other with 20 pins, it is likely that the second black box will not be created, and design will not link. Other than GTECH, PrimeTime does not support generic logic. Designs containing such generic logic should be removed or replaced by empty designs.

Performance Considerations

The PrimeTime linker starts with the top design along with any other designs and libraries you loaded. During the link process, other necessary designs and libraries are loaded. When the link process completes successfully, it produces a design that can be analyzed.

By default, all subdesigns used to build the linked design are removed from memory. If you want to keep all designs and analyze them at later point, use the - **keep_sub_designs** option to maintain the subdesigns. Note that this takes more memory.

You need subdesigns only if you want to analyze more than one design in a session; the linked design contains all information necessary for analysis. For example, in PrimeTime, wireload models are set on specific *instances* of the design, without requiring a subdesign.

EXAMPLES

The following examples show how a design links with many of the different linker options. First, the link fails when a library cannot be found in the link path because of a typo in the search_path.

```
pt_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmoss"  
/designs/newcpu/v1.6/dbs /libs/cmoss  
pt_shell> set link_path "* cmos.db"  
* cmos.db
```

```

pt_shell> read_db newcpu.db
Loading db file '/design/newcpu/v1.6/dbs/newcpu.db'
1
pt_shell> link_design newcpu
Error: Can't read link_path file 'cmos.db' (LNK-001)
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Warning: Unable to resolve reference to 'NR4' in 'newcpu'. (LNK-005)
Warning: Unable to resolve reference to 'ND2' in 'newcpu'. (LNK-005)
Warning: Unable to resolve reference to 'FD2' in 'newcpu'. (LNK-005)
Information: Design 'newcpu' was not successfully linked:
          16 unresolved references. (LNK-003)

```

In the following example, correcting the value of the **search_path** variable and doing an initial link completely links the design without black boxes.

```

pt_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"
/designs/newcpu/v1.6/dbs /libs/cmos
pt_shell> link_design newcpu
Loading db file '/libs/cmos/cmos.db'
Linking design newcpu...
Loading db file '/designs/newcpu/v1.6/dbs/BOX1.db'
Loading db file '/designs/newcpu/v1.6/dbs/BOX2.db'
Loading db file '/designs/newcpu/v1.6/dbs/padring.db'
Design 'newcpu' was successfully linked.
1

```

SEE ALSO

```

current_design(2), list_designs(2), list_libraries(2), read_db(2),
auto_link_disable(3), link_create_black_boxes(3), link_force_case(3), link_path(3),
search_path(3).

```

list_attributes

Lists currently defined attributes.

SYNTAX

```
string list_attributes  
[-application]  
[-class class_name]  
[-nosplit]
```

Data Types

class_name string

ARGUMENTS

-application
 Lists application attributes as well as user-defined attributes.

class *class_name*
 Limit the listing to attributes of a single class. Valid classes are design, port, cell, net, and so on.

-nosplit
 Prevents line-splitting and facilitates writing software to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

DESCRIPTION

The **list_attributes** command displays an alphabetically sorted list of currently defined attributes. PrimeTime attributes divide into two categories: application-defined and user-defined. By default, **list_attributes** lists all user-defined attributes.

Using the **-application** option adds all application attributes to the listing. Note that there are many application attributes. It is often useful to limit the listing to a specific object class using the *class_name*.

EXAMPLES

This is an example listing of some attributes defined with **define_user_attribute**.

```
pt_shell> list_attributes  
  
*****  
Report : List of Attribute Definitions
```


Design :

Properties:

A - Application-defined

U - User-defined

I - Importable from db (for user-defined)

Attribute Name	Object	Type	Properties	Constraints
attr_b	cell	boolean	U	
attr_d	cell	double	U	
attr_f	cell	float	U	
attr_i	cell	int	U,I	
attr_ir1	cell	int	U	0 to 100
attr_ir2	cell	int	U	>= 0
attr_ir3	cell	int	U	<= 100
attr_oo	cell	string	U	A, B, C, D
attr_s	cell	string	U	
attr_s	net	string	U	

This example limits the listing to net attributes only.

pt_shell> **list_attributes -application -class net**

Report : List of Attribute Definitions

Design :

Properties:

A - Application-defined

U - User-defined

I - Importable from db (for user-defined)

Attribute Name	Object	Type	Properties	Constraints
area	net	float	A	
attr_s	net	string	U	
ba_capacitance_max	net	float	A	
ba_capacitance_min	net	float	A	
ba_resistance_max	net	float	A	
ba_resistance_min	net	float	A	
base_name	net	string	A	
full_name	net	string	A	
net_resistance_max	net	float	A	
net_resistance_min	net	float	A	
object_class	net	string	A	
pin_capacitance_max	net	float	A	
pin_capacitance_min	net	float	A	
total_capacitance_max	net	float	A	
total_capacitance_min	net	float	A	
wire_capacitance_max	net	float	A	
wire_capacitance_min	net	float	A	

SEE ALSO

`define_user_attribute(2)`
`get_attribute(2)`
`remove_user_attribute(2)`
`report_attribute(2)`
`set_user_attribute(2)`

list_delcalc_resources

Displays all delcalc resources. This includes both the name and the value.

SYNTAX

```
int list_delcalc_resources resource_name
string resource_name
```

ARGUMENTS

resource_name
Specifies the name of the resource.

DESCRIPTION

Displays all the delcalc resources. This includes both the name and the value. The argument is optional. Without the argument, all delcalc resources are displayed. With the argument, the specify resource is displayed.

EXAMPLES

This first example displays resource1. The second example displays all the resources that were previously defined.

```
pt_shell> list_delcalc_resources resource1
pt_shell> list_delcalc_resources
```

SEE ALSO

set_delcalc_resource (2), **remove_delcalc_resource** (2),

list_designs

Lists designs that have been read into PrimeTime.

SYNTAX

```
string list_designs [-all] [-only_used]
```

ARGUMENTS

-all

In addition to listing the designs in memory, **-all** lists the designs that are instantiated in the current design but are removed from memory.

-only_used

Lists only designs in use. These include the current design, any design that is linked or partially linked, and any designs instantiated in a linked design.

DESCRIPTION

The **list_designs** command lists the designs that have been read into PrimeTime. By default, only designs that are currently in memory are listed. When used with the **-all** option, **list_designs** also lists the designs that are instantiated in a linked design but have been removed from memory.

You can filter unused designs out of the display by using the **-only_used** option. Designs that are in use include the current design, any design that is linked or partially linked, and any designs instantiated in a linked design.

The **list_designs** command displays the status of the design. The notation used is

- * - Indicates that the design is the current design
- L - Indicates that the design is linked.
- N - Indicates that the design is not in memory.
- l - Indicates that the design is partially linked.

EXAMPLES

The following command lists the designs in memory.

```
pt_shell> list_designs
```

Design Registry:

*L AD4FULA	/u/foo/designs/top.db:AD4FULA
AD4PG	/u/foo/designs/add1.db:AD4PG
ADD5A	/u/foo/designs/add2.db:ADD5A
MULT1	/u/foo/designs/top.db:MULT1

The following command lists only the designs that are in use and in memory.

```
list_designs
```

```
pt_shell> list_designs -only_used
```

```
Design Registry:
```

```
*L AD4FULA      /u/foo/designs/top.db:AD4FULA
  ADD5A         /u/foo/designs/add2.db:ADD5A
  MULT1         /u/foo/designs/top.db:MULT1
```

After removing design MULT1, you can display it using the **-all** option only.

```
pt_shell> remove_design MULT1
```

```
Removing design 'MULT1'...
```

```
1
```

```
pt_shell> list_designs -all -only_used
```

```
Design Registry:
```

```
*L AD4FULA      /u/foo/designs/top.db:AD4FULA
  ADD5A         /u/foo/designs/add2.db:ADD5A
  N MULT1       /u/foo/designs/top.db:MULT1
```

SEE ALSO

current_design (2), **link_design** (2), **remove_design** (2).

list_key_bindings

Displays all the key bindings and edit mode of current shell session.

SYNTAX

```
int list_key_bindings [-nosplit]
```

ARGUMENTS

-nosplit

Indicates that lines are not to be split when column fields overflow.

DESCRIPTION

The **list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, variable **line_editing_mode** can be set in either the .synopsys_pt.setup file or directly in the shell.

The text CTRL+K is read as 'Control+K' and describes the character produced when the k key is hit while the Control key is depressed.

The text META+K is read as 'Meta+K' and describes the character produced when the Meta key is depressed, and the k key is hit. The Meta key is labeled ALT on many keyboards. On keyboards with two keys labeled ALT (usually to either side of the space bar), the ALT on the left side is generally set to work as a Meta key. The ALT key on the right may also be configured to work as a Meta key or may be configured as some other modifier, such as a Compose key for typing accented characters.

If you do not have a Meta or ALT key, or another key working as a Meta key, the identical keystroke can be generated by typing ESC first, and then typing k. Either process is known as metafying the k key.

Alternative key bindings work only in vi alternate(command) mode.

EXAMPLES

SEE ALSO

sh_enable_line_editing (3), **sh_line_editing_mode** (3).

list_libraries

Lists all libraries that are read into PrimeTime.

SYNTAX

```
string list_libraries [-only_used]
```

ARGUMENTS

-only_used

Indicates only the list libraries in use. A library is in use if a linked design links to library cells from the library.

DESCRIPTION

The **list_libraries** command lists the libraries that are read into PrimeTime. You can filter unused libraries out of the display by using the **-only_used** option. A library is in use if a linked design links to library cells from the library.

An asterisk (*) to the left of a library indicates that this is the main library for the current design. The main library is the first library in the link path which has a time unit. An upper-case 'M' to the left of a library indicates that the library is the max library of a max/min library relationship created by **set_min_library**. A lower-case 'm' to the left of a library indicates that the library is the min library of a max/min library relationship created by **set_min_library**.

EXAMPLES

The following example lists all libraries.

```
pt_shell> list_libraries
```

Library Registry:

```
bus1_lib      /u/foo/lib/bus1_lib.db:bus1_lib
tech1         /u/foo/lib/tech1.db:tech1
```

After linking a design, the main lib can be identified. Further, using **-only_used** limits the display to the libraries that were used to link the current design.

```
pt_shell> link_design top_flat
```

Linking design top_flat...

Designs used to link top_flat:

<None>

Libraries used to link top_flat:

tech1

Design 'top_flat' was successfully linked.

1

```
pt_shell> list_libraries -only_used
```

Library Registry:

```
* tech1          /u/foo/lib/tech1.db:tech1
```

SEE ALSO

current_design (2), **link_design** (2), **list_designs** (2), **set_min_library** (2).

list_licenses

Shows the licenses which are currently checked out.

SYNTAX

string **list_licenses**

DESCRIPTION

The **list_licenses** command lists the licenses which you currently have checked out. **list_licenses** always returns the empty string.

EXAMPLES

This example shows the output from **list_licenses**.

```
pt_shell> list_licenses
```

Licenses in use:

 PrimeTime

 Stamp-Compiler

SEE ALSO

get_license (2), **license_users** (2), **remove_license** (2).

load_of

Gets the capacitance of a library cell pin. It is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
float load_of lib_pin  
string lib_pin
```

ARGUMENTS

lib_pin

Specifies the name of the library cell pin, or a collection that contains the library cell pin, for which to get the capacitance.

DESCRIPTION

The **load_of** command returns the capacitance of the given library cell pin. The command exists in PrimeTime for compatibility with Design Compiler. Complete information about the **load_of** command can be found in the Design Compiler documentation. The supported method for getting the capacitance of a library cell pin is by using the **get_attribute** command with the **pin_capacitance** attribute.

SEE ALSO

get_attribute (2).

load_upf

Reads in a script in Unified Power Format (UPF) format.

SYNTAX

```
int load_upf  
upf_file_name  
[-scope instance_name]  
[-version upf_version]
```

Data Types

```
upf_file_name      string  
instance_name      string  
upf_version        string
```

ARGUMENTS

```
file_name  
    Specifies the name of the file that contains the UPF script to be read.  
  
-scope instance_name  
    Specifies the scope where the UPF commands contained in file upf_file_name  
    are to be executed.  
  
-version upf_version  
    Specifies the version of UPF to which the file conforms. Allowed version is  
    1.1 (default).
```

DESCRIPTION

The **load_upf** command sets the scope to the specified instance and executes the set of UPF commands in the file upf_file_name. Upon return, the current scope is restored to what it was prior to invocation. When -scope is not specified, the current scope is used. The commands are executed the same way as TCL **source** command.

EXAMPLES

Following example loads UPF script file top.upf. Usage of -scope and -version options have been illustrated.

```
prompt> load_upf top.upf  
1
```

SEE ALSO

`help upf`, `source(2)`, `create_power_domain(2)`.

map_design_mode

Maps specified design modes to cell modes and/or paths

SYNTAX

```
string map_design_mode  
design_mode  
[-from from_pin_list]  
[-to to_pin_list]  
[-through through_pin_list]  
[cell_mode_list]  
[instance_list]
```

```
list design_mode  
list from_pin_list  
list to_pin_list  
list through_pin_list  
list cell_mode_list  
list instance_list
```

ARGUMENTS

design_mode
Specifies a design mode to be mapped to cell modes and/or paths. The design modes on the list must have been previously created using the **define_design_mode_group** command.

-from from_pin_list
Specifies a timing path, from a given pin of the design, that is active only for the specified design mode. The given paths are inactive for other design modes which are in the same design mode group.

-to to_pin_list
Specifies a timing path, to a given pin of the design, that is active only for the specified design mode. The given paths are inactive for other design modes which are in the same design mode group.

-through through_pin_list
Specifies a timing path, through a given pin of the design, that is active only for the specified design mode. The given paths are inactive for other design modes which are in the same design mode group.

instance_list
Specifies a list of cells in which the specified modes are to be mapped to the design mode. This list must be accompanied by a **cell_mode_list**

cell_mode_list
Specifies a list of cell modes which are to be mapped to the design_mode.

DESCRIPTION

The **map_design_mode** command maps design modes to cells and or paths. The design modes must have been previously created by **define_design_mode_group**.

A design mode can be mapped to a cell mode using **map_design_mode cell_mode_list instance_list design_mode**. If the design mode is made active then all cell modes mapped to the design mode also become active. If a design_mode changes from inactive to active.

A path can be mapped to a design mode using **map_design_mode -from [from_pin_list] -to [to_pin_list] -through [through_pin_list] design_mode** When a path is mapped to a design mode and the design mode is set inactive, the path becomes a false path. If the design mode is subsequently set active, then the path is reset and is no longer considered false.

To see a report of the design modes specified for the current design, use the **report_mode -type design** command. The report also shows the cell modes and paths mapped to each design mode.

EXAMPLES

The following example defines two design modes, DM1 and DM2, maps the cell mode READ to design mode DM1 for instance Uram1, then removes the design mode DM1. Removing DM1 also removes the mapping to the active cell mode READ.

```
pt_shell> define_design_mode_group {DM1 DM2}
pt_shell> map_design_mode READ Uram1 DM1
pt_shell> remove_design_mode DM1
```

SEE ALSO

map_design_mode (2), **define_design_mode_group** (2), **report_mode** (2), **set_mode** (2), **reset_mode** (2).

max_variation

Takes the maximum of two or more variations. Returns a collection (that corresponds to this max variation).

SYNTAX

```
collection max_variation  
variation_list  
collection variation_list
```

ARGUMENTS

```
variation_list  
    List of the variations to be max'ed.
```

DESCRIPTION

Constructs the maximum of two or more variations. This maximum is a variation. The command creates the max variation and returns it as a collection.

EXAMPLES

The following example computes the maximum of two variations \$v1 and \$v2 (assuming they have already been created) and saves it to collection \$vmax, which is a new variation collection.

```
pt_shell> set col [add_to_collection $v1 $v2]  
_sel128  
pt_shell> set vmax [max_variation $col]  
_sel129
```

SEE ALSO

create_variation (2), **add_variation** (2), **sub_variation** (2), **min_variation** (2).

mem

Retrieves the total memory allocated by the current `pt_shell` process.

SYNTAX

```
int mem
```

DESCRIPTION

The *mem* command returns the size of memory currently allocated by the current `pt_shell` process for the purposes of storing design netlist, design annotations, and constraints information as well as computed timing information. The value printed represents the amount in kilobytes (kB).

Note that the value reported would not match values obtained by the user through Unix process tracking commands, such as *top*. This is the case because the *mem* command does not track memory used to load the executable and maintain process stack space. Also, it does not differentiate between resident and swapped memory.

```
pt_shell> mem  
8092
```

SEE ALSO

`cputime(2)`

merge_models

Merges multiple timing models (in LIB format) together to be one.

SYNTAX

```
string merge_models  
[-lib_files lib_files]  
-mode_names mode_names  
[-group_name mode_group_name]  
-output output_file_name  
[-formats format_list]  
[-tolerance merge_tolerance]  
[-single_mode]  
[-keep_all_arcs]
```

Data Types

<i>model_files</i>	list
<i>data_files</i>	list
<i>lib_files</i>	list
<i>mode_names</i>	list
<i>format_list</i>	list
<i>mode_group_name</i>	string
<i>output_file_name</i>	string
<i>merge_tolerance</i>	float

ARGUMENTS

-lib_files *lib_files*
Specifies a list of Synopsys internal library (.lib) files to read, compile, and merge. The files must have been extracted using the **extract_models** command. Substitute the list you want for the *lib_files* option.

-mode_names *mode_names*
Specifies a list of mode names for the list of models. The mode names must match the list of library files specified. Substitute the list you want for the *mode_names* option

-group_name *mode_group_name*
Specifies the name of the mode group to which the models belong. If you do not specify this option, the default group name is *etm_modes*.

-output *output_file_name*
Specifies the name of the output file that is used to save the merged model. Substitute the name you want for *output_file_name*.

-formats *format_list*
Specifies the format(s) in which the merge model will be written, either db, lib, or both. Substitute the format(s) you want for *format_list*.

-tolerance *merge_tolerance*
Specifies the float tolerance used to compare delay or transition values of

arcs in the timing models. It defines the tolerance for floating point number comparison when the timing arcs in the models are matched and compared against each other to determine they are equal. Float numbers are considered equal if the difference between them is within the tolerance. The default is 0.04. Substitute the tolerance you want for *merge_tolerance*, but the value must be greater than or equal to zero. This cannot be used with *keep_all_arcs*.

-single_mode

Forces one timing arc to have only one defined mode or forces there to be only one single mode per timing arc in the merged model. This is necessary for downstream tools that cannot handle more than one mode on a timing arc. This normally results in larger, less compact timing models. This *single_mode* cannot be used with *keep_all_arcs*.

-keep_all_arcs

This makes the *merge_models* option ignore the *merge_tolerance* and suppresses all arc merging, and it implies *single_mode*. Use this option when performing two independent merges and the resulting merged models must have the same number of arcs. Resulting models are very large. This option cannot be used with either the *single_model* or *merge_tolerance* option.

DESCRIPTION

Merges multiple timing models together to be one. The timing models must be in Synopsys internal library (.lib) format. This command reads in all the specified files to generate a list of timing models. These timing models are processed, matched, compared, and merged to create a single timing model. This model has moded timing arcs such that, in any given mode, the static timing behavior of the merge model is equivalent to one of the models merged. The final merged model is saved on disk in the specified files and requested formats. PrimeTime and Design Compiler recognize the merged model with moded timing arcs for performing timing analysis.

The *keep_all_arcs* argument should only be used when a timing model has been extracted under several different conditions and the results are to be merged into two or more models in situations where the number of arcs must match. For example, say that a model TEST is extracted with two modes A and B, operating conditions min and max. There are four extractions, min A, min B, max A and max B. Now you want to merge the min's for modes A and B to get a min model, and the max's to get a max model. If you were going to use the merged models as the min and max libraries of **set_min_library** then their arcs must match, so you should merge the models using the *keep_all_arcs* option.

EXAMPLES

The following command reads in the LIB models specified by the sram_write.lib / sram_read.lib files and generates a merged model containing an "SRAM_MODE" mode group with two "READ" and "WRITE" component modes. Timing arcs existing only in sram_write.lib are put into the merged model and marked with "WRITE" mode. Timing arcs existing only in sram_read.lib are marked with "READ" mode. Timing arcs existing in both models are compared to each other by their delay/transition values. If they are within 0.1 time units of each other, they are considered equal, and only one is kept in the merged model with no mode marked on the arc (valid for both modes); otherwise both arcs are kept in the merged model with each arc marked with a

proper mode depending on which model it came from. Because the command requests all formats, the merged model is saved as `sram_model_lib.db` in Synopsys Database format (`.db`), and `sram_model.lib` in Synopsys internal library format (`.lib`).

```
pt_shell> merge_models -lib_files \  
{sram_write.lib sram_read.lib} \  
-mode_names {WRITE READ} -mode_group SRAM_MODE \  
-output sram_model -format {db lib} \  
-tolerance 0.1
```

SEE ALSO

`extract_model(2)`

merge_saif

Reads a list of SAIF files with their corresponding weights, annotates switching activity attributes with merged toggle_rate, glitch_rate and merged static_probability for nets, pins, ports, and power arcs in the current instance, and generates a merged output SAIF file.

SYNTAX

```
int merge_saif -input_list saif_file_and_weight_list
    [-strip_path inst_name]
    [-path prefix]
    [-output merged_saif_name]
    [-simple_merge]
    [-ignore ignore_name]
    [-exclude exclude_file_name]
    [-derate_glitch value]
    [-quiet]
```

```
list saif_file_and_weight_list
string inst_name
string merged_saif_name
string ignore_name
string exclude_file_name
float value
```

ARGUMENTS

-input_list *saif_file_and_weight_list*

Specifies the name and the corresponding weight of the SAIF file list. *saif_file_and_weight_list* contains a list of { -input name.saif -weight number }, where "-input" and "-weight" are keywords. In addition, 0 < number < 100, and the sum of all weights is equal to 100. For example, { -input gate_back_1.saif -weight 20 -input gate_back_2.saif -weight 80 } means that the files gate_back_1.saif and gate_back_2.saif are weighted by 20% and 80%, respectively.

-strip_path *inst_name*

Specifies the name as it appears in each SAIF file of the current design instance you want to annotate. **merge_saif** annotates all sub instances in the hierarchy of the specified instance, as well as annotating the instance itself.

-path *prefix*

Specifies a relative path from the current design to the hierarchical low-level design for which the SAIF file has been created. By default, absolute path names are used. Use this option if the SAIF file refers to an object in a hierarchy. Also if the user wants to read the switching information for a portion of the design this option can be used. For example if the SAIF was generated for the whole design and the user wants to put switching information only on the nets/ports on the boundary and hierarchically under instance add14, then **-path** add14 option can be used.

-output *merged_saif_name*
 Specifies the name of the output merged SAIF file.

-simple_merge
 Indicates that all SAIF files are to annotate 100% of the nets, ports and pins of the current instance, and the **merge_saif** command *is to perform a simple merge of SAIF file data (without propagating the switching activity of non-annotated objects)*. Use the **report_switching_activity** command to verify that each SAIF file annotates 100% of the current instance.

-ignore *ignore_name*
 Specifies the name of an instance in the SAIF file for which switching activity is to be ignored. The **merge_saif** command ignores switching activity within the hierarchy of that instance in the SAIF file.

-exclude *exclude_file_name*
 Specifies the name of a file that contains a list of names to be ignored. *exclude_file_name* is used when you want to ignore switching activity for several instances. The file must contain each *ignore_name* on a separate line, without the **-exclude** option. As with the **-ignore** option, the ignore names are recognized after the **-strip_path** argument.

-derate_glitch *value*
 Specifies a default derating factor value to be used for inertial glitches in the SAIF file. If **-derate_glitch** is not specified, a default derating factor of 0.5 is used.

-quiet
 Specifies quiet mode and for instance suppresses warnings on design objects in the SAIF file that could not be annotated on the design.

DESCRIPTION

The **merge_saif** command reads a list of SAIF files with weight, computes the merged toggle_rate, glitch_rate, and static_probability, and annotates the switching activity attributes toggle_rate, glitch_rate, and static_probability for nets, ports, pins, and power arcs of the current instance. In addition, the **merge_saif** command optionally generates a merged output SAIF file. To identify the instance (and corresponding subinstances) you want to annotate, use the **-strip_path** argument. To specify a default derating factor other than 0.5, use the **-derate_glitch** option.

EXAMPLES

The following example reads and merges weighted SAIF files, annotates switching activity for the current design, and generates an output SAIF file.

```
pt_shell> merge_saif -input_list { -input gate_back1.saif -weight 10 -input
gate_back2.saif -weight 20 -input gate_back3.saif -weight 30 -input
gate_back4.saif -weight 40 } -strip_path E/UUT -output merged_saif.saif
```

SEE ALSO

```
set_switching_activity(2), set_rtl_to_gate_name(2), get_switching_activity(2),  
reset_switching_activity(2), report_switching_activity(2), write_saif(2),  
report_power(2);
```

min_variation

Takes the minimum of two or more variations. Returns a collection (that corresponds to this min variation).

SYNTAX

```
collection min_variation  
variation_list  
collection variation_list
```

ARGUMENTS

```
<italic>variation_list  
    List of the variations to be min'ed.
```

DESCRIPTION

Constructs the minimum of two or more variations. This minimum is a variation. The command creates the min variation and returns it as a collection.

EXAMPLES

The following example computes the minimum of two variations \$v1 and \$v2 (assuming they have already been created) and saves it to collection \$vmin, which is a new variation collection.

```
pt_shell> set col [add_to_collection $v1 $v2]  
_sel128  
pt_shell> set vmin [min_variation $col]  
_sel129
```

SEE ALSO

create_variation (2), **add_variation** (2), **sub_variation** (2), **min_variation** (2).

print_message_info

Prints information about diagnostic messages which have occurred or have been limited.

SYNTAX

```
string print_message_info [-ids id_list] [-summary]  
list id_list
```

ARGUMENTS

List of message identifiers to report. Each entry can be a specific message or a glob-style pattern which matches one or more messages. If this option is omitted and no other options are given, then all messages which have occurred or have been limited will be reported.

Generate a summary of error, warning and informational messages which have occurred so far.

DESCRIPTION

The `print_message_info` command enable you to print summary information about error, warning, and informational messages which have occurred or have been limited with the `set_message_info` command. For example, if the following message is generated, information about it is recorded.

```
Error: unknown command 'wrong_command' (CMD-005)
```

It is useful to be able to summarize all recorded information about generated diagnostic messages. Much of this can be done using the `get_message_info` command but you need to know a specific message id. By default, `print_message_info` will summarize all of the information. It will provide a single line for each message which has occurred or has been limited, and one summary line which shows the total number of errors, warnings, and informational messages which have occurred so far. If an `id_list` is given, then only messages matching those patterns will be displayed. If `-summary` is given, then a summary will be displayed.

Using a pattern in the `id_list` is intended to show a specific message prefix, for example, "CMD*". Note that this will not show all messages with that prefix. Only those which have occurred or have been limited will be displayed. The limit column shows the limit set by `set_message_info` by default. But if the limit is not set, and this type of message is included in `sh_limited_messages`, this column shows the default limit `sh_message_limit`, and a symbol '*' also appears to show that this limit is from `sh_message_limit`.

EXAMPLES

The following example uses `print_message_info` to show a few specific messages.

```
shell> print_message_info -ids [list "CMD*" APP-99]
```



```
Id Severity Limit Occurrences Suppressed -----  
----- CMD-005 Error 0 7 2 APP-99 Information 1 0 0
```

At the end of the session, you might want to generate some information about a set of interesting messages, such as how many times each occurred (which includes suppressions), how many times each was suppressed, and whether a limit was set for any of them. The following shows how to use `print_message_info` in this way. The symbol '*' in the limit column of PARA-004 means that this message is in **sh_limited_messages**, and this limit is the **sh_message_limit**.

```
shell> print_message_info
```

```
Id Severity Limit Occurrences Suppressed -----  
----- CMD-005 Error 0 12 0 PARA-004 Warning 100 * 1 0 APP-027  
Information 100 150 50 APP-99 Information 0 1 0
```

Diagnostics summary: 12 errors, 151 warnings, 1 informational

Note that the suppressed count is not necessarily the difference between the limit and the occurrences, since the limit can be dynamically changed with `set_message_info`, or the limit is from **sh_message_limit**.

The sorting is by Severity (Error, Warning, Information), then Occurrences in descending order, then by Id.

SEE ALSO

get_message_info (2), **set_message_info** (2), **suppress_message** (2), **sh_message_limit** (3), **sh_limited_messages** (3)

query_objects

Searches for and displays objects in the database.

SYNTAX

```
string query_objects  
[-verbose]  
[-class class_name]  
[-truncate elem_count]  
object_spec
```

Data Types

<i>class_name</i>	string
<i>elem_count</i>	int
<i>object_spec</i>	list

ARGUMENTS

-verbose

Displays the class of each object found. By default, only the name of each object is listed. With this option, each object name is preceded by its class, as in "cell:U1/U3".

-class *class_name*

Establishes the class for a named element in the *object_spec*. Valid classes are design, cell, net, and so on.

-truncate *elem_count*

Truncates display to *elem_count* elements. By default, up to 100 elements display. To see more or less elements, use this option. To see all elements, set the *elem_count* to 0.

object_spec

Provides a list of objects to find and display. Each element in the list is either a collection or an object name. Object names are explicitly searched for in the database with class *class_name*.

DESCRIPTION

The **query_objects** command (a simple interface) finds and displays objects in PrimeTime's runtime database. The command does not have a meaningful return value; it simply displays the objects found and returns the empty string.

The *object_spec* is a list containing collections and/or object names. For elements of the *object_spec* that are collections, **query_objects** simply displays the contents of the collection.

For elements of the *object_spec* that are names (or wildcarded patterns), **query_objects** searches for the objects in the class specified by *class_name*. Note that **query_objects** does not have a predefined, implicit order of classes for which searches are initiated. If you do not specify *class_name*, only those elements that are collections are displayed. Messages are displayed for the other elements (see

EXAMPLES).

To control the number of elements displayed, use the *-truncate* option. If the display is truncated, you see the elipsis (...) as the last element. Note that if the default truncation occurs, a message displays showing the total number of elements that would have displayed (see EXAMPLES).

NOTE: The output from **query_objects** looks similar to the output from any command which creates a collection, but remember that the result of **query_objects** is always the empty string.

EXAMPLES

These following examples show the basic use of the **query_objects** command.

```
pt_shell> query_objects [get_cells o*]
{"or1", "or2", "or3"}
pt_shell> query_objects -class cell U*
{"U1", "U2"}
pt_shell> query_objects -verbose -class cell \
?
[["list U* [get_nets n1]]]
{"cell:U1", "cell:U2", "net:n1"}
```

When you omit the *-class* option, only those elements of the *object_spec* that are collections generate output. The other elements generate error messages.

```
pt_shell> query_objects [list U* [get_nets n1] n*]
Error: No such collection 'U*' (SEL-001)
Error: No such collection 'n*' (SEL-001)
{"n1"}
```

When the output is truncated, you get the elipsis at the end of the display. For the following example, assume the default truncation is 5 (it is actually 100).

```
pt_shell> query_objects [get_cells o*] -truncate 2
{"or1", "or2", ...}
pt_shell> query_objects [get_cells *]
{"or1", "or2", "or3", "U1", "U2", ...}
Output truncated (total objects 126)
```

SEE ALSO

collections(2) get_cells(2) get_clocks(2) get_designs(2) get_generated_clocks(2)
get_lib_cells(2) get_lib_pins(2) get_libs(2) get_nets(2) get_path_groups(2)
get_pins(2) get_ports(2) get_qtm_ports(2) get_timing_paths(2)

read_aocvm

Reads advanced on-chip variation (OCV) derate factor tables.

SYNTAX

```
int read_aocvm  
aocvm_file
```

Data Types

aocvm_file string

ARGUMENTS

aocvm_file
 Specifies the name of the advanced OCV file.

DESCRIPTION

The **read_aocvm** command reads advanced OCV derate factor tables from a disk file. The tables are annotated onto one or more design objects. Allowed design object types are hierarchical cells, library cells and designs. The advanced OCV data is used to reduce pessimism and improve the accuracy of results.

When applying advanced OCV tables on multiple object types that apply to a given arc, then the following priority (in decreasing order of precedence) is used to determine the table that applies to the arc:

- 1) *Hierarchical cell table*
- 2) *Library cell table*
- 4) *design table*

Tables are processed in the order that they appear in the advanced OCV file. The last table of a specific type {object_type, rf_type, delay_type, derate_type} to be defined for an object in the file overwrites previous tables defined for the same object of the same type.

The **read_aocvm** command can read binary and compressed binary advanced OCV files created using the **write_binary_aocvm** command. No additional arguments are required to read binary or compressed binary advanced OCV files.

EXAMPLES

The following example shows an advanced OCV derate file *test.aocvm* with a single table annotated on all library cells.

```
pt_shell> sh cat test.aocvm
```

```
version:          1.0

object_type:      lib_cell
rf_type:          rise
delay_type:       cell
derate_type:      late
object_spec:      my_lib/*
voltage:          1.02
depth:            1 2 3
distance:         100 1000
table:            1.20 1.10 1.08 \
                  1.22 1.15 1.11
```

```
pt_shell> read_aocvm test.aocvm
```

SEE ALSO

```
report_timing(2)
get_timing_path(2)
remove_aocvm(2)
report_aocvm(2)
set_aocvm_coefficient(2)
write_binary_aocvm(2)
```

read_db

Reads in one or more design or library files in Synopsys database (db) format.

SYNTAX

```
string read_db [-netlist_only] [-library] file_names  
list file_names
```

ARGUMENTS

-netlist_only

For designs only; ignored for libraries. Indicates that only the netlist is to be read; attributes are not to be read. This can greatly increase performance while reading designs.

-library

Indicates that the file is a library db file; using this option optimizes the reading of large library db files.

file_names

Specifies names of one or more files to be read. Typically, only one file is read.

DESCRIPTION

Reads design and library information from Synopsys database (db) files into PrimeTime.

To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**. To determine the file that **read_db** loads, use the **which** command.

Each file can contain one or more design objects. After the files are loaded, to view the design objects, use the **list_designs** and **list_libraries** commands. The db file itself is used only during the read process, and is transformed into the internal format of PrimeTime during **read_db**. To remove designs and libraries, use the **remove_design** and **remove_lib** commands.

Synopsys design db files are used for many tools and can contain many complex attributes. However, only certain attributes apply to PrimeTime (for example, clocks, back-annotation). By default, these attributes are loaded from the db file. However, all information can also be brought into PrimeTime through scripts or other file formats (for example, SDF). In some cases (most notably, SDF), you can save CPU time and memory usage by reading the information from the alternate source. In these cases, use the **-netlist_only** option to read the db file. The **-netlist_only** option speeds up the read process and drastically reduces memory usage in the case of SDF back-annotation. You can then read the SDF file using **read_sdf**.

If the variable **power_enable_analysis** is set to true, **read_db** will also load power models in library db files.

EXAMPLES

In the following example, the file `newcpu.db` is read based on the `search_path`.

```
pt_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"  
/designs/newcpu/v1.6/dbs /libs/cmos
```

```
pt_shell> read_db newcpu.db  
Loading db file '/designs/newcpu/v1.6/dbs/newcpu.db'  
1
```

SEE ALSO

```
list_designs(2), list_libraries(2), read_sdf(2), remove_design(2), remove_lib(2),  
which(2); search_path(3).
```

read_ddc

Reads in design files in the Synopsys DDC format.

SYNTAX

```
string read_ddc [-netlist_only] [-scenario scenario_name] file_names  
string scenario_name  
list file_names
```

ARGUMENTS

-netlist_only

Indicates that only the netlist is to be read; attributes are not to be read.
This can greatly increase performance while reading designs.

-scenario *scenario_name*

Specifies the name of the scenario **read_ddc** will read to get the attributes.
The DDC file should have been created using scenarios.

file_names

Specifies names of one or more files to be read. Typically, only one file is read.

DESCRIPTION

Reads design information from Synopsys DDC files into PrimeTime. DDC is a proprietary file format to store netlist information and constraint data. It is generated by DesignCompiler, PhysicalCompiler XG mode, or IC Compiler.

To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**.

The DDC design files can contain many complex attributes. However, only certain attributes apply to PrimeTime (for example, clocks and other design constraints). By default, these attributes are loaded from the DDC file. However, all information can also be brought into PrimeTime through external scripts or other file formats (for example, SDC files). In these cases, use the **-netlist_only** option to read the DDC file.

EXAMPLES

In the following example, the file newcpu.ddc is read based on the *search_path*.

```
pt_shell> set search_path "/designs/newcpu/v1.6/dbs /libs/cmos"  
/designs/newcpu/v1.6/dbs /libs/cmos
```

```
pt_shell> read_ddc newcpu.ddc  
Loading ddc file '/designs/newcpu/v1.6/dbs/newcpu.ddc'  
Loaded 1 design.  
newcpu
```

```
read_ddc  
376
```


SEE ALSO

`read_db(2)`, `list_designs(2)`, `remove_design(2)`, `search_path(3)`.

read_file

Reads a netlist or library file. This is a DC Emulation command provided for compatibility with Design Compiler.

SYNTAX

```
string read_file [-format type] [-single_file ns1] [-names_file ns2] [-define ns3]  
file_list  
stringtype  
stringns1  
stringns2  
stringns3  
list file_list
```

ARGUMENTS

-format *type*
Specifies the file format. Allowed values are *db*, *edif*, *vhdl*, and *verilog*.

-single_file *ns1*
Not supported by PrimeTime.

-names_file *ns2*
Not supported by PrimeTime.

-define *ns3*
Not supported by PrimeTime.

file_list
A list of files that are to be read.

DESCRIPTION

The **read_file** command reads in one or more netlist or library files. The command exists in PrimeTime for compatibility with Design Compiler. Complete information about the **read_file** command can be found in the Design Compiler documentation. The supported method for reading netlist or library files is to use the following commands: **read_db**, **read_verilog**, **read_edif**, and **read_vhdl**.

SEE ALSO

read_db (2), **read_edif** (2), **read_verilog** (2), **read_vhdl** (2).

read_lib

Reads in a Synopsys library (.lib) file.

SYNTAX

```
Boolean read_lib file_name  
list file_name
```

ARGUMENTS

file_name
Specifies the name of a library file to read.

DESCRIPTION

The **read_lib** command reads a Synopsys library (.lib) file into PrimeTime using the PrimeTime external reader (ptxr). To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**. To determine the file that **read_lib** loads, use the **which** command. After the library file is loaded, to list the library objects, use the **list_libraries** command. To remove libraries, use the **remove_lib** command.

If the variable `power_enable_analysis` is set to true, `read_lib` will also load power models. If the variable is not set or it's false, neither `read_lib` nor `link_design` and the first power related command will load load power models. In other words, the incremental power model loading is not supported if `read_lib` is used. The incremental power model loading only works on db files by using `set link_library` and/or `read_db`.

For more information, consult the Library Compiler Reference Manual.

EXAMPLES

In the following example, the file `bus1.lib` is read based on the **search_path**.

```
pt_shell> set search_path "/designs/newcpu/v1.6 /libs/cmos"  
/designs/newcpu/v1.6 /libs/cmos
```

```
pt_shell> read_lib bus1.lib  
Beginning read_lib...  
Reading '/libs/cmos/bus1.lib' ...  
Technology library 'bus1' read successfully  
1  
pt_shell> list_libraries
```

```
Library Registry:  
    bus1          bus1.lib:bus1
```

1

SEE ALSO

```
list_libraries(2), remove_lib(2), which(2); search_path(3).  
power_enable_analysis(3).
```

read_milkyway

Reads in one linked design from milkyway database.

SYNTAX

```
int read_milkyway [-version version] [-netlist_only] [-library design_library] [-  
scenario scenario_name] CEL_name  
string CEL_name  
string scenario_name  
string design_library
```

ARGUMENTS

-version *version*

Specifies the version of the design to be read. For example, there are design files under the CEL view in the milkyway design library design_lib: 'design_lib/CEL/design1_pre_route:1', 'design_lib/CEL/design1_post_route:2' etc. The 1 or 2 after the ':' is the version number of the design. The default is to read the most current version.

-netlist_only

Indicates that only the netlist is to be read; constraints are not read. The default is to read both netlist and constraints.

-library *design_library*

Specifies the absolute or relative path to the MW design library. This option can be left out if the variable **mw_design_library** specifies the path to the MW design library.

-scenario *scenario_name*

MW database is capable of storing multiple constraints that can correspond to various scenarios of running the design. This option specifies the name of the scenario for reading in constraints from MW database. The default is to not use a scenario.

CEL_name

Specifies the design filename to be read. For example, there are design files under the CEL view in the milkyway design library design_lib: 'design_lib/CEL/design1_pre_route:1', 'design_lib/CEL/design1_post_route:2' etc. The design1_pre_route or design1_post_route are the CEL_name argument. Do not include version number in this argument.

DESCRIPTION

Reads a linked design from milkyway database into PrimeTime. The command can also optionally load the timing constraints from the milkyway database.

The **read_milkyway** command always reads in a linked design. Conceptually it takes the place of the usual read / link sequence in PrimeTime, and so the client must set the **link_path** and **search_path** before invoking the command. If there is already a linked design when **read_milkyway** is invoked, then the existing design is unlinked before

the Milkyway design is read in. This is consistent with the way the **link_design** command works.

A **link** following a **read_milkyway** is not necessary and should not be used because it could invalidate certain types of cell hierarchy. This method of reading Milkyway designs is consistent with other tools, but is different from reading Verlog or VHDL designs, which require a link.

read_milkyway command can be affected by the values of the following variables:

link_path: This will be used to perform the implicit link during the **read_milkyway** command.

link_path_per_instance: If **link_path_per_instance** variable is set so that certain instances have special **link_path**, that variable setting will be honored even in **read_milkyway** flow.

search_path: This will be used for auto-loading the technology libraries.

mw_design_library: If the library option is missing from the command line then **read_milkyway** gets the library path from this variable. If the library option is present then the variable is set to the value of the library option.

mw_logic0_net: Use this name for logic 0 constant nets. The default is "VSS".

mw_logic1_net: Use this name for logic 1 constant nets. The default is "VDD".

The variable **link_create_black_boxes** will not affect the functionality of **read_milkyway** command. This is because **read_milkyway** reads a linked design. If an expected library cell is not present in the libraries specified by **link_path** (or **link_path_per_instance**), then a black box will be created irrespective of the value of the variable **link_create_black_boxes**.

EXAMPLES

In the following example, the latest version of design **mw_des** is read into PrimeTime from the **mw_db** milkyway database. Any nets set to a constant 0 are created as "VSS", and any set to a constant 1 are created as "VDD".

```
pt_shell> read_milkyway -netlist_only -library ./mw_db mw_des
1
```

SEE ALSO

link_path(3), **search_path(3)**, **mw_logic0_net(3)**, **mw_logic1_net(3)**,
link_path_per_instance(3), **remove_design(2)**, **list_designs(2)**, **mw_design_library(3)**.

read_parasitics

Reads net parasitics information from an SPEF, DSPF, RSPF, PARA, or binary parasitics file (SBPF) and uses it to annotate the currently linked design.

SYNTAX

Boolean **read_parasitics**
[-format *file_fmt*]
[-complete_with *completion_type*]
[-lumped_cap_only]
[-pin_cap_included] [-increment]
[-path *list*]
[-keep_capacitive_coupling]
[-coupling_reduction_factor *factor*]
[-triplet_type *ttype*]
[-verbose] [-syntax_only]
 [-eco]
 [-original_file_name *file_name*]
[-ilm_context]
[-keep_variations]
 [-create_default_variations]
file_names

Data Types

file_fmt string *completion_type* string *path_name* string *file_names* string *ofname*
string *factor* float

ARGUMENTS

-format *file_fmt*
Specifies the format of the parasitics file. Allowed values are SPEF, DSPF, RSPF, PARA (Milkyway), and SBPF (Synopsys Binary Parasitics Format). If the -format option is not specified, the application can determine whether the file is SBPF, SPEF, DSPF, RSPF, or a compressed version of those three ASCII formats. However, to read a file in PARA you must specify the -format PARA option.

-complete_with *completion_type*
This option does not apply to the RSPF format.
Indicates that a net with partially annotated parasitics is to be completed by inserting capacitances and resistances according to the *completion_type* option. The allowed values are *zero*, which completes the net by inserting zero capacitances and resistances, and *wlm*, which completes the net by inserting capacitances and resistances derived from wire load models. This option is equivalent to reading the parasitics file and then using the **complete_net_parasitics -complete_with** command.

Note: The **complete_net_parasitics** and **read_parasitics -complete_with** commands complete a net only if all missing segments are between two pins and the nets are partially annotated (nets are not affected if they are fully annotated or have no annotation at all). The net must also be hierarchical, so that if the parasitics for the block-level parts of a net are missing,

those parasitics could exist in the top-level net. If any of these conditions are not met, you must correct the SPEF or DSPF file manually.

-lumped_cap_only

This option does not apply to the SBPF format.

Indicates that only the total capacitance of nets is to be annotated as a lumped capacitance on the annotated nets. The RC networks specified in the parasitics file are discarded. The annotated lumped capacitance is the capacitance specified when the net is declared in the parasitics file.

-keep_capacitive_coupling

Indicates that the cross capacitors are to be kept in the RC networks data structure. This facilitates the capacitive crosstalk analysis, but does not turn it on. This option disables the *-coupling_reduction_factor* option. The command fails if both options are specified. All coupling capacitors are split to ground with a factor of 1.0 if crosstalk analysis is not activated. This option applies to both the SPEF and the SBPF format and requires a PrimeTime SI license.

-pin_cap_included

Indicates that the RC networks are to include the pin capacitances. By default, the RC network does not include pin capacitances. This option does not apply to the RSPF format. The RC pi model in RSPF format has to always include effect of pin capacitances.

-increment

Indicates not to overwrite previously annotated parasitics that are on the nets listed in the parasitics. Additionally, any incomplete annotations in the parasitics file are not to be rejected. By default, the RC annotation specified in the parasitics file overwrites the previous parasitics annotations of the nets listed in the parasitics file. Use this option for annotating hierarchical parasitics files.

-path *list*

Specifies a list of relative paths from the current design to the hierarchical instance names for which the parasitics file has been created. By default, absolute path names are used. Use this option if the parasitics file refers to an object (for example, *net*) in a hierarchy (for example, *hier*). Do not use this option if the parasitics file refers to an absolute path (for example, *hier/net*). If more than one prefix is supplied, all of the instances of the design can be annotated simultaneously.

-coupling_reduction_factor *factor*

This option applies only to the SPEF format and the SBPF format. A positive floating point number that specifies the factor to apply when reducing coupling capacitances to grounded capacitances. The default value is 1.0. This option is disabled if the *-keep_capacitive_coupling* option is specified. The command fails if both options are specified.

-triplet_type *ttype*

This option applies only to the SPEF and PARA formats. Several values in SPEF and PARA, such as capacitor and resistor values, can be specified as triplets *-min:typ:max*. By default, PrimeTime takes the max value. Using this option, can select the min or typ value. Allowed values are *max* (the default), *typ*, and *min*.

-verbose

Indicates that the **report_annotated_parasitics** report is to be generated when the parasitics file has been read. By default, after reading the parasitics file, the **report_annotated_parasitics** command is not executed.

-syntax_only

Indicates that the **read_parasitics** command is to parse the file for syntax errors without performing any parasitic annotation. Use this option to troubleshoot your parasitics file and avoid generating error messages during the actual annotation. No design is required to use the **-syntax_only** option.

-ilm_context

Indicates that the annotation is being performed in the presence of Interface Logic Models (ILMs). An original design parasitics can be used to annotate a design with ILMs using this option. This option does not issue error messages for missing nets, cells, and pins.

-eco

Indicates that the files being currently annotated are ECO parasitics from StarRC. PrimeTime SI can read ECO parasitics that are written out only by StarRC. The ECO parasitics can be annotated only when there are some existing parasitics that are already annotated. ECO parasitic files contain re-extracted parasitics for only the ECO nets and their immediate coupling neighbors and do not contain all the nets of the design. Incremental analysis can be performed after reading ECO parasitics.

-original_file_name *orig_file_name*

This option can only be used when the **-eco** option is being used. If the original annotation is performed through multiple parasitic files into PrimeTime SI, the ECO parasitic file corresponds to one of the original files (because it corresponds to one extracted database in StarRC). PrimeTime SI attempts to determine the corresponding original file, but it is not always possible. You can use this option to specify which original parasitic file the ECO file correspond to.

file_names

When the format is one of SPEF, DSPF, RSPF, and SBPF, it specifies a list of files from which parasitics information is to be read.

-keep_variations

Indicates that the statistical parasitic information are to be kept in the RC networks data structure. This facilitates the variation aware timing analysis, but does not turn it on. This option applies only to SBPF and SPEF formats and requires a PrimeTime VX license.

-create_default_variations

Specifies that default parasitic variations should be created for all the variation parameters. The default variations created are all assumed to be of normal distribution. The mean and sigma values are already present in the parasitic file.

DESCRIPTION

The **read_parasitics** command reads parasitics information from a disk file and

annotates it on nets of the current design. This command supports the SPEF, DSPF, RSPF, and Synopsys Binary Parasitics (SBPF) formats. For SPEF, RSPF, and DSPF, the file can be a simple ASCII file, or it can be compressed with gzip. The **read_parasitics** command automatically detects this format information from the file, but you can specify the base format using the *-format* option.

SBPF is an efficient format for sharing parasitics among Synopsys applications. You can use the PrimeTime **write_parasitics** command to write a parasitics file in SBPF.

You can specify parasitics in either reduced or detailed form. Reduced parasitics consist of an RC pi model specified for each driver pin of a net. An RC pi model contains two capacitances and one resistance. The first capacitance connects to the net driver pin; the resistance connects to the net driver pin and to a subnode to which the second capacitance connects. Both capacitances connect to the ground. The parasitics file, using the reduced form, also specifies the pin-to-pin net delays. The RC pi model is used to compute the effective capacitance of the nets at each driver of the net. The effective capacitance determines the cell delays and output slews. You can specify reduced parasitics with SPEF, DSPF, or RSPF formats.

Detailed parasitics consist of a network of resistances and capacitances for each net specified in the parasitics file. The capacitances must be with respect to the ground. Coupling capacitances are not supported unless the *-keep_capacitive_coupling* option is specified and you are using the SPEF or SBPF format. Resistances connected to the ground are not connected. The detailed RC network must be complete; incomplete RC networks are discarded and the delays are computed using wire-load models. You can specify detailed parasitics with the SPEF, SBPF, DSPF, or RSPF formats. Variation-aware parasitics are not supported unless the *-keep_variations* option is specified and SBPF or SPEF format is used.

The **read_parasitics** command does not overwrite lumped parasitics (set using the **set_load** or **set_resistance** command) if they already exist on the design. Instead a warning is issued. If the lumped parasitics are subsequently removed (using the **remove_capacitance** and **remove_resistance** commands) PrimeTime reverts to the reduced or detailed parasitics.

Incremental annotations are supported; if a small number of nets are affected, a full timing update is not required if the design is already timed. The maximum number of affected nets that avoid a full timing update depends on the total design size.

Multiple resistances and capacitances between the same nodes are also accepted. The values of multiple capacitances between a node are added together and multiple resistances between the same two nodes are kept separately and are considered during the delay calculation.

Normally, it is most efficient to read the top level last to minimize the amount of stitching which must be performed. However, when node coordinates are loaded by setting `read_parasitics_load_locations` to true, it is more efficient to read the top level first so that block coordinate systems can be transformed as they are read in.

Net and instance pin names in the design must match instance names in the parasitics file. For example, if you create the parasitics file from a design using VHDL naming conventions, the design name must use VHDL naming conventions.

To remove parasitics that were read and annotated with the **read_parasitics** command,

use the **remove_annotated_parasitics** command. The **reset_design** command removes all attributes from the current design, including annotated parasitics.

Annotated parasitics are used to compute cell delays and net delays with more accuracy. Because the delay calculation involves the computation of the actual waveforms of the transitions, you must provide PrimeTime with a way to interpret the delays from the Synopsys library. The definitions of the delays and transition times is communicated to PrimeTime with environment variables, which specify the different characterization trip-points used when generating the Synopsys library. For example, these trip-points can specify that for a rising cell delay, the delay is defined as the delay between the time that the input transition reaches 50 percent of the voltage source, and the time that the output transition reaches 50 percent of the voltage source. Similarly, the trip-points can specify that a rising transition time is defined as the delay between the time that the transition reaches 20 percent of the voltage source and the time that the transition reaches 80 percent of the voltage source. You specify these trip-points with the following variables:

Variable Name	Default
rc_slew_lower_threshold_pct_rise	20.0
rc_slew_lower_threshold_pct_fall	20.0
rc_slew_upper_threshold_pct_rise	80.0
rc_slew_upper_threshold_pct_fall	80.0
rc_input_threshold_pct_fall	50.0
rc_input_threshold_pct_rise	50.0
rc_output_threshold_pct_fall	50.0
rc_output_threshold_pct_rise	50.0

You can turn on the capacitive crosstalk effect on the delay by setting the **si_enable_analysis** variable to *true*. You can filter some of the coupling capacitances by specifying the following variables:

Variable Name	Default
si_filter_total_aggr_xcap	0.0
si_filter_total_aggr_xcap_to_gcap_ratio	0.0
si_filter_per_aggr_xcap	0.0
si_filter_per_aggr_xcap_to_gcap_ratio	0.0
si_filter_per_aggr_to_average_aggr_xcap_ratio	0.0
si_filter_single_xcap_to_gcap_ratio	0.0
si_filter_per_aggr_noise_peak_ratio	0.01

In PrimeTime, the delay calculation algorithms impose no limit on the number of resistances and capacitances that can be annotated onto a single network; however, there are two shell variables you can manipulate to track and filter annotations that might cause unexpectedly large runtimes. You use the **parasitics_warning_net_size** variable to specify the threshold number of nodes for

which the PARA-003 message is to be issued, indicating that a large amount of annotation has been encountered for a given network. Similarly, you use the **parasitics_rejection_net_size** variable to specify when such detailed annotation is to be rejected in favor of lumped annotation, issuing the PARA-004 warning message.

PrimeTime can also load locations information from parasitic files. This capability is controlled by the **read_parasitics_load_locations** variable. By default, this variable is *false*, and no locations are read into PrimeTime. You can set this variable to *true* to load locations.

When reading PARA format the parasitics may not be the same as what is specified by the *triplet_type* option. This happens when the specified type does not exist. In that case, the **read_parasitics** command chooses something according to the following table:

triplet_type		Parasitics stored as		
		typ	min-max	min-max-typ
<i>min</i>	/	typ	min	min
<i>typ</i>	/	typ	max	typ
<i>max</i>	/	typ	max	max

EXAMPLES

The following example reads the parasitics file *adder.spef* from disk and uses it to annotate the RC parasitics on the current design. The parasitics file contains a description of the RC network for nets of a design.

```
pt_shell> read_parasitics adder.spef
```

The following example illustrates annotating hierarchical parasitics files. The second command reads parasitics information, for instance *u1* of design *mult16* from the disk file *mult16_u1.spef*. The parasitics are annotated on the design, *MY_DESIGN*. The **-verbose** option enables the automatic call to **report_annotated_parasitics** command, which would report RC networks of the boundary nets of instance *u1* that are not complete. Pin capacitances are included in the annotated parasitics, as specified by the **-pin_cap_included** option. The second command reads another parasitics file in incremental mode, so that parasitics annotated on the boundary nets of instance *u1* are not discarded.

```
pt_shell> current_design MY_DESIGN
pt_shell> read_parasitics -pin_cap_included -increment -path u1 mult16_u1.spef
pt_shell> read_parasitics -pin_cap_included -increment -verbose mydesign.spef
```

The following is another example of hierarchical back annotation. The design, *top_level*, contains two instances, *inst0* and *inst1*, of the design *mydesign*.

```
pt_shell> read_parasitics top_level.spf -increment
```

```
pt_shell> read_parasitics mydesign.spf -increment -path inst0  
pt_shell> read_parasitics mydesign.spf -increment -path inst1
```

The following example removes annotated parasitics information from the current design.

```
pt_shell> remove_annotated_parasitics
```

The following example loads locations into PrimeTime memory.

```
pt_shell> set read_parasitics_load_locations true  
pt_shell> read_parasitics adder.spef
```

SEE ALSO

```
complete_net_parasitics(2)  
remove_annotated_parasitics(2)  
report_annotated_parasitics(2)  
reset_design(2)  
write_parasitics(2)  
si_enable_analysis(3)  
si_filter_total_aggr_xcap(3)  
si_filter_total_aggr_xcap_to_gcap_ratio(3)  
si_filter_per_aggr_xcap(3)  
si_filter_per_aggr_xcap_to_gcap_ratio(3)  
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)  
si_filter_per_aggr_noise_peak_ratio(3)  
rc_input_threshold_pct_fall(3)  
rc_input_threshold_pct_rise(3)  
rc_output_threshold_pct_fall(3)  
rc_output_threshold_pct_rise(3)  
rc_slew_lower_threshold_pct_fall(3)  
rc_slew_lower_threshold_pct_rise(3)  
rc_slew_upper_threshold_pct_fall(3)  
rc_slew_upper_threshold_pct_rise(3)  
read_parasitics_load_locations(3)  
PARAMETER-003 (n)  
PARAMETER-004 (n)
```

read_saif

Reads a SAIF file and annotates switching activity information on nets, pins, ports, and cells in the current design.

SYNTAX

```
int read_saif file_name
    [-strip_path prefix]
    [-path prefix]
    [-ignore ignore_name]
    [-exclude exclude_file_name]
    [-derate_glitch value]
    [-quiet]

string file_name
string prefix
string ignore_name
string exclude_file_name
float value
```

ARGUMENTS

file_name

Specifies the name of the SAIF file to read. If file name ends with .gz, .Z, or .bz2, it is read as a gzipped file, a compressed file, or a bzip2ed file, respectively. Otherwise, it is assumed that the file is uncompressed.

-strip_path *prefix*

The **read_saif** command assumes that the current design is instantiated as an instance in the testbench used to generate the SAIF file. The current design, therefore, appears as an instance in the SAIF file. The **-strip_path** argument specifies the name of the instance of the current design as it appears in the SAIF file.

The **read_saif** command annotates all subinstances in the hierarchy of the specified instance and annotates the instance itself. Enter each instance name completely, without any trailing hierarchy separator (/). For example, if the current design is instantiated in the simulation environment as TOP/DUT/U1, use **-strip_path** TOP/DUT/U1 and not **-strip_path** TOP/DUT/U.

-path *prefix*

Specifies a relative path from the current design to the hierarchical low-level design for which the SAIF file has been created. By default, absolute path names are used. Use this option if the SAIF file refers to an object in a hierarchy. If you want to read the switching information for a portion of the design, you can also use this option. For example, if the SAIF was generated for the whole design and you want to put only switching information on the nets/ports on the boundary and hierarchically under instance add14, then use the **-path** add14 option.

-ignore *ignore_name*

Specifies the name of an instance in the SAIF file for which switching activity is to be ignored. The **read_saif** command ignores switching activity

within the hierarchy of that instance in the SAIF file. *ignore_name* can be a hierarchical name separated by a slash (/). **-strip_path** is applied first. The **read_saif** command then strips off the prefix of a net name that matches *prefix* before deciding whether this net name is under the hierarchy specified by the **-ignore** option.

-exclude *exclude_file_name*

Specifies the name of a file that contains a list of names to be ignored. *exclude_file_name* is used when you want to ignore switching activity for several instances. The file must contain each *ignore_name* on a separate line, without the **-exclude** option. As with the **-ignore** option, the ignore names are recognized after the **-strip_path** argument.

-derate_glitch *value*

Specifies a default derating factor value to be used for inertial glitches in the SAIF file. If **-derate_glitch** is not specified, a default derating factor of 0.5 is used.

-quiet

Specifies quiet mode and for instance suppresses warnings on design objects in the SAIF file that could not be annotated on the design.

DESCRIPTION

The **read_saif** command reads a SAIF file and annotates the switching activity attributes **toggle_rate** and **static_probability** for nets, ports, and pins of the current design. The **update_power** command uses this information to calculate dynamic power values. If a particular object in the SAIF file cannot be found in the current design, the object is ignored and a warning message is provided. The **read_saif** command returns 1 if at least one of the objects in the file is successfully annotated. Otherwise, it returns 0.

To identify the instance (and corresponding sub-instances) you want to annotate, use the **-strip_path** option. To annotate switching information about a particular instance, use the **-path** option. To specify a default derating factor other than 0.5, use the **-derate_glitch** option.

If a SAIF file comes from RTL simulation, **read_saif** tries to find the name mapping between the objects in the SAIF file and in the gate level netlist automatically. The name mapping rules can be set by the **set_rtl_to_gate_name** command. During the average power calculation, **update_power** also automatically propagates the switching activity for those unannotated nets.

EXAMPLES

The following example annotates switching activity for the current design in the **pt_shell**, for a design instantiated as **Test/U1** in the simulation testbench.

```
pt_shell> read_saif file -strip_path Test/U1
```

The following examples annotate the instance **U10** under current design.

```
pt_shell> read_saif -file -strip_path Test/U1 -path U10
```

The following examples demonstrate annotation of multiple designs; the design "name1" is instantiated as Test1/U1, and design "name2" is instantiated as as Test1/U2.

```
pt_shell> current_design name1
pt_shell> read_saif file_1 -strip_path Test1/U1
pt_shell> current_design name2
pt_shell> read_saif file_2 -strip_path Test1/U2
```

SEE ALSO

```
set_switching_activity(2), set_rtl_to_gate_name(2), get_switching_activity(2),  
reset_switching_activity(2), report_switching_activity(2), report_power(2),  
update_power(2);
```


read_sdc

Reads in a script in Synopsys Design Constraints (SDC) format.

SYNTAX

```
int read_sdc file_name [-echo] [-syntax_only] [-version sdc_version]  
string file_name  
string sdc_version
```

ARGUMENTS

-echo

Indicates that each constraint is to be echoed as it is executed.

-syntax_only

Indicates that SDC script is processed only to check the syntax and semantics of the script.

-version *sdc_version*

Specifies the version of SDC to which the file conforms. Allowed values are *1.0*, *1.1*, *1.2*, *1.3*, *1.4*, *1.5*, *1.6*, *1.7* and *latest* (the default).

file_name

Specifies the name of the file that contains the SDC script to be read.

DESCRIPTION

The **read_sdc** command reads in a script file in Synopsys Design Constraints (SDC) format, which contains commands for use by PrimeTime or by Design Compiler in its Tcl mode. SDC is also licensed by external vendors through the Tap-in program. SDC formatted script files are Tcl scripts that use a subset of the commands supported by PrimeTime and Design Compiler.

By default, **read_sdc** executes the constraints as they are read. If there are errors part way through the script, it is possible that some constraints are applied, and some are not. You can use the **-syntax_only** option to simply check the script without applying any constraints. This also verifies conformance to the specified SDC version. If there are any errors during the checking phase, then the result of **read_sdc** is 0.

Like **source**, the **read_sdc** command is sensitive to variables that control script execution (for example, **sh_continue_on_error** and **sh_script_stop_severity**). The **read_sdc** command uses **sh_source_uses_search_path** to determine if **search_path** is used to find the script.

read_sdc supports several file formats. The file can be a simple ascii script file, an ascii script file compressed with gzip, or a Tcl bytecode script, created by TclPro Compiler.

Note that SDC does not allow command abbreviation. Also note that **exit** and **quit** do not cause the application to exit, but they do cause the reading of the SDC file to

stop.

The latest SDC Version supports the following commands. Those added for versions 1.3 and later are noted. Some constraints which PrimeTime ignores are not listed.

General Purpose Commands:

list
expr
set

Object Access Functions:

all_clocks
all_inputs
all_outputs
current_design
current_instance
get_cells
get_clocks
get_libs
get_lib_cells
get_lib_pins
get_nets
get_pins
get_ports
set_hierarchy_separator

Basic Timing Assertions:

create_clock
create_generated_clock (1.3)
set_clock_gating_check
set_clock_latency
set_clock_transition
set_clock_uncertainty
set_data_check (1.4)
set_false_path
set_input_delay
set_max_delay
set_min_delay
set_multicycle_path
set_output_delay
set_propagated_clock

Secondary Assertions:

set_disable_timing
set_max_time_borrow
set_timing_derate (1.5)

Environment Assertions:

set_case_analysis

read_sdc

```

set_drive
set_driving_cell
set_fanout_load
set_input_transition
set_load
set_logic_zero
set_logic_one
set_logic_dc
set_max_area
set_max_capacitance
set_max_fanout
set_max_transition
set_min_capacitance
set_min_fanout
set_operating_conditions
set_port_fanout_number
set_resistance
set_wire_load_min_block_size
set_wire_load_mode
set_wire_load_model
set_wire_load_selection_group

```

For a complete guide to using SDC with Synopsys applications, see the *Using the Synopsys Design Constraints Format Application Note* in Synopsys Online Documentation (SOLD), which is included with the software, or from Documentation on the Web, which is available through SolvNET at <http://solvnet.synopsys.com>.

The usage of some of these commands is restricted when reading SDC. In some cases, some command options are not allowed. In some applications, some of the commands are not supported. For example, PrimeTime does not support the **set_logic*** commands. These commands are ignored when loaded in with **read_sdc**, with a message (for an example, see the EXAMPLES section). If a command not supported by SDC is found by **read_sdc**, it appears as an unknown command, with a message. The same condition exists for command options. If an option is not supported by SDC, a message is issued indicating that condition. However, if the option is unknown, a different message is issued.

Note that although **create_generated_clock** is supported (beginning with version 1.3), there is no provision for the **get_generated_clocks** shortcut which is available in PrimeTime and Design Compiler. Generated clocks are simply clocks with some additional attributes and in SDC, they are retrieved through the **get_clocks** command.

The following features are added for SDC Version 1.5 or later: **set_clock_latency -clock** option, **set_timing_derate** command with all options, **set_max_transition -clock_path -data_path** -rise -fall options, **set_clock_uncertainty -rise/fall_from/to** options, **set_operating_conditions** -object_list option, synonyms for all **get_object** commands, **get_objects -nocase** support independently with -regexp, **get_objects -regexp** support, **get_objects -of_objects** support for **get_cells/get_nets/get_pins**.

EXAMPLES

The following example reads the SDC script top.sdc.

```
prompt> read_sdc top.sdc -version 1.2
```

```
Reading SDC version 1.2...
```

```
1
```

```
prompt>
```

The following example reads the SDC script top2.sdc, The script contains commands not supported by SDC, and CMD-005 messages are generated for those commands.

```
prompt> read_sdc -syntax_only top2.pt -version 1.2
```

```
Checking syntax/semantics using SDC version 1.2...
```

```
Error: Unknown command 'link_design' (CMD-005)
```

```
** Completed syntax/semantic check with 1 error(s) **
```

```
0
```

```
prompt>
```

The following example shows the result when an unsupported command is in an SDC file. One SDC message is generated per instance of an unsupported command. At the end of the read, a summary of all unsupported commands in the file is generated. This SDC file was read into PrimeTime.

```
pt_shell> read_sdc t2.sdc -version 1.2
```

```
Reading SDC version 1.2...
```

```
Warning: Constraint 'set_logic_zero' is not supported by pt_shell. (SDC-3)
```

```
Warning: Constraint 'set_logic_zero' is not supported by pt_shell. (SDC-3)
```

```
Warning: Constraint 'set_logic_one' is not supported by pt_shell. (SDC-3)
```

```
Summary of unsupported constraints:
```

```
Information: Ignored 1 unsupported 'set_logic_one' constraint. (SDC-4)
```

```
Information: Ignored 2 unsupported 'set_logic_zero' constraints. (SDC-4)
```

```
1
```

```
pt_shell>
```

The following example is a script that is not SDC-compliant, because it contains unsupported commands or command options.

```
current_design TOP
```

```
set_resistance -value 12.2 [get_nets i1t2]
```

The following example shows the output generated by **read_sdc** when reading the previous script. In SDC, **current_design** can be used only to get the current design;

```
read_sdc
```

```
396
```

no arguments are allowed. Also, there is no **-value** option for **set_resistance**, so an appropriate message is generated.

```
prompt> read_sdc t3.tcl -echo -version 1.2
Reading SDC version 1.2...
current_design TOP
Error: extra positional option 'TOP' (CMD-012)
set_resistance -value 12.2 [get_nets i1t2]
Error: unknown option '-value' (CMD-010)
0
prompt>
```

The following example shows the message generated when an option is supported by the command but not by SDC.

Information: The '-value' option for set_resistance is unsupported. (CMD-038)

SEE ALSO

source (2), **write_sdc** (2), **search_path** (3), **sh_continue_on_error** (3),
sh_script_stop_severity (3), **sh_source_uses_search_path** (3).

read_sdf

Reads leaf cell and net timing information from a file in Standard Delay Format (SDF) and uses that information to annotate the current design.

SYNTAX

```
string read_sdf [-load_delay net | cell]
[-analysis_type single | bc_wc | on_chip_variation]
[-min_file min_fname]
[-max_file max_fname]
[-path path_name]
[-type sdf_min | sdf_typ | sdf_max]
[-min_type sdf_min | sdf_typ | sdf_max]
[-max_type sdf_min | sdf_typ | sdf_max]
[-cond_use min | max | min_max] [-syntax_only]
[-strip_path strip_path_name]
[-verbose] [-worst ]
file_name
```

```
string path_name
string sdf_file_name
string min_sdf_file_name
string max_sdf_file_name
string strip_path_name
```

ARGUMENTS

-load_delay net | cell

Indicates whether load delays are included in net delays or in cell delays in the timing file being read. The default is *cell*. The load delay is the portion of cell delay arising from the capacitive load of the net driven by the cell.

-analysis_type single | bc_wc | on_chip_variation

Use this option only if you have not already set an analysis type with **set_operating_conditions -analysis_type**. If you are in min_max mode, the default is *bc_wc*. *single* indicates that only one operating condition is to be used. Specifying either *bc_wc* or *on_chip_variation* switches to min_max mode and causes both minimum and maximum delays to be read from the SDF file. Delays in SDF are represented in the form of triplets (sdf_min:sdf_typ:sdf_max). By default, the **-analysis_type bc_wc | on_chip_variation** option reads the *sdf_min* and *sdf_max* delays, respectively. To change this, use the **-min_type** and **-max_type** options.

-min_file *min_sdf_file_name*

Use this option only if the minimum and maximum delays are in two separate SDF files. Specifies the file from which minimum delay timing information is to be read. The timing file must be in SDF format version v1.0, v2.0, v2.1 or v3.0.

-max_file *max_sdf_file_name*

Use this option only if the minimum and maximum delays are in two separate

SDF files. Specifies the file from which maximum delay timing information is to be read. The timing file must be in SDF format version v1.0, v2.0, v2.1 or v3.0.

-path *path_name*

Specifies the path from the current design to the subdesign for which the timing file has been created.

-type *sdf_min | sdf_typ | sdf_max*

Indicates which of the SDF triplet delay values are to be read from the SDF file. Delays in SDF are represented in the form of triplets (*sdf_min:sdf_typ:sdf_max*). By default, **read_sdf** reads the maximum delays *sdf_max*.

Note: If you use **-type** while in min/max mode (for example, if you use **-operating_conditions bc_bw | on_chip_variation**), a single value is annotated onto both min and max values of an arc.

-min_type *sdf_min | sdf_typ | sdf_max*

Specifies which of the SDF triplet delay values are to be read from the SDF file for minimum delay. Delays in SDF are represented in the form of triplets (*sdf_min:sdf_typ:sdf_max*). By default, **read_sdf** reads the minimum delays *sdf_min*. Use this option only with option **-analysis_type bc_wc | on_chip_variation**.

-max_type *sdf_min | sdf_typ | sdf_max*

Specifies which of the SDF triplet delay values are to be read from the SDF file for maximum delay. Delays in SDF are represented in the form of triplets (*sdf_min:sdf_typ:sdf_max*). By default, **read_sdf** reads the maximum delays *sdf_max*. Use this option only with option **-analysis_type bc_wc | on_chip_variation**.

-cond_use *min | max | min_max*

Use this option only if the SDF file includes some conditional delays using the SDF construct COND, and if the Synopsys library in use does not specify conditional delays. *min* indicates that the minimum of all conditional delays is to be used to annotate the corresponding timing arc. *max* indicates to use the maximum; *min_max* indicates min_max operating conditions; the minimum of all conditional delays is to be used for the minimum operating condition, and the maximum of all conditional delays is to be used for the maximum operating condition. You cannot use *min_max* with a single operating condition; you must be in min_max mode.

-syntax_only

Indicates that no timing annotation is to be performed; syntax only is to be processed. Use this option to verify that your SDF syntax is correct and will not issue any error messages.

-strip_path *strip_path_name*

Specifies a prefix path that is to be stripped from all SDF objects. Such a prefix path is usually a result of generating an SDF file for a subdesign, and using this subdesign as the current design.

-verbose

Use this option to enable execution of **report_annotated_delay** and **report_annotated_check** after reading SDF.

-worst

Indicates that **read_sdf** is to annotate the current design only with delays worse than the current annotated delays; applies to annotated net and cell delays and annotated timing checks. The worst delay is defined as the most pessimistic delay. This means primetime annotates the min of minima, and max of maxima values.

sdf_file_name

Specifies the file from which timing information is to be read. The timing file must be in SDF format version v1.0, v2.0, v2.1 or v3.0.

DESCRIPTION

The **read_sdf** command reads from a disk file leaf cell and net timing information and uses it to annotate the current design. The timing file must be in SDF format v1.0, v2.0, v2.1 or v3.0. The file can be simple ascii, or it can be compressed with gzip. Instance-specific pin-to-pin cell and net delays are read from the SDF file and annotated on the current design. When you specify **-path**, **read_sdf** annotates the current design with information from a timing file created for an instance of a subdesign of the current design. When you specify a subdesign, you cannot use the net delays to the ports of the subdesign to annotate the current design.

The load delay, also known as extra source gate delay, is that portion of the cell delay caused by the capacitive load of the driven net. Some delay calculators consider the load delay part of the net delay. Others consider the load delay part of the cell delay. By default, **read_sdf** assumes the load delay is included in the cell delay in the timing file being read. The **-load_delay** option allows you to inform **read_sdf** if your timing file includes the load delay in the net delay rather than in the cell delay.

Setup and hold, recovery, removal, period, width, nochange and skew timing checks, when present in the timing file, are used to annotate the current design. Prior to SDF v3.0 the SDF construct for removal is 'hold'.

Instance names in the design must match instance names in the timing file. For example, if the timing file is created from a design using VHDL naming conventions, the design must use VHDL naming conventions.

To remove delays read and annotated with **read_sdf**, use **reset_design** or **remove_annotated_delay**. To remove timing checks annotated with **read_sdf**, use **remove_annotated_check**.

Only partial support for SDF v3.0 is available. Supported v3.0 constructs are: REMOVAL, RECREM, RETAIN, and CONDELSE.

Note that **read_sdf** does not support hierarchical SDF annotation. All pins must be specified at the leaf cell level. It is not possible to specify hierarchical pins.

When multiple annotations against the same arc are present in the annotation file, the last annotation will take precedence. If the **-worst** option is specified, each annotation will be applied only if it is worse than the existing timing value for that arc.

Also note that if the variable **timing_use_zero_slew_for_annotated_arcs** is set to its

default value *auto*, and at least 95% of delay arcs are annotated, the pure SDF flow is switched on automatically. This will result in faster performance of *update_timing*, at the expense of no longer calculating the slews on the load pins of annotated arcs. See man page of ***timing_use_zero_slew_for_annotated_arcs*** for more details.

EXAMPLES

The following example reads the timing file "adder.sdf" from the disk and uses it to annotate the timing on the current design. The timing file contains load delays included in the cell delays.

```
pt_shell> read_sdf -load_delay cell adder.sdf
```

The following example reads timing information for instance u1 of design MULT16 from the disk file "mult16_u1.sdf", which contains the timing for the instance u1 of design MULT16. The timing is annotated on the design MY_DESIGN. In this case, load delay is included in the net delays.

```
pt_shell> current_design MY_DESIGN
pt_shell> read_sdf -load_delay net -path u1 mult16_u1.sdf
```

The following example reads timing information and annotates the current design with the max timing when the timing file has different timing conditions for the same pin pair. The load delay is assumed to be included in the cell delay in this example.

```
pt_shell> remove_annotated_delay
pt_shell> read_sdf -cond_use max foo.sdf
```

The following example reads min and max timing information and annotates the current design with delays corresponding to minimum and maximum operating conditions, respectively. When reporting minimum delay, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delay, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type bc_wc foo.sdf
```

The following example reads min and max timing information from two separate SDF files and annotates the current design with delays corresponding to minimum and maximum operating conditions, respectively. When reporting minimum delay, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delay, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type bc_wc -min_file foo_min.sdf -
max_file foo_max.sdf
```

SEE ALSO

remove_annotated_check (2), ***remove_annotated_delay*** (2), ***report_annotated_check*** (2), ***report_annotated_delay*** (2), ***reset_design*** (2), ***set_annotated_check*** (2), ***set_annotated_delay*** (2), ***write_sdf*** (2), ***timing_use_zero_slew_for_annotated_arcs*** (3).

read_vcd

Specifies the switching activity information generated by simulation for use in power calculation. Internally, non-VCD format switching activity is converted to VCD.

SYNTAX

```
int read_vcd
[-path prefix]
[-strip_path prefix]
[-zero_delay]
[-pipe_exec command]
[-format format]
[-time window_list]
[-rtl]
file_name
```

Data Types

<i>prefix</i>	string
<i>file_name</i>	string
<i>command</i>	string
<i>format</i>	string
<i>window_list</i>	list

ARGUMENTS

-path *prefix*

Specifies a relative path from the current design to the hierarchical low-level design for which the VCD file has been created. By default, absolute path names are used. Use this option if the VCD file refers to an object in a hierarchy. Do not use this option if the VCD file refers to an absolute path.

-strip_path *strip*

Specifies a path prefix that is to be stripped from all the object names read from the VCD file. This option is applied to strip the testbench/instance path from the VCD file.

-zero_delay

Specifies the VCD file comes from a zero delay simulation.

-pipe_exec *command*

Specifies a shell command that is used to generate the VCD file *file_name*. This option invokes the command and directly pipe the output VCD file to PrimeTime PX. In other words, the simulation and power analysis are in parallel run. No VCD disk file is generated at all.

-format

Specifies the language format for names in the VCD. Available options are **Verilog** , **VHDL** and **SystemVerilog**. The default format is **Verilog**.

-time window_list

Specifies time window(s) in which the activities are counted for power calculation. The option accepts an even number list of float values in increasing order. For example, **-time {10 20 50 70}** specifies the time window [10ns, 20ns] and [50ns, 70ns]. If you do not know the end of simulation time, you can use a negative number in the **-time** option to indicate the end of the simulation. For example, **-time {10 -1}** specifies the activity time window from 10 ns to the end of simulation time in the VCD file. The default activity time window is the whole simulation time in VCD file. This option applies to averaged, time_based power analysis and statistical leakage power variation analysis. The time will be adjusted by the tool automatically according to the timescale in the VCD file or the **-interval** value in **create_power_waveforms**. For example, if the **-interval** is 10, then **-time {13 33}** will become **-time {10 30}**.

-rtl

Specifies the VCD file comes from a rtl simulation simulation. Typically this means that most nets in the design will not be directly annotated from the VCD, because combinational nets are not included in rtl simulation. The name mapping commands (**set_rtl_to_gate_name**) are only honored if this option is set. Also, in the **time_based** power analysis mode, event propagation will be done during power analysis only if the **-rtl** option is set when the **read_vcd** command was issued.

file_name

Specifies the switching activity file name to be read. If the **file_name** option ends with the extension .gz, .Z, .bz or .bz2, .vpd or .fsdb, it is read as a gzipped file, a compressed file, a bzip2ed file, a VPD (VCD Plus) file, and a FSDB file, respectively, otherwise, it is assumed to be in the VCD format.

DESCRIPTION

The **read_vcd** command specifies the switching activity file generated by simulation to be used for time_based power calculation.

To read in a gzipped VCD file, a compressed VCD file, a BZIP2 file, a VPD file, and an FSDB file, your PATH should include gunzip, uncompress, bunzip, vcs, and fsdb2vcd (a utility from FSDB inventor Novas <http://www.novas.com>). The VPD file is generated by VCS. So you should define the environment variable VCS_HOME to tell the tool where to find the VCS utility. If you need to invoke 64-bit VCS utility, you need to set the environment variable VCS_MODE_FLAG to 64. You can also define the environment variable NOVAS_HOME to tell the tool where to find the fsdb2vcd executable.

The **-pipe_exec** option allows PrimeTime PX to read in a VCD file directly from simulator, thus avoiding the need to generate a large VCD file. The VCD file name in this scenario serves as a pipe name. After simulation, the VCD file does not exist. There is no need to modify the existing test bench. The file name is treated like a normal VCD dump.(inserting \$dumpfile, \$dumpvars, etc.) Nor is there any need need to invoke the simulator. PrimeTime PX automatically runs the simulation and directly read in the VCD output from the simulator.

When a VCD file comes from a zero delay gate level simulation; most of events happen at clock edges. This leads to a very large un-realistic peak power. With **-zero_delay**

option, the power will be averaged over each clock cycle. This requires you to specify the SDC (Synopsys Design Constraint) file, at least clocks and transition time at primary inputs have to be defined.

If a VCD file comes from RTL simulation, the `-rtl` option should be used. In this case, `read_vcd` tries to find the name mapping between the objects in the VCD file and in the gate level netlist automatically. The name mapping rules can be set by the `set_rtl_to_gate_name` command. During the average power calculation, `update_power` also automatically propagates the switching activity for those unannotated nets.

Behavior For Different Power Modes

Because of the way analysis is performed in different power analysis modes (see the **power_analysis_mode** variable to set the mode), the **read_vcd** command has different behavior depending on the power analysis mode.

In the *averaged* and *leakage_variation* mode, power analysis is performed using toggle rates. In these modes the input activity file is converted to toggle rates and static probabilities, which are annotated onto the design.

In the *time_based* flow, power analysis is performed for individual events in the activity file. The **read_vcd** command is a required part of this flow. In this flow, the header of the activity file is read when the **read_vcd** command is issued, to determine what portion of the design is annotated. The remainder of the activity file is not read until power analysis is performed.

In the *time_based* flow, if the `-rtl` option is used, name mapping is used (see the **set_rtl_to_gate_name** command) and power analysis employs zero delay simulation to create events on unannotated nets. Without the `-rtl` option, name mapping is not used and no events is simulated. It is intended that RTL VCD be in the input if the option is used. In this case, sequential cells, primary inputs, and black box outputs are typically annotated, while combinational nets are not annotated. Events on these unannotated nets must be produced via simulation.

BACKWARD COMPATIBILITY MODE

In the B-2008.12 release, the options and functional capability of the **read_vcd** command changed. For this release, it is possible to use the prior functionality of the command, by setting the **power_ui_backward_compatibility** variable to **true**. In this backward compatibility mode, the **read_vcd** command takes one additional option, described below:

`[-cells cell_list]` (backward compatibility mode only)

`-cells cell_list`

Specifies the cell names or collections of cells for which power needs to be calculated. If you do not specify this option, PrimeTime PX calculates power for the whole design. In the B-2008.12 release, this option was moved to the **set_power_analysis_options** command.

EXAMPLES

The following example reads the VCD file `dump.vcd` from the disk and uses the first 100ns of the events in the file.

```
pt_shell> read_vcd dump.vcd -time {0 100}
```

The following example runs VCS and pipes the VCD file `dump.vcd` directly to PrimeTime PX:

```
pt_shell> read_vcd -pipe "vcs -R tb.v design.v" dump.vcd
```

SEE ALSO

```
read_verilog(2)  
read_vhdl(2)  
read_db(2)  
set_rtl_to_gate_name(2)  
power_analysis_mode(3)  
set_power_analysis_options(2)  
power_ui_backward_compatibility(3)
```

read_verilog

Reads in one or more Verilog files.

SYNTAX

```
string read_verilog [-hdl_compiler] file_names  
list file_names
```

ARGUMENTS

-hdl_compiler

Indicates that the Verilog files are to be read using the PrimeTime external reader (ptxr) that uses HDL Compiler. Reading files in this way requires an HDL Compiler license while the read is in progress. HDL Compiler supports the complete Verilog language, but uses more CPU and memory than does the native PrimeTime Verilog reader.

file_names

Specifies names of one or more files to be read.

DESCRIPTION

Reads one or more structural, gate-level Verilog netlists into PrimeTime. To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**. To determine the file that **read_verilog** loads, use the **which** command. After the Verilog files are loaded, to view the design objects, use the **list_designs** command. To remove designs, use the **remove_design** command.

The Verilog netlist must contain fully-mapped, structural designs. PrimeTime cannot link or perform timing analysis with netlists that are not fully mapped at the gate level. There must be no Verilog high-level constructs in the netlist.

By default, **read_verilog** invokes a native PrimeTime Verilog reader. For large netlists, this reader is much faster and more memory-efficient than HDL Compiler, but is limited to structural Verilog. The file can be a simple ascii file, or it can be compressed with gzip. The native Verilog reader automatically detects all of this format information from the file. Files which are compressed with gzip are first uncompressed to a temporary file in the directory stored in the variable **pt_tmp_dir**.

Generally, the native Verilog reader will not create unconnected nets. If you need to have these nets in your design, set the variable **svr_keep_unconnected_nets** to true.

If you have an HDL Compiler license and prefer to use HDL Compiler, you can invoke it from PrimeTime using **read_verilog -hdl_compiler**. When reading Verilog with a PrimeTime external reader (in this case, HDL Compiler), the procedure for interrupting the **read_verilog** command is slightly different than for most PrimeTime commands. Normally, one Control-c terminates a command, and three Control-c entries in sequence terminates PrimeTime. However, to terminate **read_verilog**, enter Control-

c three times. This terminates the read process but does not terminate PrimeTime. This is especially useful if you want to terminate the reading of a very long netlist file.

You can use the **bus_naming_style** variable to control bus naming, but setting this variable in PrimeTime does *not* affect the reading of Verilog files with HDL Compiler.

Limitations of the Native Verilog Reader in PrimeTime

The native Verilog reader in PrimeTime has the following limitations:

- No non-structural constructs are supported. For details, see the section entitled "Supported Language Subset for PrimeTime's Native Verilog Reader".
- Parameters are not supported. The parameter and defparam statements are read and ignored.
- Gate instantiations are not supported.

Supported Language Subset for the Native Verilog Reader in PrimeTime

The native Verilog reader supports a small structural subset of the Verilog language. Additional constructs are recognized and ignored. Supported constructs are as follows:

- module/endmodule
- input, inout, and output
- wire
- tri, wor, and wand: These constructs are supported, but are considered to be the same as the wire construct. That is, there is no special significance to these constructs.
- supply0 and supply1: These create wires that are logic 0 and 1, respectively.
- assign
- tran: An exception from the gate instantiation subset, tran is supported to the extent that it relates one wire to another, as with assign. Beyond that, tran has no special significance.

- The preprocessor directives ``define`, ``undef`, ``include`, ``ifdef`, ``else`, and ``endif` are not supported by default, as the target for this reader is structural, machine generated Verilog, which rarely contains these constructs. You can enable a preprocessor phase by setting the variable **svr_enable_vpp** to true. This will make an pre-pass over the file looking for and expanding these constructs, outputting a temporary file in the directory stored in the variable **pt_tmp_dir**.
- All simulation directives (for example, ``timescale`) are recognized and ignored.
- The `specify/endspecify` construct and its contents are recognized and ignored.
- Like HDL Compiler, the comment directives *translate_off* and *translate_on* are used to bypass Verilog features not supported by the reader. The following example shows how to bypass an unsupported feature:

```
// synopsys translate_off
nand (n2, a1, s2);
/* synopsys translate_on */
```

Limitations of the HDL Compiler

Reading with the **-hdl_compiler** option has these limitations:

- Any variables that affect Verilog reading are read from `.synopsys_dc.setup` files, not from `.synopsys_pt.setup`. These variables cannot be set from within PrimeTime. You can set the variable **ptxr_setup_file** in PrimeTime to restrict ptxr so that it reads only the system setup file and ignores your home and local setup files. For details, see the *PrimeTime User Guide: Fundamentals*.
- Some of the error messages that appear from the **read_verilog** process cannot be found using the **man** command from PrimeTime. You can access these message man pages from other Synopsys shells or from the UNIX `man` command.
- Pragmas are ignored.
- No warnings are issued if unmapped logic or HDL constructs exist in the Verilog netlist.

EXAMPLES

In the following example, the file `newcpu.v` is read based on the **search_path**.

```
pt_shell> set search_path "/designs/newcpu/v1.6 /libs/cmos"
/designs/newcpu/v1.6 /libs/cmos
```



```
pt_shell> read_verilog newcpu.v
Loading verilog file '/designs/newcpu/v1.6/newcpu.v'
1
pt_shell> remove_design -all
Removing design newcpu...

pt_shell> read_verilog newcpu.v -hdl_compiler
Beginning read_verilog...
Loading db file '/release/libraries/syn/standard.sldb'
Loading db file '/release/libraries/syn/gtech.db'
Loading verilog file '/designs/newcpu/v1.6/newcpu.v'
Reading in the Synopsys verilog primitives.
/designs/newcpu/v1.6/newcpu.v:
1
```

SEE ALSO

```
list_designs (2), remove_design (2), which (2); bus_naming_style (3), pt_tmp_dir
(3), ptxr_setup_file(3), search_path (3), svr_enable_vpp (3),
svr_keep_unconnected_nets (3).
```

read_vhdl

Reads in one or more VHDL files.

SYNTAX

```
Boolean read_vhdl [-vhdl_compiler] file_names  
list file_names
```

ARGUMENTS

-vhdl_compiler

Indicates that the VHDL files are to be read using VHDL Compiler. Reading files in this way requires a VHDL Compiler license while the read is in progress.

file_names

Specifies names of one or more files to be read.

DESCRIPTION

Reads one or more structural, gate-level VHDL netlists into PrimeTime using the PrimeTime external reader (ptxr). To locate files that have relative pathnames, the command uses the **search_path** variable and searches for each file in each directory listed in the **search_path**. Files that have absolute pathnames are loaded as though there were no **search_path**. To determine the file that **read_vhdl** loads, use the **which** command. After the VHDL files are loaded, to view the design objects, use the **list_designs** command. To remove designs, use the **remove_design** command.

The VHDL netlist must contain fully-mapped, structural designs. PrimeTime cannot link or perform timing analysis with netlists that are not fully mapped at the gate level. There must be no VHDL high-level constructs in the netlist.

By default, the **read_vhdl** command uses the same VHDL netlist reader as Design Compiler. This reader does not require any licenses. If you have a VHDL Compiler license and prefer to use VHDL Compiler, you can invoke it from PrimeTime using **read_vhdl -vhdl_compiler**.

The procedure for interrupting the **read_vhdl** command is slightly different than most PrimeTime commands. Normally, one Control-c terminates a command, and three Control-c entries in sequence terminates PrimeTime. However, to terminate **read_vhdl**, enter Control-c three times. This terminates the read process but does not terminate PrimeTime. This is especially useful if you want to terminate the reading of a very long netlist file.

Reading with the **read_vhdl** command has these limitations:

- Any variables that affect VHDL reading are read from .synopsys_dc.setup files, not from .synopsys_pt.setup. These variables cannot be set from within PrimeTime. You can set the variable **ptxr_setup_file** in PrimeTime to restrict ptxr so that it reads only the system setup file and ignores your home and local setup files. For

details, see the *PrimeTime User Guide: Fundamentals*.

- In order to support reading of entity/architecture pairs from separate files, all files in the *file_names* list are read at once. If the VHDL files you are reading are unrelated, you should read them in separate **read_vhdl** commands. This will avoid memory peaks in either ptxr or PrimeTime.

- Some of the error messages that appear from the **read_vhdl** process cannot be found using the **man** command from PrimeTime. You can access these message man pages from other Synopsys shells or from the UNIX man command.

- Pragmas are ignored.

- No warnings are issued if unmapped logic or HDL constructs exist in the VHDL netlist.

EXAMPLES

In the following example, the file newcpu.vhd is read based on the **search_path**.

```
pt_shell> set search_path "/designs/newcpu/v1.6 /libs/cmos"  
/designs/newcpu/v1.6 /libs/cmos
```

```
pt_shell> read_vhdl newcpu.vhd  
Beginning read_vhdl...  
Loading vhd1 file '/designs/newcpu/v1.6/newcpu.vhd'  
Performing 'read' command.  
New VHDL reader completed successfully.  
end  
1
```

SEE ALSO

```
list_designs(2), man(2), remove_design(2), which(2); ptxr_setup_file(3).  
search_path(3).
```

remote_execute

Builds a buffer of commands and triggers execution of these commands on remote slaves.

SYNTAX

```
boolean remote_execute  
[-pre_commands pre_command_string]  
command_string  
[-post_commands post_command_string]  
[-verbose]
```

Data Types

<i>pre_command_string</i>	string
<i>post_command_string</i>	string
<i>command_string</i>	string

ARGUMENTS

-pre_commands pre_command_string
A string of commands to be executed in the slave context before the execution of the default command string. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

command_string
A string of commands to be executed in the slave context. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

-post_commands post_command_string
A string of commands to be executed in the slave context after the execution of the default command string. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

-verbose
Specifies that the slave output should be piped back to the master console.

DESCRIPTION

The `remote_execute` command builds up a buffer of commands and triggers execution of these commands on remote slaves. All commands in the buffer are executed in the order in which they entered the buffer. This command can only be executed when the user has set the current session. The very first invocation of `remote_execute` for a particular session will trigger baseline and netlist image generation. Use curly braces to force variables and expressions to be evaluated in a slave context. Use quotes to force variables and expressions to be evaluated in a master context

The *-pre_command* and *-post_command* options can be used to place commands at the start and end of the command buffer.

EXAMPLES

In the examples below, a scenario named `s1` is in focus. In the following example, the **`remote_execute`** command is used to execute a nonmerged `report_timing`. The output of the **`report_timing`** command is found in the `$working_directory/s1/out.log` file.

```
pt_shell> remote_execute {report_timing}

Start of Master/Slave Task Processing
-----
Started   : Command execution in scenario 's1'
Command execution in scenario 's1' : Succeeded
Finished  : Command execution in scenario 's1'
-----
End of Master/Slave Task Processing

1
```

In this next example the situation is as in the first example except that the `-pre_commands` option is used to set a variable prior to the report.

```
pt_shell> remote_execute -pre_commands {set
rc_slew_lower_threshold_pct_fall 30.0} {report_timing}
1
Start of Master/Slave Task Processing
-----
Started   : Command execution in scenario 's1'
Command execution in scenario 's1' : Succeeded
Finished  : Command execution in scenario 's1'
-----
End of Master/Slave Task Processing

1
```

SEE ALSO

`report_multi_scenario_design(2)`

remove_annotated_check

Removes annotated timing checks from the design, either on specific cells, between specific pins, or on all cells in the current design.

SYNTAX

```
string remove_annotated_check
[-all]
[-from from_pins]
[-to to_pins]
[-rise] [-fall]
[-clock clock_check]
[-setup | -recovery]
[-hold | -removal]
[-nochange_high | -nochange_low]
[object_spec]

string clock_check
list from_pins
list to_pins
list object_spec
```

ARGUMENTS

-all

Specifies that all annotated timing checks in the design are to be removed. This option is exclusive of the **-from**, **-to**, and *object_spec* options. Options that specify the type of the timing check (for example, **-setup**, and **-hold**) are ignored when you use the **-all** option.

-from *from_pins*

Specifies a list of leaf cell pins that are the startpoints of the timing arcs for which annotated checks are to be removed. You must use the **-from** option with the **-to** option, and you cannot combine these with the *object_spec* option.

-to *to_pins*

Specifies a list of leaf cell pins that are the endpoints of the timing arcs for which annotated checks are to be removed. You must use the **-from** option with the **-to** option, and you cannot combine these with the *object_spec* option.

-rise

Specifies that the timing check is for the rise transition on the constrained (the data -to pin). The **-rise** and **-fall** options are mutually exclusive. If you do not specify either **-rise** or **-fall**, both values are removed.

-fall

Specifies that the timing check is for the fall transition on the constrained (the data -to pin). The **-rise** and **-fall** options are mutually exclusive. If you do not specify either **-rise** or **-fall**, both values are removed.

remove_annotated_check

`-clock clock_check`
 Specifies whether the check is for clock rising or falling. By default, checks for both clock rise and fall are removed. The valid values for *clock_check* are **rise** or **fall**.

`-clock clock_check`
 Specifies whether the check is for clock rising or falling. By default, checks for both clock rise and fall are removed. The valid values for *clock_check* are **rise** or **fall**.

`-setup | -recovery`

`-hold | -removal`

`-nochange_high | -nochange_low`
 Specifies the type of the timing check. If none are specified, all annotated timing checks are removed. More than one can be specified, but the following combinations are not allowed:
-setup with **-recovery**
-hold with **-removal**
-nochange_high with **-nochange_low**

`object_spec`
 Specifies a list of leaf cells for which annotated timing checks are to be removed. You cannot combine this option with the **-from** and **-to** options.

DESCRIPTION

Removes annotated timing checks from the design, either on specific cells, between specific pins, or on all cells in the current design. Timing checks are annotated either from a Standard Delay Format (SDF) file (using the **read_sdf** command), or by using the **set_annotated_check** command. Timing check types include setup, hold, recovery, removal, and nochange.

You can use the **remove_annotated_check** command in the following ways:

- You can remove all annotated checks from the entire design by using the **-all** option; in this case, you cannot remove a specific check. The command ignores options that specify the type of the timing check (for example, **-setup** and **-hold**) when you use the **-all** option.
- You can remove annotated checks from a list of cells by using the **object_spec** option. Without specifying any timing check type options, the command removes all annotated checks from each cell in the list. By using options that specify the type of the timing check (for example, **-setup** and **-hold**) you can remove only specific checks. You can use this method to remove all checks of a certain type from the entire design. For an example, see the EXAMPLES section.
- You can remove annotated checks between specific pins by using the **-from** and **-to** options. Each from/to pair must be on the same cell. Without specifying any timing check type options, the command removes all annotated checks from each from/to pair.

By using options that specify the type of the timing check (for example, **-setup** and **-hold**), you can remove only specific checks.

EXAMPLES

The following example removes all annotated cell timing checks from the current design.

```
pt_shell> remove_annotated_check -all
1
```

The following example removes only setup checks from all cells in the current design.

```
pt_shell> remove_annotated_check -setup [get_cells *]
1
```

The following example removes checks between pins. It also shows the error condition when the pins are not on the same cell and the warning when there are no annotated checks.

```
pt_shell> remove_annotated_check -from ffb/CP -to ffa/D
```

```
Error: Cannot remove annotated check from 'ffb/CP' to 'ffa/D':
pins are on different cells (PTE-032)
```

```
0
```

```
pt_shell> remove_annotated_check -from ffa/CP -to ffa/D -setup -hold
1
```

```
pt_shell> remove_annotated_check -from ffa/CP -to ffa/D
Warning: No annotated timing checks were removed. (PTE-031)
0
```

SEE ALSO

read_sdf (2), **report_annotated_check** (2), **report_annotated_delay** (2), **reset_design** (2), **set_annotated_check** (2), **set_annotated_delay** (2).

remove_annotated_clock_network_power

Remove the annotate power on clock networks.

SYNTAX

```
string remove_annotated_clock_network_power  
[-clock clock_object]
```

Data Types

clock_object string

ARGUMENTS

-clock clock_object
 Specifies the clock of the related clock network on which the annotated power values are removed.

DESCRIPTION

The **remove_annotated_clock_network_power** command removes the annotated power information specified by the **set_annotated_clock_network_power** command on the current design. If the **-clock** option is used, only the annotated clock network power for the specific clock domain is removed; otherwise, the annotated power values for all clock domains are removed. If the power is not annotated for the specified clock domain, a **PWR-294** error message is issued and the command fails.

In the following example, the **remove_annotated_clock_network_power** command is used before the **report_power** command and after the **set_annotated_clock_network_power** command. The output from the **report_power** command is not affected.

```
pt_shell> set_annotated_clock_network_power -internal 1.0e-03 -switching 2.0e-03  
pt_shell> remove_annotated_clock_network_power  
pt_shell> report_power
```

SEE ALSO

```
set_annotated_clock_network_power(2)  
report_power(2)
```

remove_annotated_delay

Removes annotated delays from the design, either on specific cells or nets, between specific pins, or all annotated delays in the design.

SYNTAX

```
string remove_annotated_delay
[-all]
[-from from_list]
[-to to_list]
[object_spec]

list  from_list
list  to_list
list  object_spec
```

ARGUMENTS

-all

Indicates that all annotated delays in the design are to be removed. This option is exclusive of the **-from**, **-to**, and *object_spec* options.

-from *from_list*

Specifies a list of pins or ports that are the startpoints of the timing arcs for which annotated delays are to be removed. You cannot combine this option with *object_spec*.

-to *to_list*

Specifies a list of pins or ports that are the endpoints of the timing arcs for which annotated delays are to be removed. You cannot combine this option with *object_spec*.

object_spec

Specifies a list of leaf cells or nets for which all annotated delays are to be removed. You cannot combine this option with **-from** and **-to**.

DESCRIPTION

The **remove_annotated_delay** command removes annotated cell and net delays from the current design. Delays are annotated either from an SDF file (using the **read_sdf** command) or by using the **set_annotated_delay** command.

There are several ways to use the **remove_annotated_delay** command.

- You can remove all annotated delays from the entire design using the **-all** option.
- You can remove all annotated delays from a list of cells or nets using the *object_spec* option.

- You can remove annotated delays between specific pins using the **-from** and **-to** options.

EXAMPLES

The following example removes all annotated net and cell delays from the current design.

```
pt_shell> remove_annotated_delay -all
1
```

The following example removes annotated delays from some cells.

```
pt_shell> remove_annotated_delay [get_cells u1*]
1
```

This example removes annotated delays between pins. Also shown is the warning when there are no annotated delays between the specified pins.

```
pt_shell> remove_annotated_delay -from ffb/Q
1
pt_shell> remove_annotated_delay -from ffb/Q -to u1/A
Warning: No annotated delays from 'ffb/Q' to 'u1/A'. (PTE-030)
0
```

SEE ALSO

```
read_sdf (2), remove_annotated_check (2), report_annotated_check (2),
report_annotated_delay (2), reset_design (2), set_annotated_check (2),
set_annotated_delay (2).
```

remove_annotated_parasitics

Removes all annotated parasitics from nets of the current design.

SYNTAX

Boolean **remove_annotated_parasitics**
[-all | *net_list*]

list *net_list*

ARGUMENTS

-all

Indicates that all annotated nets in the design are to be removed; this is the default. **-all** and *net_list* are mutually exclusive.

net_list

Specifies a list of nets from which annotated parasitics are to be removed. By default, all annotated parasitics are removed from the design. **-all** and *net_list* are mutually exclusive.

DESCRIPTION

Removes all net parasitics annotated on nets of the current design. The annotated parasitics are usually added using the **read_parasitics** command to read an SPEF or SPF file and annotate the parasitics on nets of the current design.

remove_annotated_parasitics also removes the annotated parasitics completed by the **complete_net_parasitics** command.

There are two ways to use the **remove_annotated_parasitics** command.

- You can remove all annotated parasitics from the entire design using the **-all** option, or by executing the command without options.
- You can remove all annotated parasitics from a list of nets using the *net_list* option.

EXAMPLES

The following example removes all annotated net parasitics from the current design.

```
pt_shell> remove_annotated_parasitics -all  
1
```

The following example removes annotated parasitics from the net "CLK".

```
pt_shell> remove_annotated_parasitics [get_nets CLK]
```

```
remove_annotated_parasitics  
420
```

SEE ALSO

`complete_annotated_parasitics (2)`, `read_parasitics (2)`, `report_annotated_parasitics (2)`, `reset_design (2)`.

remove_annotated_power

Remove previously-annotated power from unresolved black-box cells or leaf cells.

SYNTAX

```
int remove_annotated_power -all | cell_list

list    cell_list
```

ARGUMENTS

- all
Indicates that all annotated powers in the design are to be removed. -all and cell_list are mutually exclusive.
- cell_list
Specifies a list of cells from which annotated powers are to be removed. -all and cell_list are mutually exclusive.

DESCRIPTION

The remove_annoated_power command removes powers previously annotated on cells using set_annotated_power. Both the internal power and the leakage power annotated by the set_annotated_power command are to be removed. There is no way to remove only internal power or leakage power. You can remove all annotated power by using -all option, or remove only the power annotated on the specified cell list.

EXAMPLES

The following example shows how power is annotated, removed and reported.

```
pt_shell> set_annotated_power -int 1 -leak 0.01 u0/*
1
pt_shell> report_annotated_power -list_annotated
*****
Report : annotated_power
        -list_annotated
*****

Annotated cell powers:
-----
1.  u0/u0  (internal: 1  leakage: 0.01)
2.  u0/u1  (internal: 1  leakage: 0.01)
```

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	1	2

		5		2		3	

```

1
pt_shell> remove_annotated_power u0/u0
1
pt_shell> report_annotated_power
*****
Report : annotated_power
*****

```

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	0	3
	5	1	4

```

1

```

SEE ALSO

`set_annotated_power (2)`, `report_annotated_power (2)`.

remove_annotated_transition

Removes previously-annotated transition times from pins or ports in the current design.

SYNTAX

```
int remove_annotated_transition
    -all | pin_list
```

```
list pin_list
```

ARGUMENTS

-all

Indicates that all annotated transition times in the design are to be removed. **-all** and *pin_list* are mutually exclusive; you must use one of these, but not both.

pin_list

Specifies a list of pins or ports from which annotated transition times are to be removed. **-all** and *pin_list* are mutually exclusive; you must use one of these, but not both.

DESCRIPTION

The **remove_annotated_transition** command removes transition times previously annotated on pins or ports from the current design using **set_annotated_transition**.

There are two ways to use the **remove_annotated_transition** command.

- You can remove all annotated transition times from the entire design using the **-all** option.

- You can remove all annotated transition times from a list of pins or ports using the *pin_list* option.

To view current settings, use **report_timing**.

EXAMPLES

The following example removes all annotated pin and port transition times from the current design.

```
pt_shell> remove_annotated_transition -all
1
```

The following example removes annotated transition time from the input pin **A** of cell "U1/U2/U3".


```
pt_shell> remove_annotated_transition [get_pins U1/U2/U3/A]
1
```

SEE ALSO

`remove_annotated_delay` (2), `reset_design` (2), `report_timing` (2), `set_annotated_delay` (2), `set_annotated_transition` (2).

remove_aocvm

Removes advanced on-chip variation (OCV) information.

SYNTAX

```
int remove_aocvm  
[-coefficient] [-derate]  
[object_list]
```

Data Types

object_list list

ARGUMENTS

-coefficient

Indicates that only advanced OCV coefficients are removed. The *-coefficient* and *-derate* options are mutually exclusive.

-derate

Indicates that only advanced OCV derate factors are removed. The *-coefficient* and *-derate* options are mutually exclusive.

object_list

If *-coefficient* has been specified, indicates that the advanced OCV coefficients set on the library cells, library timing arcs, and cells in the *object_list* are removed. If *-derate* has been specified, indicates that the advanced OCV derate factors set on the design, hierarchical cells, and library cells in the *object_list* are removed.

DESCRIPTION

Removes user-specified advanced OCV coefficients and advanced OCV derate factors that were previously set using the **set_aocvm_coefficient** and **read_aocvm** commands respectively. If the command has been specified with no options, all advanced OCV data is removed.

To display advanced OCV derates and advanced OCV coefficients, use the **report_aocvm** command.

EXAMPLES

The following example removes the random advanced OCV coefficients from the library cell named *lib/bufA*.

```
pt_shell> remove_aocvm -coefficient lib/bufA
```

The following example removes all advanced OCV derate factors that were set using the **read_aocvm** command.

```
pt_shell> remove_aocvm -derate
```

The following example removes all advanced OCV derate factors from the current design.

```
pt_shell> remove_aocvm -derate [current_design]
```

SEE ALSO

```
read_aocvm(2)  
report_aocvm(2)  
set_aocvm_coefficient(2)
```

remove_buffer

Removes specified buffers from the current design.

SYNTAX

```
int remove_buffer cell_list  
list cell_list
```

ARGUMENTS

cell_list
Specifies a list of buffer cells to be removed.

DESCRIPTION

The **remove_buffer** command is used to remove buffer cells from the netlist. A buffer is one or more cells that conforms to the rules **insert_buffer** uses to create buffers. Like all other netlist editing commands, for **remove_buffer** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **remove_buffer** returns a 1 if successful and a 0 if unsuccessful.

Like **insert_buffer**, the arguments to **remove_buffer** can be grouped. The basic interpretation of the *cell_list* is

(one non-inverting buffer | two cascaded inverting buffers)+

To determine whether a removable buffer configuration exists, **remove_buffer** applies the following rules to the cells in *cell_list*:

- Each cell must be a buffer as defined by **insert_buffer**.
- The input pin of each cell must be connected to a net.
- For multi-output cells, only one output pin can be connected.
- There can be no standalone inverting cells.
- The output net cannot be multi-driven.
- Inverter pairs must be connected together, and the intermediate net can be connected only to the two inverters.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

Refer to the following diagram, which contains inverters U1 and U2:

```
e1/Z -- old_net -- U1 -- net1 --- U2 -- net2 --- e3/A
```

The following example removes U1 and U2.

```
pt_shell> remove_buffer {U1 U2}  
Removed buffer 'U1', 'U2'  
1
```

After removing the buffer, the configuration is as follows:

```
e1/Z -- old_net --- e3/A
```

SEE ALSO

insert_buffer (2), size_cell (2), swap_cell (2), write_changes (2), PARA-061.

remove_capacitance

Removes capacitance on nets or ports.

SYNTAX

```
string remove_capacitance net_or_port_list  
list  net_or_port_list
```

ARGUMENTS

net_or_port_list
Specifies a list of ports and nets in the current design, whose capacitances are removed.

DESCRIPTION

Specifies that the lumped capacitance annotated on the list of nets or ports must be removed. PrimeTime will revert to using the capacitance from detailed parasitics (set using **read_parasitics**), if they exist. If not the internally estimated net capacitance will be used.

Annotated capacitances are also removed on the full design by doing a *reset_design*.

EXAMPLES

The following example removes the annotated capacitance on net w23.

```
pt_shell> remove_capacitance [get_nets "w23"]
```

The following example removes the annotated capacitance on all ports that name match "P2*".

```
pt_shell> remove_capacitance [get_ports "P2*"]
```

SEE ALSO

all_outputs (2), **current_design** (2), **set_load** (2), **report_net** (2),
report_internal_loads (2), **report_port** (2), **reset_design** (2), **set_drive** (2),
set_wire_load (2).

remove_case_analysis

Removes the case analysis value on input.

SYNTAX

```
string remove_case_analysis port_or_pin_list  
list port_or_pin_list
```

ARGUMENTS

port_or_pin_list
Lists ports or pins for which the case analysis entry is to be removed.

DESCRIPTION

This command removes previously specified entries of case analysis on ports or pins. Case analysis is specified using command **set_case_analysis**. The command specifies that a pin or port is at a constant logic value (0 or 1), or that only one of the rising or falling transition is valid.

EXAMPLES

The following example specifies that ports in0 and in2 are set to the constant logic value 0.

```
pt_shell> set_case_analysis 0 {in0 in2} pt_shell> report_case_analysis
```

```
*****  
Report : case_analysis  
Design : middle  
Version: 1997.01-development  
Date   : Mon Jul 22 08:13:50 1996  
*****
```

Pin name	Case analysis value
in2	0
in0	0

The following example removes the case analysis entry on port in2.

```
pt_shell> remove_case_analysis {in2} pt_shell> report_case_analysis
```

```
*****  
Report : case_analysis  
Design : middle  
Version: 1997.01-development  
Date   : Mon Jul 22 08:14:16 1996  
*****
```

Pin name	Case analysis value

remove_case_analysis

in0

0

SEE ALSO

`set_case_analysis (2)`, `report_case_analysis (2)`.

remove_cell

Removes cells from the current design.

SYNTAX

```
int remove_cell cell_list | -all  
list cell_list
```

ARGUMENTS

-all

Indicates that all cells are to be removed from the current design. **-all** and *cell_list* are mutually exclusive; you can specify only one. **Note:** Using *-all* at the top of the design effectively removes the entire design.

cell_list

Specifies a list of cells to be removed from the current design. **-all** and *cell_list* are mutually exclusive; you can specify only one.

DESCRIPTION

The **remove_cell** command removes cells from the current design. Like all other netlist editing commands, for **remove_cell** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **remove_cell** returns a 1 if successful and a 0 if unsuccessful.

Each cell specified must be in scope; that is, at or below the current instance. The **remove_cell** command can be used on leaf cells or hierarchical cells. Duplicate arguments to **remove_cell** are ignored with a warning. As **remove_cell** progresses, it echoes the cells passed to it. However, only the cells that were actually deleted are logged to the change list. For more information, see the EXAMPLES section.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK,

PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following example, the first command shows the cells within u1 and u2. The second command removes what appears to be four cells; in fact, more than four cells are deleted. Because u1 and u2 each contain three cells, a total of eight cells are actually deleted. Note also that the change list has only three entries. Because u1/u2 is contained within u1 and is removed after u1, u1/u2 is dropped from the change list.

```
pt_shell> get_cells {u1/* u2/*}
{"u1/u1", "u1/u2", "u1/u3", "u2/u1", "u2/u2", "u2/u3"}
pt_shell> remove_cell {u1 u1/u2 u2/u3 u2}
Information: Removed cell 'u1'. (NED-015)
Information: Removed cell 'u1/u2'. (NED-015)
Information: Removed cell 'u2/u3'. (NED-015)
Information: Removed cell 'u2'. (NED-015)
```

```

1
pt_shell> write_changes
#####
# Change list, formatted for PrimeTime
#####
current_instance
remove_cell {u1}
current_instance
current_instance u2
remove_cell {u3}
current_instance
remove_cell {u2}
1

```

SEE ALSO

size_cell (2), **swap_cell** (2), **write_changes** (2).

remove_clock

Removes one or more clocks from the current design.

SYNTAX

```
string remove_clock -all | clock_list  
list clock_list
```

ARGUMENTS

-all

Specifies to remove all clocks in the current design.

clock_list

Specifies a list of collections containing clocks or patterns matching the clock names.

DESCRIPTION

Removes the specified clock objects from the current design. You must specify either *clock_list* or **-all**.

If a removed clock is the only member of a path group, the path group is also removed. For information about path groups, refer to the **group_path** man page. To list all clock sources in the design, use **report_clock**.

All arrival and required times specified by commands **set_input_delay** and **set_output_delay** relatively to the clock that is being removed are also removed.

EXAMPLES

The following example removes clock CLK1.

```
pt_shell> remove_clock CLK1
```

The following example removes all clocks from the current design.

```
pt_shell> remove_clock -all
```

SEE ALSO

create_clock (2), **current_design** (2), **get_clocks** (2), **group_path** (2), **report_clock** (2), **set_input_delay** (2), **set_output_delay** (2), **reset_design** (2).

remove_clock_gating_check

Captures clock-gating checks.

SYNTAX

```
string remove_clock_gating_check [-setup] [-hold] [-rise] [-fall] [-high | -low]
[object_list]
list object_list
```

ARGUMENTS

-setup

Indicates the removal of the clock-gating constraint on the setup time only. If you do not specify either the **-setup** or **-hold** option, both setup and hold constraints are removed.

-hold

Indicates the removal of the clock-gating constraint on the hold time only. If you do not specify either the **-setup** or **-hold** option, both setup and hold constraints are removed.

-rise

Indicates the removal of the clock-gating constraint on the rising delays only. If you do not specify either the **-rise** or **-fall** option, constraints on both rising and falling delays are removed.

-fall

Indicates the removal of the clock-gating constraint on the falling delays only. If you do not specify either the **-rise** nor **-fall** option, constraints on both rising and falling delays are removed.

-high

Remove the high specification from the object list, previously set up by `set_clock_gating_check` command. This option has to be either high or low..

-low

Remove the low specification from the object list, previously set up by `set_clock_gating_check` command. This option has to be either high or low.

object_list

Specifies a list of objects in the current design for which to remove the clock gating check. The objects can be clocks, ports, pins, or cells. If you specify a cell, all input pins of that cell are affected. If you do not specify any objects, the clock-gating check is removed from the current design.

DESCRIPTION

The **remove_clock_gating_check** command removes clock gating checks for design objects set by **set_clock_gating_check**.

EXAMPLES

The following example removes the setup requirement (for rising and falling delays) on all gates in the clock network involved with clock CK1 path.

```
pt_shell> remove_clock_gating_check -setup [get_clocks CK1]
```

The following example removes the hold requirement on the rising delay of gate and1.

```
pt_shell> remove_clock_gating_check -hold -rise [get_cells and1]
```

An alternative way to remove information set by **set_clock_gating_check** is to use the **reset_design** command.

SEE ALSO

```
set_clock_gating_check (2), set_disable_clock_gating_check (2),  
remove_disable_clock_gating_check (2), current_design (2), report_constraint (2),  
reset_design (2).
```

remove_clock_groups

Removes specific exclusive or asynchronous clock groups from the current design.

SYNTAX

Boolean **remove_clock_groups**

-physically_exclusive | **-exclusive** | **-asynchronous**
-name *name_list* | **-all**

list *name_list*

ARGUMENTS

-physically_exclusive

Specifies that groups set for physically exclusive clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-logically_exclusive

Specifies that groups set for logically exclusive clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-asynchronous

Specifies that groups set for asynchronous clocks are to be removed. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-name *name_list*

Specifies a list of clock groups to be removed, which matches the groups in the given names. You should use the **set_clock_groups** command to predefine these names. Substitute the list you want for *name_list*. The **-name** and **-all** options are mutually exclusive.

-all

Specifies to remove all groups set for exclusive or asynchronous clocks in the current design. The **-name** and **-all** options are mutually exclusive.

DESCRIPTION

Removes specific exclusive or asynchronous clock groups from the current design. These clock groups are specified by the *name_list* from the current design. You must specify either **-exclusive** or **-asynchronous**. You must specify either *name_list* or **-all**.

To find the names for existing groups, use the **report_clock** command with the **-groups** option.

EXAMPLES

The following example removes exclusive clock groups named *mux*.

```
remove_clock_groups
```



```
pt_shell> remove_clock -exclusive -name mux
```

The following example removes all asynchronous clock groups from the current design.

```
pt_shell> remove_clock -asynchronous -all
```

SEE ALSO

report_clock (2), **set_clock_groups** (2).

remove_clock_latency

Removes clock latency information from specified objects.

SYNTAX

```
string remove_clock_latency [-source]  
[-clock clock_list]  
object_list
```

```
list    clock_list  
list    object_list
```

ARGUMENTS

-source

Specifies that clock source latency should be removed.

-clock *clock_list*

Removes any network latency defined on the pin/port objects in *object_list* which refers the clocks in *clock_list* from the design. If the **-clock** option is supplied when *object_list* refers to clock objects, a warning is issued that the option is not relevant in this case and execution of the command proceeds as if **-clock** was not given. This option does not remove a more general latency setting without any specific clock.

object_list

Provides a list of clocks, ports, or pins.

DESCRIPTION

Removes clock latency information from specified objects. It removes the user-specified clock network or source latency information from specified objects. If a dynamic component of clock source latency has been specified this will also be removed. Clock Network latency is the time it takes for a clock signal on the design to propagate from the clock definition point to a register clock pin. Clock Source latency (also called insertion delay) is the time it takes for a clock signal to propagate from its actual ideal waveform origin point to the clock definition point in the design. Clock network and source latency information is set on objects using the **set_clock_latency** command. See the **set_clock_latency** man page for more details.

To report clock network and source latency information, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes clock latency information from the clock named **CLK1**.

```
pt_shell> remove_clock_latency [get_clocks CLK1]
```

The following example removes clock source latency information from the clock named **CLK1**.

```
pt_shell> remove_clock_latency -source \  
[get_clocks CLK1]
```

SEE ALSO

```
create_clock (2), remove_clock (2), remove_clock_uncertainty (2), report_clock (2),  
set_clock_latency (2), set_clock_uncertainty (2).
```

remove_clock_sense

Removes unateness information defined on pins or cell timing arcs.

SYNTAX

```
string remove_clock_sense  
[-all]  
[-clocks clock_list]  
object_list
```

```
list clock_list  
list object_list
```

ARGUMENTS

-clocks *clock_list*
Optionally specifies a list of clock objects to be associated with the given pin objects in *object_list*. If the **-clocks** option is specified, only the unateness specified for that particular clock domain will be removed. Otherwise, unateness information for all clocks passing through the given pin objects will be removed. The **-clocks** option can only remove clock sense predefined by **set_clock_sense -clock**. It does not remove the default clock sense setting for this given pin.

-all
Remove all unateness information in current design.

object_list
Lists of pins or cell timing arcs with predefined unateness to remove.

DESCRIPTION

Removes the unateness information predefined by the user. To set the unateness information, use the **set_clock_sense** command.

EXAMPLES

The following example removes positive unateness defined on a pin named **XOR/Z** with respect to clock **CLK1**, but does not remove the negative unateness.

```
pt_shell> set_clock_sense -positive -clocks [get_clocks CLK1] XOR/Z  
pt_shell> set_clock_sense -negative XOR/Z  
pt_shell> remove_clock_sense -clocks [get_clocks CLK1] XOR/Z
```

The following example removes all unateness information in the current design.

```
pt_shell> remove_clock_sense -all
```

SEE ALSO

`set_clock_sense` (2).

remove_clock_transition

Removes clock transition time information.

SYNTAX

```
string remove_clock_transition clock_list  
list clock_list
```

ARGUMENTS

clock_list
Specifies a list of clocks for which to remove clock transition time.

DESCRIPTION

Removes clock transition information specified by the **set_clock_transition** command. To list all the set clock transition values, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes clock transition information from all clocks in the current design.

```
pt_shell> remove_clock_transition [all_clocks]
```

SEE ALSO

all_clocks (2), **report_clock** (2), **set_clock_transition** (2), **create_clock** (2), **set_clock_latency** (2), **set_clock_uncertainty** (2).

remove_clock_uncertainty

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command.

SYNTAX

```
string remove_clock_uncertainty  
[object_list |  
  -from from_clock  
    | -rise_from rise_from_clock  
    | -fall_from fall_from_clock  
  -to to_clock  
    | -rise_to rise_to_clock  
    | -fall_to fall_to_clock]  
[-rise]  
[-fall]  
[-setup]  
[-hold]  
[object_list]  
  
list from_clock  
list rise_from_clock  
list fall_from_clock  
list rise_to_clock  
list fall_to_clock  
list to_clock  
list object_list
```

ARGUMENTS

-from *from_clock* -to *to_clock*

These two options specify the source and destination clocks for interclock uncertainty. You must specify either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to**, or *object_list*; you cannot specify both.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to rising edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options. Use **-rise_from** instead of the obsolete option **-from_edge rise**.

-fall_from *fall_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to falling edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options. Use **-fall_from** instead of the obsolete option **-from_edge fall**.

-rise_to *rise_to_clock*

Same as the **-to** option, but indicates that *uncertainty* applies only to rising edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options. Use **-rise_to** instead of the obsolete option **-to_edge rise**.

`-fall_to` *fall_to_clock*

Same as the **-to** option, but indicates that *uncertainty* applies only to falling edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options. Use **-fall_to** instead of the obsolete option **-to_edge fall**.

`object_list`

Specifies a list of clocks, ports or pins from which uncertainty information is to be removed. You can use either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to** options or the *object_list* option, but you cannot specify both; they are mutually exclusive.

`-rise`

Specifies that uncertainty is to be removed for only the rising clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-rise_to** instead.

`-fall`

Specifies that uncertainty is to be removed for only the falling clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-fall_to** instead.

`-setup`

Specifies that only setup check uncertainty is to be removed. By default, both setup and hold check uncertainties are removed.

`-hold`

Specifies that only hold check uncertainty is to be removed. By default, both setup and hold check uncertainties are removed.

DESCRIPTION

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command from clocks, ports, pins, or between specified clocks. To display clock uncertainty information, use the **report_clock** command with the **-skew** option.

EXAMPLES

The following example removes uncertainty information from a clock named *CLK* and a pin named *clk_buf/Z*.

```
pt_shell> remove_clock_uncertainty [get_clocks CLK]
pt_shell> remove_clock_uncertainty [get_pins clk_buf/Z]
```

The following example removes interclock uncertainties between the *PHI1* and *PHI2* clock domains.

```
pt_shell> remove_clock_uncertainty -from PHI1 -to PHI1
pt_shell> remove_clock_uncertainty -from PHI2 -to PHI2
pt_shell> remove_clock_uncertainty -from PHI1 -to PHI2
pt_shell> remove_clock_uncertainty -from PHI2 -to PHI1
```


SEE ALSO

`all_clocks` (2), `create_clock` (2), `report_clock` (2), `set_clock_latency` (2),
`set_clock_transition` (2), `set_clock_uncertainty` (2).

remove_connection_class

Removes connection class value from ports.

SYNTAX

```
int remove_connection_class  
object_list
```

```
list object_list
```

ARGUMENTS

object_list
Specifies ports whose connection classes are to be removed.

DESCRIPTION

Removes any values for previously set on port with the *set_connection_class* class command.

EXAMPLES

The following example removes any existing connection class values for an xyz port name.

```
pt_shell> remove_connection_class xyz
```

SEE ALSO

set_connection_class (2), **report_constraint** (2)

remove_context

Deletes the timing context information.

SYNTAX

```
string remove_context [-timing][-environment] [-design_rules] [-constant_inputs]  
cell_list  
list cell_list
```

ARGUMENTS

-timing

Deletes timing-related context information such as clocks, input and output delays, and timing exceptions.

-environment

Deletes environment related information such as operating conditions (process, temperature, and voltage), wire load model, capacitive loads on input and output pins, and driving cell information on input pins.

-design_rules

Deletes characterized design rule checks such as max_capacitance, max_transition, and max_fanout.

-constant_inputs

Deletes characterized logic constants propagated to input pins.

cell_list

Provides a list of instances for which the context is deleted.

DESCRIPTION

Deletes the timing context captured using the **characterize_context** command for the specified list of instances. Options specify the categories of context information to delete. All context information is deleted if no option is specified.

EXAMPLES

The following command deletes the timing-related context information for instances I1 and I2.

```
pt_shell> remove_context -timing {I1 I2}
```

The following command deletes the environment and design rule context information for instance I2.

```
pt_shell> remove_context -env -design_rules I2
```

SEE ALSO

`characterize_context` (2), `write_context` (2), `report_context` (2).

remove_coupling_separation

Removes the constraints set by **set_coupling_separation** command.

SYNTAX

```
int remove_coupling_separation [-pairwise pair_nets] [-all] nets  
list pnets  
list nets
```

ARGUMENTS

-pairwise

When **-pairwise pnets** is applied, all coupling capacitances between only **pnets** and **nets** are excluded. This option **-pairwise** can be only used to remove separation constraints applied on the nets using the command **set_coupling_separation** with the option **-pairwise**. Any coupling separation applied on the nets globally without the **-pairwise** option cannot be reset by this option.

-all

When **-all** option is used, coupling separation constraints are reset on all the nets.

DESCRIPTION

This command removes the constraints set by **set_coupling_separation** command.

When **-pairwise pnets** is applied, only separation constraints between **pnets** and **nets** are removed.

Any prior settings on **pnets** or **nets** from a **set_si_delay_analysis** or **set_si_noise_analysis** command are restored.

Instances of this command are recorded for output by the **write_changes** command. This feature allows the user to communicate the separation constraint to other implementation tools along with netlist changes.

The **remove_coupling_separation** command returns a **1** if successful and a **0** if unsuccessful. If remove command is used to remove a constraint that has not been set, warning message **XTALK-107** is issued.

To view the result of this command, use the **report_si_delay_analysis** or **report_si_noise_analysis** with the **-coupling_separated** option.

EXAMPLES

In the following example, all the nets named **CLK_NET_*** are returned to their original state in crosstalk analysis.

```
pt_shell> set_coupling_separation [get_nets CLK_NET*]  
1
```

```
pt_shell> remove_coupling_separation [get_nets CLK_NET*]  
1
```

In the following example, the separation is applied pairwise on the nets and is removed by using the **-pairwise** option.

```
pt_shell> set_coupling_separation -pairwise [get_nets {n1 n2}] [get_nets n3]  
1  
pt_shell> remove_coupling_separation -pairwise [get_nets {n1 n2}] [get_nets n3]  
1
```

However if the separation is applied globally on the net for all its coupled nets, it cannot be removed by using the **-pairwise** option.

```
pt_shell> set_coupling_separation [get_nets {n4}]  
1  
pt_shell> remove_coupling_separation [get_nets {n4}] -pairwise [get_nets n3]  
Warning: Cannot remove an effect that was not set on net(s) n4 n3. (XTALK-107)  
1
```

The coupling separation can be applied on the net pairwise with all its coupled nets in the following way so that pairwise exclusions can be applied on the net with any of its coupled nets.

```
pt_shell> set_coupling_separation [get_nets VIC_NET] -pairwise  
[get_attribute -class net [get_nets VIC_NET] aggressors]  
1  
  
pt_shell> remove_coupling_separation [get_nets VIC_NET] -pairwise  
[get_nets AGG_NET*]  
1  
pt_shell> remove_coupling_separation -all  
1
```

To verify if the separation constraint was removed on the nets, use the **report_si_delay_analysis** or **report_si_noise_analysis** commands with the **-coupling_separated** option.

SEE ALSO

set_coupling_separation (2), **report_si_delay_analysis** (2), **report_si_noise_analysis** (2), **write_changes** (2).

remove_current_session

Removes the previously set session.

SYNTAX

boolean **remove_current_session**

DESCRIPTION

In multi-scenario analysis, a list of scenarios can be brought into session and command focus (selected for analysis) using the **current_session** command. These scenarios can then be defocused using the **remove_current_session** command, however, the scenarios will not be removed from memory and could be set in focus for another subsequent session.

EXAMPLES

In the following example, two scenarios are in the current session. These are then removed from focus.

```
pt_shell> current_session {scen2 scen3}
1
pt_shell> remove_current_session
1
```

SEE ALSO

report_multi_scenario_design (2), **remove_scenario**(2)

remove_data_check

Removes specified data-to-data checks previously set by `set_data_check`.

SYNTAX

```
string remove_data_check  
{-from from_object  
  | -rise_from from_object  
  | -fall_from from_object}  
{-to to_object  
  | -rise_to to_object  
  | -fall_to to_object}  
[-setup | -hold]  
[-clock clock]
```

```
object from_object  
object to_object  
object clock_object
```

ARGUMENTS

`-from from_object`

Specifies a pin or port in the current design as the related pin of the data-to-data check to be removed. Both rising and falling checks are removed. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

`-to to_object`

Specifies a pin or port in the current design as the constrained pin of the data-to-data check to be created. Both rising and falling checks are removed. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

`-rise_from from_object`

Similar to the **-from** option, but applies to only rising delays at the related pin. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

`-fall_from from_object`

Similar to the **-from** option, but applies to only falling delays at the related pin. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

`-rise_to to_object`

Similar to the **-to** option, but applies to only rising delays at the constrained pin. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

`-fall_to to_object`

Similar to the **-to** option, but applies to only falling delays at the constrained pin. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

`-setup`

Indicates that only the setup data check is to be removed. If neither **-setup** nor **-hold** is specified, both setup and hold checks are removed.

`remove_data_check`

-hold

Indicates that only the hold data check is to be removed. If neither **-setup** nor **-hold** is specified, both setup and hold checks are removed.

-clock *clock_object*

Indicates that the data check for the specified clock at the related pin is to be removed. This option applies only if a previous **set_data_check** command used the **-clock** option to specify the same clock; otherwise, this option is ignored.

DESCRIPTION

The **remove_data_check** command specifies data-to-data check(s) to be removed between the from object and to object for the specified options. These checks were previously set using the **set_data_check** command.

EXAMPLES

The following example removes a data-to-data check from and1/B to and1/A with respect to the rising edge of signals at and1/B. Both setup and hold checks are removed.

```
pt_shell> remove_data_check -rise_from and1/B -to and1/A
```

The following example removes setup data-to-data check between and1/B to and1/A with respect to the rising edge of signals at and1/B, coming from startpoints triggered with clock CK1, falling edge of signals at and1/A.

```
pt_shell> remove_data_check -rise_from and1/B -fall_to and1/A -setup -  
clock [get_clock CK1]
```

SEE ALSO

```
report_timing (2), set_data_check (2), update_timing (2);  
timing_enable_multiple_clocks_per_reg (3),  
timing_propagate_through_unclocked_latches (3).
```

remove_delcalc_resource

Removes the previously defined delcalc resources.

SYNTAX

```
int remove_delcalc_resource resource_name -all  
string resource_name
```

ARGUMENTS

resource_name
Specifies the name of the resource.

-all
Specifies all of the resources.

DESCRIPTION

Removes the previously defined delcalc resource. The argument can be either a single resource name or **-all**. The **-all** option removes all the resources that were previously defined.

EXAMPLES

This first example removes resource1. The second example removes all the resources which were previously defined.

```
pt_shell> remove_delcalc_resource resource1  
pt_shell> remove_delcalc_resource -all
```

SEE ALSO

set_delcalc_resource (2), **list_delcalc_resources** (2),

remove_design

Removes one or more designs from memory.

SYNTAX

```
string remove_design -all -hierarchy designs  
list designs
```

ARGUMENTS

-all

Indicates that all designs are to be removed. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

-hierarchy

Indicates that all designs in the hierarchy of the current design are to be removed, including the current design. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

designs

Specifies a list of designs to remove. **-all**, **-hierarchy**, and *designs* are mutually exclusive; you can specify only one.

DESCRIPTION

The **remove_design** command removes designs from pt_shell. You must specify either **-all**, **-hierarchy** or *designs*, but not more than one of those. **remove_design** does not remove libraries; use the **remove_lib** command for that purpose.

Note that you cannot combine the **-hierarchy** option with a list of designs. **-hierarchy** is restricted to the current design. If the current design is not linked, it is automatically linked. Then, all designs in the hierarchy of the current design are removed, including the current design itself.

EXAMPLES

The following example removes the design 'foo' and 'foo1'.

```
pt_shell> remove_design {d1 d2}  
Removing design 'd1'...  
Removing design 'd2'...  
1
```

The following example removes all designs in memory.

```
pt_shell> remove_design -all  
Removing design 'd3'...  
1
```

The following example removes all designs in the hierarchy of the current design. In this example, the design was already linked.

```
pt_shell> remove_design -hierarchy  
Removing design 'TOP'...  
Removing design 'eBlock'...  
Removing design 'iBlock'...  
Removing design 'mBlock'...  
1
```

SEE ALSO

current_design (2), **list_designs** (2), **remove_lib** (2).

remove_design_mode

Removes specified design modes and/or cell mode and path mappings to design modes.

SYNTAX

```
string remove_design_mode  
mode_list  
[-from from_pin_list]  
[-to to_pin_list]  
[-through through_pin_list]  
[cell_mode_list]  
[instance_list]
```

```
list mode_list  
list from_pin_list  
list to_pin_list  
list through_pin_list  
list cell_mode_list  
list instance_list
```

ARGUMENTS

mode_list

Specifies a list of design modes to be removed. Design modes on the list must have been previously created using the **define_design_mode_group** command.

-from from_pin_list

Specifies a timing path, from a given pin of the design, that is active only for the specified design mode. The given paths are inactive for other design modes. These paths will be unmapped from the design_mode and any previous settings on the path due to the design mode will be removed.

-to to_pin_list

Specifies a timing path, to a given pin of the design, that is active only for the specified design mode. The given paths are inactive for other design modes. These paths will be unmapped from the design_mode and any previous settings on the path due to the design mode are removed.

-through through_pin_list

Lists the pins, ports, or nets through which the paths pass. The paths become active for the specified mode. You can specify **-through** more than once in one command invocation. Nets are interpreted to imply the leaf-level driver pins. These paths will be unmapped from the design_mode and any previous settings on the path due to the design mode will be reset.

instance_list

Specifies a list of cells in which the specified modes are to be unmapped from the design mode. This list must be accompanied by a **cell_mode_list**

cell_mode_list

Specifies a list of cell modes which are to be unmapped from the design_mode. Any previous settings on any cell mode due to the design mode will be reset

DESCRIPTION

The **remove_design_mode** command removes specified design modes previously created by **define_design_mode_group** and/or removes mappings for specified design modes created by **map_design_mode**.

A design mode can be removed from a design mode group using **remove_design_mode mode_list**. When a design mode is removed from a design mode group all previous settings on cell modes and paths are reset. Also, if this design mode was the active design mode then all other design modes in the design mode group are reset to default.

A path can be unmapped from a design mode using **remove_design_mode -from [from_pin_list] -to [to_pin_list] -through [through_pin_list]** The pin specification must exactly match the pin specification set using a previous **map_design_mode** command. When a path is unmapped from a design mode, all paths are reset if they have been previously set false by that design mode. When a cell mode is unmapped from a design mode, all cell modes are reset if they have been previously set by that design mode.

To see a report of the design modes specified for the current design, use the **report_mode -type design** command. The report also shows the cell modes and paths mapped to each design mode.

EXAMPLES

The following example defines two design modes, DM1 and DM2, maps the cell mode READ to design mode DM1 for instance Uram1, then removes the design mode DM1. Removing DM1 also removes the mapping to the active cell mode READ.

```
pt_shell> define_design_mode_group {DM1 DM2}
pt_shell> map_design_mode READ Uram1 DM1
pt_shell> remove_design_mode DM1
```

SEE ALSO

map_design_mode (2), **define_design_mode_group** (2), **report_mode** (2), **set_mode** (2), **reset_mode** (2).

remove_disable_clock_gating_check

Restores clock gating checks previously disabled by **set_disable_clock_gating_check**, for specified cells and pins.

SYNTAX

```
string remove_disable_clock_gating_check object_list  
list object_list
```

ARGUMENTS

object_list
Specifies a list of cells and pins for whom previously-disabled clock gating checks are to be restored.

DESCRIPTION

Restores timing clock gating checks previously disabled with the **set_disable_clock_gating_check** command.

EXAMPLES

The following example restores disabled clock gating checks for the object an1.

```
pt_shell> remove_disable_clock_gating_check an1/A
```

The following example restores all disabled gating checks on the cell U1/or2.

```
pt_shell> remove_disable_clock_gating_check U1/or2
```

SEE ALSO

set_disable_clock_gating_check (2).

remove_disable_timing

Enables the previously disabled timing arcs.

SYNTAX

```
string remove_disable_timing  
[-from from_pin_name]  
[-to to_pin_name]  
object_list
```

```
string from_pin_name  
string to_pin_name  
list object_list
```

ARGUMENTS

-from *from_pin_name*

-to *to_pin_name*
Specifies that only the arcs between these two pins on the specified cell or library cell be disabled.

object_list
Specifies a list of cells, pins, library cells, library cell pins, or ports. The *object_list* must contain only cells or library cells (if **-from** and **-to** are specified).

DESCRIPTION

Restore timing arcs previously disabled with the **set_disable_timing** command. When **-from** and **-to** pins are specified, all arcs between these two pins on the cell or library cell are restored.

To list the timing arcs for a library cell, use **report_lib -timing_arcs**.

EXAMPLES

The following example restores disabled setup and hold arcs between pins CLK and TE on library cell SFF1.

```
pt_shell> remove_disable_timing -from CLK -to TE tech_lib/SFF1
```

The following example restores all disabled cell arcs from or to pin A on cell U1/U2.

```
pt_shell> remove_disable_timing U1/U2/A
```

SEE ALSO

report_lib (2), **set_disable_timing** (2), **report_lib** (2).

remove_distributed_hosts

Remove all the hosts added to the distributed hosts list.

SYNTAX

```
int remove_distributed_hosts  
-all
```

ARGUMENTS

DESCRIPTION

The **remove_distributed_hosts** command removes all the hosts that were added to the list of distributed hosts by the **add_distributed_hosts** command. The command will only remove all the hosts provided none of the slave processes have been launched. If any of the slave processes have been launched then the command will fail.

EXAMPLES

In the following example, the following distributed hosts are added platinum1, two 64 bit hosts platinum2, four 32 bit hosts 1 lsf queue, four 64 bit hosts All entries are then removed.

```
pt_shell> add_distributed_hosts -farm now -num_of_hosts 2 platinum1  
1  
pt_shell> add_distributed_hosts -farm now -num_of_hosts 4 platinum2 -32bit  
1  
pt_shell> add_distributed_hosts -farm lsf -submission_script "/bin/  
lsf_stub"  
-options { -R "tmp>300" -q} -num_of_hosts 4  
1  
pt_shell> remove_distributed_hosts  
1
```

SEE ALSO

add_distributed_hosts (2)

remove_drive_resistance

Removes drive resistance for input or inout ports.

SYNTAX

```
string remove_drive_resistance port_list  
list port_list
```

ARGUMENTS

port_list
Specifies a list of input or inout port names of the current design from which the drive values are to be removed.

DESCRIPTION

The **remove_drive_resistance** command removes the drive resistance value from one or more input or inout ports. This is equivalent to using **set_drive_resistance 0**. In fact, that is how this command works. So, if output ports are passed into **remove_drive_resistance**, a DES-003 message is issued, indicating that **set_drive_resistance** could not be performed on those ports.

EXAMPLES

```
pt_shell> set_drive_resistance 5 [get_ports {A B C}]  
1  
pt_shell> report_port -drive {A B C}
```

```
*****  
Report : port  
        -drive  
Design : counter  
*****
```

Input Port	Resistance		Transition	
	Rise	Fall	Rise	Fall
A	5.00	5.00	--	--
B	5.00	5.00	--	--
C	5.00	5.00	--	--

```
1  
pt_shell> remove_drive_resistance A  
1  
pt_shell> report_port -drive {A B C}
```

```
*****  
Report : port  
        -drive  
Design : counter
```

Input Port	Resistance		Transition	
	Rise	Fall	Rise	Fall

A	--	--	--	--
B	5.00	5.00	--	--
C	5.00	5.00	--	--
1				

SEE ALSO

`report_port (2), set_capacitance (2), set_drive_resistance (2).`

remove_driving_cell

Removes port driving cell information.

SYNTAX

```
string remove_driving_cell [-rise]
[-fall]
[-min]
[-max]
[-clock clock_name]
[-clock_fall]
port_list
```

```
string clock_name
list port_list
```

ARGUMENTS

-rise
Removes rise driving cell information.

-fall
Removes fall driving cell information.

-min
Removes min driving cell information.

-max
Removes max driving cell information.

-clock *clock_name*
Removes the driving cell set relative to the specified clock.

-clock_fall
Removes the driving cell relative to the falling edge of the clock. The default is the rising edge.

port_list
Provides a list of input or output ports.

DESCRIPTION

Removes driving cell information from the specified ports. The driving cell information is specified with the **set_driving_cell** command. The default is to remove all **-rise** and **-fall** information.

To see port drive information, use the **report_port -drive** command.

EXAMPLE

The following example removes all driving cell information for port IN2.

```
remove_driving_cell
468
```

```
pt_shell> remove_driving_cell IN2
```

SEE ALSO

```
report_port (2), set_driving_cell (2).
```

remove_fanout_load

Removes fanout load information from output ports in the current design.

SYNTAX

```
string remove_fanout_load port_list  
list port_list
```

ARGUMENTS

port_list
Specifies a list of output ports.

DESCRIPTION

Removes fanout load information specified by the **set_fanout_load** command. Fanout load for a net is the sum of **fanout_load** attributes for all input pins and output ports connected to the net. Output pins can have maximum fanout limits, specified in the library or with the **set_max_fanout** command. The **fanout_load** is used when checking max_fanout design values.

Use the **report_constraint** command with the **-max_fanout** option to show maximum fanout constraint evaluations. Use the **report_port** command with the **-design_rule** option to show port fanout load values.

EXAMPLES

The following example removes the fanout load on ports matching "OUT*".

```
pt_shell> remove_fanout_load "OUT*"
```

SEE ALSO

report_constraint (2), **report_port** (2), **set_fanout_load** (2), **set_max_fanout** (2), **set_min_fanout** (2).

remove_from_collection

Removes objects from a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection remove_from_collection  
base_collection  
xlcollectionbase_collection  
list      object_spec
```

ARGUMENTS

base_collection
Specifies the base collection to be copied to the result collection. Objects matching *object_spec* are removed from the result collection.

object_spec
Specifies a list of named objects or collections to remove. The object class of each element in this list must be the same as in the base collection. If the name matches an existing collection, the collection is used. Otherwise, the objects are searched for in the database using the object class of the base collection.

DESCRIPTION

The **remove_from_collection** command removes elements from a collection, creating a new collection.

If the base collection is homogeneous, any element of the *object_spec* that is not a collection is searched for in the database using the object class of the base collection. If the base collection is heterogeneous, then any element of the *object_spec* that is not a collection is ignored.

If nothing matches the *object_spec*, the resulting collection is a copy of the base collection. If everything in *base_collection* matches the *object_spec*, the result is the empty collection.

For background on collections and querying of objects, see the **collections** man page.

EXAMPLES

The following example from PrimeTime gets all input ports except "CLOCK".

```
pt_shell> set cPorts [remove_from_collection [all_inputs] CLOCK]  
{"in1", "in2"}
```

SEE ALSO

`add_to_collection(2)`, `collections(2)`, `query_objects(2)`.

remove_generated_clock

Removes generated clock objects from the current design.

SYNTAX

```
Boolean remove_generated_clock -all | clock_list  
list clock_list
```

ARGUMENTS

-all

Indicates that all generated clocks are to be removed.

clock_list

Specifies a list of names of generated clocks to be removed.

DESCRIPTION

This command removes from the current design generated clocks previously created using the **create_generated_clock** command. All attributes set on generated clocks (for example, **false_paths**, **multicycle_paths**) are also removed.

To display information about clocks and generated clocks in the design, use the **report_clock** command.

EXAMPLES

The following example removes the generated clock GEN1 from the design.

```
pt_shell> remove_generated_clock GEN1
```

The following example removes all generated clocks from the design.

```
pt_shell> remove_generated_clock -all
```

SEE ALSO

create_generated_clock (2), **report_clock** (2).

remove_host_options

Removes hosts options set using the **set_host_options** command.

SYNTAX

```
int remove_host_options  
[host_option_names]
```

list *host_option_names*

ARGUMENTS

host_option_names

This option specifies a list of named host options to remove.

DESCRIPTION

Given a list of host option names, removes those host options. The host option names specified must have been previously set using the **set_host_options** command. If no host option names are specified, all host options are removed. Regardless of whether host option names are specified or not, all processes will be shutdown when any host options are removed.

EXAMPLES

In the following example, the master process is running on the host named ptopt018 using a 64-bit binary and specifies five different sets of host options. The first host options, named "my_opts1", specifies 1 process with the same architecture as the master on the same host as the master. The second host options, named "my_opts2", specifies 2 processes using the 32-bit binary on the same host as the master but explicitly mentions the name of the master's host. The third host options, named "my_opts3", specifies 4 processes (with the same architecture as the master) on the host named ptopt010 using rsh to connect to ptopt010. The fourth host options named "my_opts4", specifies 5 processes (with the same architecture as the master) on an LSF farm requesting 500MB of memory on the compute servers. The fifth host options, named "my_opts5", specifies 2 processes (with the same architecture as the master) on a Grid farm requiring the jobs to be launched using the project named bnormal.

Note: After the **remove_host_options** command has been called all processes are shutdown and all the host options are removed.

Call the **set_host_options** command to specify the host options.

```
pt_shell> set_host_options -name my_opts1  
1  
pt_shell> set_host_options -name my_opts2 -32bit -num_processes 2 \  
ptopt018  
1  
pt_shell> set_host_options -name my_opts3 -num_processes 4 \  
-submit_command "/usr/bin/rsh -n" ptopt010
```

remove_host_options

```

1
pt_shell> set_host_options -name my_opts4 -num_processes 5 \
-submit_command "/lsf/bin/bsub -R rusage\[mem=500\]"
1
pt_shell> set_host_options -name my_opts5 -num_processes 2 \
-submit_command "/grid/bin/qsub -b y -P bnormal"
1

```

Call the **start_hosts** command to start the slave processes. Notice that each instance launched is assigned a number e.g. 1] Launched ...

```

pt_shell> start_hosts
1] Launched : ...
    ...
    Status : Forking successful
    Stdout  : Process group is (7199)
    Stderr  : **<<EMPTY>>**

2] Launched : ...
    ...
    Status : Forking successful
    Stdout  : Process group is (7233)
    Stderr  : **<<EMPTY>>**

```

```

...
...

```

```

-----
Distributed farm creation timeout : 21600 seconds
Waiting for 14 (of 14) distributed hosts (Wed Jul 29 09:07:07 2009)
Waiting for  9 (of 14) distributed hosts (Wed Jul 29 09:07:17 2009)
Waiting for  0 (of 14) distributed hosts (Wed Jul 29 09:07:27 2009)
-----

```

```

1

```

Call the **report_host_usage** command with the **-verbose** option after starting the hosts to report the host options created by the **set_host_options** command, as well as the real time data associated with the processes.

```

pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:07:42 2009
*****

```

Options	Host	32Bit	Num

Name	Processes				
my_opts1	**localhost**	N	1		
my_opts2	**ptopt018**	Y	2		
my_opts3	ptopt010	N	4		
my_opts4	>>farm<<	N	5		
my_opts5	>>farm<<	N	2		

Options Name	#	Host Name	Job ID	Process ID	Status
my_opts1	1	ptopt018	7199	7199	ONLINE
my_opts2	2	ptopt018	7233	7233	ONLINE
	3	ptopt018	7269	7269	ONLINE
my_opts3	4	ptopt010	8481	8481	ONLINE
	5	ptopt010	8558	8558	ONLINE
	6	ptopt010	8629	8629	ONLINE
	7	ptopt010	8716	8716	ONLINE
my_opts4	8	maddog108	627124	7718	ONLINE
	9	maddog108	627125	7715	ONLINE
	10	maddog108	627126	7717	ONLINE
	11	maddog108	627127	7716	ONLINE
	12	maddog108	627128	7714	ONLINE
my_opts5	13	peopf11	8998610	30080	ONLINE
	14	peopf1	8998611	11076	ONLINE

Options Name	Action	Command
my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

Call the **remove_host_options** command to remove all host options.

```
pt_shell> remove_host_options
Shutting down all slaves processes ...
Shutdown Process 3
Shutdown Process 1
Shutdown Process 2
Shutdown Process 9
Shutdown Process 5
Shutdown Process 4
Shutdown Process 6
Shutdown Process 8
Shutdown Process 7
Shutdown Process 10
Shutdown Process 12
Shutdown Process 11
```

```
Shutdown Process 14
Shutdown Process 13
1
pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:37:10 2009
*****
```

1

SEE ALSO

set_host_options (2), **report_host_usage** (2), **start_hosts** (2), **stop_hosts** (2)

remove_ideal_latency

Removes ideal latency values from the specified objects.

SYNTAX

```
int remove_ideal_latency
    [-rise] [-fall]
    [-min] [-max]
    object_list
```

```
list    object_list
```

ARGUMENTS

-rise

Specifies that only rise ideal latency should be removed. By default, both rise and fall ideal latency are removed.

-fall

Specifies that only fall ideal latency should be removed. By default, both rise and fall ideal latency are removed.

-min

Specifies that only minimum ideal latency should be removed. By default, both minimum and maximum ideal latency are removed.

-max

Specifies that only maximum ideal latency should be removed. By default, both minimum and maximum ideal latency are removed.

object_list

Specifies a list of ports or pins from where to remove ideal latency.

DESCRIPTION

Removes the user-specified ideal latency that was previously set by the **set_ideal_latency** command from specified objects in *object_list*.

You can remove selected portions of ideal latency by specifying applicable **-rise**, **-fall**, **-min** and **-max** options.

To list ideal latency values, use the **report_ideal_network** command.

EXAMPLES

The following example removes ideal latency from the output pin named Z of cell instance U1/U2/U3.

```
pt_shell> remove_ideal_latency {U1/U2/U3/Z}
```

remove_ideal_latency

The following example removes only rise ideal latency information from a port named A.

```
pt_shell> remove_ideal_latency -rise {A}
```

SEE ALSO

```
remove_ideal_network (2),  
remove_ideal_transition (2),  
report_ideal_network (2),  
report_timing (2),  
set_ideal_network (2),  
set_ideal_latency (2).
```

remove_ideal_network

Removes sources of ideal networks in the current design. Cells and nets in the transitive fanout of the specified objects are no longer treated as ideal.

SYNTAX

```
int remove_ideal_network object_list
list  object_list
```

ARGUMENTS

object_list
Specifies a list of ideal sources. The source objects may be ports or pins of leaf cells at any hierarchical level of the design. If nets are specified in the *object_list* option, then all of the nets' global driver ideal source pins that were set with the **-no_propagate** option are removed.

DESCRIPTION

This command removes the ideal network property from a set of ports or pins in the design that were previously marked as ideal sources using the **set_ideal_network** command. You specify only the source of the network. The ideal network is automatically updated by PrimeTime. The criteria for propagating the ideal property are detailed in the **set_ideal_network** man page.

All ideal networks are removed using the **reset_design** command.

EXAMPLES

The following example sets up an ideal network on objects in the design CLOCK_GEN and then removes part of the network:

```
prompt> current_design CLOCK_GEN
prompt> set_ideal_network {port1 port2}
prompt> vBremove_ideal_network port2
```

The following example creates an ideal network and then removes part of the ideal network.

```
pt_shell> current_design {TOP}
pt_shell> set_ideal_network {IN1 IN2}
pt_shell> remove_ideal_network {IN1}
```

The following example removes an ideal network that was set on a net.

```
pt_shell> set_ideal_network -no_propagate {netB}
Warning: Transferring ideal net attribute onto driver pin 'AN2/
Z' of net 'netB'. (UITE-450)
pt_shell> remove_ideal_network {netB}
```


SEE ALSO

```
remove_ideal_latency (2),  
remove_ideal_transition(2),  
report_ideal_network (2),  
reset_design (2),  
set_ideal_network(2).
```

remove_ideal_transition

Removes ideal transition values from the specified objects.

SYNTAX

```
int remove_ideal_transition
    [-rise] [-fall]
    [-min] [-max]
    object_list
```

list *object_list*

ARGUMENTS

-rise

Specifies that only rise ideal transition should be removed. By default, both rise and fall ideal transitions are removed.

-fall

Specifies that only fall ideal transition should be removed. By default, both rise and fall ideal transitions are removed.

-min

Specifies that only minimum ideal transition should be removed. By default, both minimum and maximum ideal transitions are removed.

-max

Specifies that only maximum ideal transition should be removed. By default, both minimum and maximum ideal transitions are removed.

object_list

Specifies a list of ports or pins from which to remove ideal transition.

DESCRIPTION

Removes the user-specified ideal transition that was previously set by the **set_ideal_transition** command from specified objects in the *object_list* option.

You can remove selected portions of ideal transition by specifying applicable **-rise**, **-fall**, **-min**, and **-max** options.

To list ideal transition values, use the **report_ideal_network** command.

EXAMPLES

The following example removes ideal transition from the output pin named Z of cell instance U1/U2/U3.

```
pt_shell> remove_ideal_transition {U1/U2/U3/Z}
```

remove_ideal_transition

The following example removes only rise ideal transition information from a port named A.

```
pt_shell> remove_ideal_transition -rise {A}
```

SEE ALSO

```
remove_ideal_latency (2),  
remove_ideal_network (2),  
report_ideal_network (2),  
report_timing (2),  
set_ideal_network (2),  
set_ideal_transition (2).
```

remove_input_delay

Removes input delay information from ports or pins.

SYNTAX

```
string remove_input_delay [-clock clock_name] [-clock_fall] [-level_sensitive] [-  
rise] [-fall] [-max] [-min] port_pin_list  
list clock_name  
list port_pin_list
```

ARGUMENTS

-clock *clock_name*
Relative clock; '' for no clock. Use this option to remove only input delay relative to one clock.

-clock_fall
Delay is relative to falling edge of clock.

-level_sensitive
Delay is from level-sensitive latch.

-rise
Removes rising input delay.

-fall
Removes falling input delay.

-max
Removes maximum input delay.

-min
Removes minimum input delay.

port_pin_list
Specifies a list of ports and pins.

DESCRIPTION

Removes input delay information from ports or pins in the current design. Input delay is specified with the **set_input_delay** command. To show input delays on ports, use **report_port -input_delay**. The default is to remove all input delay information in the *port_pin_list*.

EXAMPLES

The following example removes all input delay from ports IN*.

```
pt_shell> remove_input_delay [get_ports IN*]
```

The following example removes input delay relative to CLK rise edge from port

remove_input_delay

DATA[5].

```
pt_shell> remove_input_delay -clock CLK { DATA[5] }
```

SEE ALSO

```
report_port (2), set_input_delay (2), set_output_delay (2), remove_output_delay (2).
```

remove_input_noise

Removes input noise for a library pin or port.

SYNTAX

```
int remove_input_noise  
[-above]  
[-below]  
[-low]  
[-high]  
object_list  
  
list object_list
```

ARGUMENTS

-above
Removes the input noise for above ground or power rail noise analysis region.

-below
Removes the input noise for below ground or power rail noise analysis region.

-low
Removes the input noise for ground rail noise.

-high
Removes the input noise for power rail noise. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

This command removes the input noise information set by the `set_input_noise` command.

EXAMPLES

This example removes input noise information for the above the ground rail noise for pin A of library cell IV in the `lsi_10k` library:

```
pt_shell> remove_input_noise -above lsi_10k/IV/A
```

SEE ALSO

set_input_noise (2).

remove_lib

Removes one or more libraries from memory.

SYNTAX

```
string remove_lib -all libraries
list  libraries
```

ARGUMENTS

-all
Removes all libraries.

libraries
Provides a list of libraries to remove.

DESCRIPTION

This command removes a list of libraries. Either *libraries* or **-all** must be specified. For a library to be removed, none of its lib cells can be instantiated in any design, nor can any environment information (such as operating conditions or wire load models) be in use from the library. If this is the case, a message is issued and you must remove the design using **remove_design**, then use **remove_lib**.

EXAMPLES

The following command removes all the libraries read in.

```
pt_shell> remove_lib -all
Removing library '/u/libraries/cmos1.db:cmos1'...
1
```

The following command removes a library which was part of a max/min pair created by **set_min_library**.

```
pt_shell> remove_lib lib_ff
Removed max/min library relationship:
  Max: /u/libraries/lib_ss.db:lib_ss
  Min: /u/libraries/lib_ff.db:lib_ff
Removing library '/u/libraries/lib_ff.db:lib_ff'...
1
```

SEE ALSO

`list_libraries` (2), `list_designs` (2), `remove_design` (2), `set_min_library` (2).

remove_license

Removes a licensed feature.

SYNTAX

```
int remove_license feature_list
```

```
list feature_list
```

ARGUMENTS

feature_list

A list of features to remove. Refer to the *Synopsys Installation Guide* for a list of features supported by the current release. Or, determine from the key file all the features that are licensed at your site.

DESCRIPTION

Removes the specified licensed features from the features you currently obtain. The **remove_license** command is valid only when Network Licensing is enabled.

The **list_licenses** command provides a list of the features that you currently have checked out.

EXAMPLES

This example removes the STAMP Compiler and VHDL Compiler licenses.

```
pt_shell> remove_license {STAMP-Compiler VHDL-Compiler}
Checked in license 'Stamp-Compiler'
Checked in license 'VHDL-Compiler'
1
```

SEE ALSO

get_license (2), **license_users** (2), **list_licenses** (2).

remove_max_area

Removes the **max_area** attribute from the current design.

SYNTAX

```
int remove_max_area
```

ARGUMENTS

None.

DESCRIPTION

The **remove_max_area** command removes the **max_area** attribute from the current design. This attribute represents the target area of the design.

Use **set_max_area** to set the **max_area** attribute on the current design.

Use **report_constraint** to show area cost of the design.

EXAMPLES

The following example removes the target area from the current design.

```
pt_shell> remove_max_area
1
pt_shell> get_attribute [get_design [current_design]] max_area
Warning: Attribute 'max_area' does not exist on design 'TOP' (ATTR-3)
```

SEE ALSO

current_design (2), **get_attribute** (2), **set_max_area** (2), **report_constraint** (2), **reset_design** (2).

remove_max_capacitance

Removes maximum capacitance limits from pins, ports, clocks or designs.

SYNTAX

```
string remove_max_capacitance object_list  
list object_list
```

ARGUMENTS

object_list
Provides a list of pins, ports, clocks or designs from which to remove maximum capacitance limits.

DESCRIPTION

Removes maximum capacitance limits from pins, ports, clocks or designs. A maximum capacitance limit is specified with the **set_max_capacitance** command.

The **report_constraint -max_capacitance** command shows maximum capacitance constraint evaluations. The **report_port -design_rule** command shows port maximum capacitance limits. The **report_design** command shows the default maximum capacitance setting for the current design.

EXAMPLES

The following example removes the maximum capacitance limit on ports "OUT*".

```
pt_shell> remove_max_capacitance [get_ports "OUT*"]
```

The following example removes the maximum capacitance limit on the current design.

```
pt_shell> remove_max_capacitance [current_design]
```

SEE ALSO

current_design (2), **remove_min_capacitance** (2), **report_constraint** (2), **report_design** (2), **report_port** (2), **get_ports** (2), **set_max_capacitance** (2), **remove_max_fanout** (2), **remove_max_transition** (2).

remove_max_fanout

Removes maximum fanout limits from ports or designs.

SYNTAX

```
string remove_max_fanout object_list  
list object_list
```

ARGUMENTS

object_list
Lists the ports or designs from which to remove maximum fanout limits.

DESCRIPTION

Removes maximum fanout limits from ports or designs. A maximum fanout limit is specified with the **set_max_fanout** command.

The **report_constraint -max_fanout** command shows maximum fanout constraint evaluations. The **report_port -design_rule** command shows port maximum fanout limits. The **report_design** command shows the default maximum fanout setting for the current design.

EXAMPLES

The following example removes the maximum fanout limit on ports "OUT*".

```
pt_shell> remove_max_fanout [get_ports "OUT*"]
```

The following example removes the maximum fanout limit on the current design.

```
pt_shell> remove_max_fanout [current_design]
```

SEE ALSO

current_design (2), **remove_min_fanout** (2), **report_constraint** (2), **report_design** (2), **report_port** (2), **get_ports** (2), **set_max_fanout** (2), **set_fanout_load** (2), **set_max_capacitance** (2), **set_max_transition** (2).

remove_max_time_borrow

Removes time borrow limit for latches.

SYNTAX

```
string remove_max_time_borrow object_list  
list object_list
```

ARGUMENTS

object_list
Lists clocks, cells, data pins, or clock (enable) pins. If you specify a cell, all enable pins on that cell are affected.

DESCRIPTION

Removes maximum time borrow limit on objects. Time borrowing limits are specified with the **set_max_time_borrow** command. When the limit is removed, full time borrowing is allowed on level-sensitive latches.

To show time borrow limits, use **report_exceptions**.

EXAMPLES

The following example removes the maximum time borrow limit on clock PHI1.

```
pt_shell> remove_max_time_borrow [get_clocks PHI1]
```

The following example removes the maximum time borrow limit on all the pins of cells matching U1/latch*.

```
pt_shell> remove_max_time_borrow [get_cells U1/latch*]
```

SEE ALSO

report_exceptions (2), **set_max_time_borrow** (2).

remove_max_transition

Removes maximum transition limits from pins, ports, clocks or designs.

SYNTAX

```
string remove_max_transition object_list  
list object_list
```

ARGUMENTS

object_list
Lists the pins, ports, clocks or designs from which to remove maximum transition limits.

DESCRIPTION

Removes maximum transition limits from pins, ports, clocks or designs. A maximum transition limit is specified with the **set_max_transition** command.

The **report_constraint -max_transition** command shows maximum transition constraint evaluations. The **report_port -design_rule** command shows port maximum transition limits. The **report_design** command shows the default maximum transition setting for the current design.

EXAMPLES

The following example removes the maximum transition limit on ports "OUT*".

```
pt_shell> remove_max_transition [get_ports "OUT*"]
```

The following example removes the maximum transition limit on the current design.

```
pt_shell> remove_max_transition [current_design]
```

SEE ALSO

current_design (2), **remove_min_transition** (2), **report_constraint** (2), **report_design** (2), **report_port** (2), **get_ports** (2), **set_max_capacitance** (2), **set_max_fanout** (2), **set_max_transition** (2).

remove_min_capacitance

Removes minimum capacitance limits from ports or designs.

SYNTAX

```
string remove_min_capacitance object_list  
list object_list
```

ARGUMENTS

object_list
Lists the ports or designs from which to remove minimum capacitance limits.

DESCRIPTION

Removes minimum capacitance limits from ports or designs. A minimum capacitance limit is specified with the **set_min_capacitance** command.

The **report_constraint -min_capacitance** command shows minimum capacitance constraint evaluations. The **report_port -design_rule** command shows port minimum capacitance limits. The **report_design** command shows the default minimum capacitance setting for the current design.

EXAMPLES

The following example removes the minimum capacitance limit on ports "OUT*".

```
pt_shell> remove_min_capacitance [get_ports "OUT*"]
```

The following example removes the minimum capacitance limit on the current design.

```
pt_shell> remove_min_capacitance [current_design]
```

SEE ALSO

current_design (2), **remove_max_capacitance** (2), **report_constraint** (2), **report_design** (2), **report_port** (2), **get_ports** (2), **set_min_capacitance** (2), **set_max_capacitance** (2), **set_max_fanout** (2), **set_max_transition** (2).

remove_min_pulse_width

Removes a previously-specified minimum pulse width constraint from specified design objects.

SYNTAX

```
string remove_min_pulse_width [-low] [-high] [object_list]  
list object_list
```

ARGUMENTS

-low

Indicates that the minimum pulse width constraint is to be removed only for low clock signal levels. If you do not specify the **-low** or **-high** option, the constraint is removed for both low and high pulses of clock signals.

-high

Indicates that the minimum pulse width constraint is to be removed only for high clock signal levels. If you do not specify the **-low** or **-high** option, the constraint is removed for both low and high pulses of clock signals.

object_list

Specifies a list of clocks, cells, pins or ports in the current design for which the minimum pulse width constraint is to be removed. If you specify a cell, all pins on that cell are affected. If you do not specify any objects, the minimum pulse width check is removed from all objects in the current design.

DESCRIPTION

The **remove_min_pulse_width** command removes the pulse width check previously specified by **set_min_pulse_width** for clock signals in clock trees or at sequential devices.

To generate a report of pulse width constraints, use **report_constraint -min_pulse_width** or **report_min_pulse_width**.

The **reset_design** command removes all user-specified attributes from a design, including those set by **set_min_pulse_width**.

EXAMPLES

The following example removes a minimum pulse width requirement previously set for clock CK1.

```
pt_shell> remove_min_pulse_width [get_clocks CK1]
```

The following example removes a minimum pulse width requirement previously set for pin U1/Z.

```
pt_shell> remove_min_pulse_width U1/Z
```


SEE ALSO

`current_design (2)`, `report_constraint (2)`, `report_min_pulse_width (2)`,
`reset_design(2)`, `set_min_pulse_width (2)`.

remove_multi_scenario_design

Removes all multi scenario objects from memory and removes from disk all images generated by multi scenario analysis.

SYNTAX

boolean **remove_multi_scenario_design**

DESCRIPTION

The **remove_multi_scenario_design** command removes all multi scenario objects from pt_shell. The current_session and all scenarios are removed from memory. All nelist, baseline and current images generated during the multi scenario analysis are removed from disk.

EXAMPLES

The following example removes all multi scenario objects and removes all images from disk.

```
pt_shell> remove_multi_scenario_design  
1
```

SEE ALSO

report_multi_scenario_design (2),

remove_net

Removes nets from the current design.

SYNTAX

```
int remove_net net_list | -all  
list net_list
```

ARGUMENTS

-all

Indicates that all nets are to be removed from the current design. **-all** and *net_list* are mutually exclusive; you can specify only one.

net_list

Specifies a list of nets to be removed from the current design. **-all** and *net_list* are mutually exclusive; you can specify only one.

DESCRIPTION

The **remove_net** command removes specified nets, or all nets, from the current design. Like all other netlist editing commands, for **remove_net** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **remove_net** returns a 1 if successful and a 0 if unsuccessful.

Each net specified must be in scope; that is, at or below the current instance.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs

can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following example, nets are removed if their names begin with "new".

```
pt_shell> remove_net [get_nets new*]
Information: Removed net 'new_net1'. (NED-017)
Information: Removed net 'new_net2'. (NED-017)
1
```

SEE ALSO

size_cell (2), **swap_cell** (2), **write_changes** (2).

remove_noise_immunity_curve

Removes noise immunity curve for a library pin or port.

SYNTAX

```
int remove_noise_immunity_curve  
[-above]  
[-below]  
[-low]  
[-high]  
object_list  
  
list object_list
```

ARGUMENTS

-above
Removes the noise immunity curve for above ground or power rail noise analysis region.

-below
Removes the noise immunity curve for below ground or power rail noise analysis region.

-low
Removes the noise immunity curve for ground rail noise.

-high
Removes the noise immunity curve for power rail noise. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

This command removes the noise immunity curve information set by the `set_noise_immunity_curve` command.

EXAMPLES

This example removes noise immunity curve information for the above the ground rail noise for pin A of library cell IV in the lsi_10k library:

```
pt_shell> remove_noise_immunity_curve -above lsi_10k/IV/A
```

SEE ALSO

set_noise_immunity_curve (2).

remove_noise_lib_pin

Removes an equivalent noise library pin for a driver or load.

SYNTAX

```
int remove_noise_lib_pin  
pins
```

```
list pins
```

ARGUMENTS

pins

Specifies collection of pins for which the equivalent noise library pin set by **set_noise_lib_pin** needs to be remove.

DESCRIPTION

Given a collection of pins, this command allows to remove a noise library pin previously set as an equivalent in terms of library noise information.

EXAMPLES

This example removes any equivalent noise information on two inputs pins of the design to be the same as an input library pin:

```
pt_shell> remove_noise_lib_pin [get_pins {cell/pin1 cell/pin2}] \
```

SEE ALSO

set_noise_lib_pin (2), **update_noise** (2), **report_noise** (2), **report_noise_calculation** (2).

remove_noise_margin

Removes noise margin for a library pin or port.

SYNTAX

```
int remove_noise_margin  
[-above]  
[-below]  
[-low]  
[-high]  
object_list  
  
list object_list
```

ARGUMENTS

-above
Removes the noise margin for above ground or power rail noise analysis region.

-below
Removes the noise margin for below ground or power rail noise analysis region.

-low
Removes the noise margin for ground rail noise.

-high
Removes the noise margin for power rail noise. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

This command removes the noise margin information set by the `set_noise_margin` command.

EXAMPLES

This example removes noise margin information for the above the ground rail noise for pin A of library cell IV in the `lsi_10k` library:

```
pt_shell> remove_noise_margin -above lsi_10k/IV/A
```

SEE ALSO

set_noise_margin (2).

remove_operating_conditions

Removes operating conditions from current design, cells or ports.

SYNTAX

```
string remove_operating_conditions  
      [-object_list objects]
```

list *objects*

ARGUMENTS

-object_list *objects*
Specifies the cells or ports to remove operating conditions from. The command undoes operating conditions set by **set_operating_conditions**. Without **-object_list** all the operating conditions are removed from the design. Both leaf cells and hierarchical blocks are accepted with **-object-list**.

DESCRIPTION

Removes operating conditions settings from the current design, individual blocks, leaf cells, or ports.

EXAMPLES

This example removes all operating conditions from the current design. Both design-specific and cell-specific operating conditions are removed.

```
pt_shell> remove_operating_conditions
```

SEE ALSO

set_operating_conditions (2).

remove_output_delay

Removes output delay from output ports or pins.

SYNTAX

```
string remove_output_delay
[-clock clock_name]
[-clock_fall]
[-level_sensitive]
[-rise]
[-fall]
[-max]
[-min]
port_pin_list

string clock_name
list port_pin_list
```

ARGUMENTS

-clock *clock_name*
Relative clock; {""} for input delay relative to no clock.

-clock_fall
Removes the delay relative to falling edge of clock. If you specify *clock_name* without **-clock_fall**, the delay relative to rising edge of the clock is removed.

-level_sensitive
Removes level-sensitive output delay.

-rise
Removes rising output delay.

-fall
Removes falling output delay.

-max
Removes maximum output delay.

-min
Removes minimum output delay.

port_pin_list
Specifies a list of ports and pins. Each element in the list is either a collection of ports or pins, or a pattern which matches ports or pins on the current design.

DESCRIPTION

Removes output delay values for objects in the current design. Set output delay with

set_output_delay. By default, all output delays on each object in *port_pin_list* are removed. To restrict the removed output delay values, use **-clock**, **-clock_fall**, **-min**, **-max**, **-rise**, or **-fall**. To show output delays associated with ports, use **report_port -output_delay**.

EXAMPLES

The following example removes all output delay values from all output ports in the current design.

```
pt_shell> remove_output_delay [all_outputs]
```

The following example removes output maximum rise delay values from OUT1, OUT3, and BIDIR12.

```
pt_shell> remove_output_delay -max -rise { OUT1 OUT3 BIDIR12 }
```

The following example removes all output delays for port OUT1 relative to the falling edge of CLK2.

```
pt_shell> remove_output_delay -clock CLK2 -clock_fall OUT1
```

The following example removes all output delays not relative to any clock for port OUT2.

```
pt_shell> remove_output_delay -clock {} OUT2
```

SEE ALSO

all_outputs (2), **collections** (2), **report_port** (2), **set_output_delay** (2), **set_input_delay** (2).

remove_parasitic_corner

Removes a previously set parasitic corner in the presence of variation-aware parasitics.

SYNTAX

Boolean **remove_parasitic_corner**

ARGUMENTS

DESCRIPTION

The **remove_parasitic_corner** command removes the parasitic corner information previously set via **set_parasitic_corner** command. The command succeeds only if variation-aware parasitics were annotated and **set_parasitic_corner** is used to set the parasitic corner.

After using this command, all the parasitic attributes and analysis will use nominal values instead of corner values.

EXAMPLES

The following example removes the parasitics corner information set previously.

```
pt_shell> remove_parasitic_corner
```

SEE ALSO

set_parasitic_corner (2), **read_parasitics** (2), **report_annotated_parasitics** (2),

remove_path_group

Removes path_group objects.

SYNTAX

```
string remove_path_group  
      -all | path_group_list
```

```
list  path_group_list
```

ARGUMENTS

-all

Removes all path_groups in the current design.

path_group_list

Specifies path_group names to remove. Each element in the list is either a collection of path groups or a pattern matching path group names.

DESCRIPTION

Removes path groups in the current design. The **group_path** and **create_clock** commands create path groups. Path groups affect the cost function for optimization and influence the timing reports. If you remove a path group, any paths in that group are implicitly assigned to the default path group. To show path group information, use **report_path_group**.

EXAMPLES

The following example removes all path groups in the design.

```
pt_shell> remove_path_group -all
```

The following example removes path group "BUS1".

```
pt_shell> remove_path_group BUS1
```

SEE ALSO

collections (2), **create_clock** (2), **group_path** (2), **report_path_group** (2), **report_timing** (2).

remove_port_fanout_number

Removes fanout number information on ports.

SYNTAX

```
string remove_port_fanout_number port_list  
list port_list
```

ARGUMENTS

port_list

Specifies a list of ports. Each element in the list is either a collection of ports or a pattern that matches ports on the current design.

DESCRIPTION

Removes port **fanout_number** settings. The **set_port_fanout_number** command sets the number of external fanout pins on ports, which is used (along with wire load models) to calculate capacitance and resistance of nets. To show port fanout number information, use **report_port -wire_load**.

EXAMPLES

This example removes port fanout_number settings from ports OUT1*.

```
pt_shell> remove_port_fanout_number [get_ports OUT1*]
```

SEE ALSO

collections (2), **get_ports** (2), **report_port** (2), **set_port_fanout_number** (2), **set_wire_load_model** (2).

remove_power_groups

Remove the existing power groups.

SYNTAX

```
string remove_power_groups groups | -all  
list   groups
```

ARGUMENTS

groups

Specifies the groups to be removed.

-all

Indicates that all the user defined power groups will be deleted. Note that predefined power groups won't be deleted by this option.

DESCRIPTION

The **remove_power_groups** command removes some or all power groups that are previously created but no longer interested. Predefined power groups can be removed, but not with the -all option. The predefined power group names need to be explicitly specified.

EXAMPLES

In the following example, all the user defined power groups are removed.

```
pt_shell> remove_power_groups -all  
1
```

In the following example, the predefined clock_network power group is removed.

```
pt_shell> remove_power_groups clock_network  
1
```

SEE ALSO

create_power_group (2), **report_power_groups** (2), **get_power_group_objects** (2).

remove_propagated_clock

Removes a propagated clock specification.

SYNTAX

```
string remove_propagated_clock object_list  
list object_list
```

ARGUMENTS

object_list
Lists clocks, ports, or pins.

DESCRIPTION

Removes propagated clock specification from clocks, ports, or pins in the current design. The **set_propagated_clock** command specifies that delays be propagated through the clock network to determine latency at register clock pins. If this is not specified, ideal clocking is assumed. Ideal clocking means clock networks have a user specified latency (set by the **set_clock_latency** command) or zero latency, by default. Propagated clock latency is normally used for post layout, after final clock tree generation.

Ideal clock latency provides an estimate of the clock tree for prelayout. You can also use **set_clock_latency** to specify an ideal latency, which overrides the propagated clock specification for an object.

For information on propagated clock attributes on clocks, see the **report_clock** command man page.

EXAMPLES

The following example removes propagated clock specifications from all clocks in the design.

```
pt_shell> remove_propagated_clock [all_clocks]
```

SEE ALSO

report_clock (2), **set_clock_latency** (2), **set_propagated_clock** (2).

remove_pulse_clock_max_transition

Removes maximum transition limits from pulse clock network and input of pulse generator.

SYNTAX

```
string remove_pulse_clock_max_transition object_list
[-rise][-fall]
[-transitive_fanout]
object_list

list object_list
```

ARGUMENTS

-rise
Removes the rise transition constraint.

-fall
Removes falling transition constraint.

-transitive_fanout
Removes the constraint from the transitive fanout of the pulse generator. If this option is not specified, then the constraint is removed from the input of pulse generator.

object_list
Lists the pulse generator cell, pulse generator lib cell, clock and design from which to remove maximum pulse clock transition limits.

DESCRIPTION

Removes maximum pulse clock transition limits from the specified objects. A maximum transition limit is specified with the **set_pulse_clock_max_transition** command. This command can change the constraint on the object, which has conflicting constraints due to overlap. Options **-rise** and **-fall** can be used to remove only the rising or the falling transition respectively. If neither **-rise** or **-fall** is specified, both rise and fall transition limits are removed. **-transitive_fanout** option needs to be specified to remove pulse clock maximum transition constraint from the transitive fanout of pulse generators.

EXAMPLES

The following example removes the pulse clock maximum transition limit from the input of all the cells corresponding to lib cell pulse_rise_high in library celllib.

```
pt_shell> remove_pulse_clock_max_transition {"celllib/pulse_rise_high"}
```

The following example removes maximum transition limit from all the pulse clock networks in the clock network clk


```
pt_shell> remove_pulse_clock_max_transition -transitive_fanout [get_clocks  
clk]
```

SEE ALSO

```
current_design (2), set_pulse_clock_max_transition (2),  
report_pulse_clock_max_transition (2), report_constraint (2).
```

remove_pulse_clock_max_width

Removes maximum pulse width limits from pulse clock network.

SYNTAX

```
string remove_pulse_clock_max_width  
[-transitive_fanout]  
object_list
```

```
list object_list
```

ARGUMENTS

-transitive_fanout

Remove the constraints from the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

object_list

Lists the pulse generator cell, pulse generator lib cell, clock and design from which to remove maximum pulse clock width limits.

DESCRIPTION

Removes maximum pulse clock width limits from the specified objects. A maximum width limit is specified with the **set_pulse_clock_max_width** command. This command can change the constraint on the object, which has conflicting constraints due to overlap.

EXAMPLES

The following example removes maximum width limit from all the pulse clock networks in the design.

```
pt_shell> remove_pulse_clock_max_width [current_design]
```

SEE ALSO

current_design (2), **set_pulse_clock_max_width** (2), **report_pulse_clock_max_width** (2), **report_constraint** (2).

remove_pulse_clock_min_transition

Removes minimum transition limits from input of pulse generator.

SYNTAX

```
string remove_min_transition object_list
[-rise][-fall]
object_list

list object_list
```

ARGUMENTS

-rise

Removes the rise transition constraint.

-fall

Removes falling transition constraint.

object_list

Lists the pulse generator cell, pulse generator lib cell, clock and design from which to remove minimum pulse clock transition limits.

DESCRIPTION

Removes minimum pulse clock transition limits from the input of specified objects. If clock or design is specified, the minimum transition limit is removed from the input of all the pulse generators in the clock network or design respectively. A minimum transition limit is specified with the **set_pulse_clock_min_transition** command. This command can change the constraint on the object, which has conflicting constraints due to overlap. Options **-rise** and **-fall** can be used to remove only the rising or the falling transition respectively. If neither **-rise** or **-fall** is specified, both rise and fall transition limits are removed.

EXAMPLES

The following example removes a minimum transition limit from input of all the cells corresponding to lib cell pulse_rise_high in library celllib.

```
pt_shell> remove_pulse_clock_min_transition {"celllib/pulse_rise_high"}
```

SEE ALSO

current_design (2), **set_pulse_clock_min_transition** (2),
report_pulse_clock_min_transition (2), **report_constraint** (2).

remove_pulse_clock_min_width

Removes minimum pulse width limits from pulse clock network.

SYNTAX

```
string remove_pulse_clock_min_width  
[-transitive_fanout]  
object_list
```

```
list object_list
```

ARGUMENTS

-transitive_fanout

Remove the constraints from the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

object_list

Lists the pulse generator cell, pulse generator lib cell, clock and design from which to remove minimum pulse clock width limits.

DESCRIPTION

Removes minimum pulse clock width limits from the specified objects. A minimum width limit is specified with the **set_pulse_clock_min_width** command. This command can change the constraint on the object, which has conflicting constraints due to overlap.

EXAMPLES

The following example removes minimum width limit from all the pulse clock networks in the clock network clk.

```
pt_shell> remove_pulse_clock_min_width [get_clocks clk]
```

SEE ALSO

current_design (2), **set_pulse_clock_min_width** (2), **report_pulse_clock_min_width** (2), **report_constraint** (2).

remove_qtm_attribute

Removes an attribute setting from the QTM object.

SYNTAX

```
string remove_qtm_attribute -class class_name attr_name object_names  
string class_name  
string attr_name  
list object_names
```

ARGUMENTS

-class *class_name*
Specifies the class of the QTM object(s) to remove attribute from. Allowed values are lib, lib_cell, or lib_pin.

attr_name
Specifies the name of the attribute.

object_names
Specifies the names of the objects from which to remove the named attribute. Each element in the list must be a name to identify an object in the specified class. The object names should not be specified when the object class is either lib or lib_cell. It must be specified when the class is lib_pin, in which case the object names should be the names of ports already defined for the QTM.

DESCRIPTION

The **remove_qtm_attribute** command removes attributes you set with **set_qtm_attribute**.

You can remove either application attributes or user attributes you defined for QTM object with command **define_qtm_attribute**.

EXAMPLES

The following example defines a boolean attribute 'is_XYZ' for QTM cell, then sets the value on the QTM cell under construction. Finally, it removes the attribute from the cell.

```
pt_shell> define_qtm_attribute -type boolean -class lib_cell "is_XYZ"  
pt_shell> set_qtm_attribute -class lib_cell "is_XYZ" "true"  
pt_shell> remove_qtm_attribute -class lib_cell "is_XYZ"
```

The following example defines a user attribute 'my_float_attr' for all the QTM port objects, then set the value for 2 QTM ports, Finally removes the attribute from one of the port.

```
pt_shell> define_qtm_attribute -type float -class lib_pin "my_float_attr"
```

```
pt_shell> set_qtm_attribute -class lib_pin "my_float_attr" 100.0 {IN, OUT}  
pt_shell> remove_qtm_attribute -class lib_pin "my_float_attr" {OUT}
```

SEE ALSO

`define_qtm_attribute` (2), `set_qtmuser_attribute` (2), `define_user_attribute` (2).

remove_rail_voltage

Removes power rail voltage that was set by the **set_rail_voltage** command on cells.

SYNTAX

```
int remove_rail_voltage cell_list
```

```
list cell_list
```

ARGUMENTS

cell_list

Specifies a list of cells from which to remove rail voltages.

DESCRIPTION

Removes power rail voltage that was set by the **set_rail_voltage** command on cells. If dynamic components of rail voltage have been specified they will also be removed along with the total rail voltage. This command "undoes" the voltages specified by **set_rail_voltage**. It can be applied to both leaf cells and hierarchical blocks.

EXAMPLES

The following example removes rail voltages from a cell named *h1/u3*.

```
pt_shell> remove_rail_voltage [get_cells {h1/u3}]  
1
```

SEE ALSO

set_operating_conditions (2), **set_rail_voltage** (2).

remove_resistance

Removes resistance on nets.

SYNTAX

```
string remove_resistance net_list  
list net_list
```

ARGUMENTS

net_list
Specifies a list of nets in the current design, whose resistances are removed.

DESCRIPTION

Specifies that the lumped resistance annotated on the list of nets must be removed. PrimeTime will revert to using the resistance from detailed parasitics (set using **read_parasitics**), if they exist. If not the internally estimated net resistance will be used.

Annotated resistances can also be removed on the full design by doing a *reset_design*.

EXAMPLES

The following example removes the annotated resistance on net w23.

```
pt_shell> remove_resistance [get_nets "w23"]
```

SEE ALSO

```
all_outputs (2), current_design (2), set_resistance (2), report_net (2),  
report_internal_loads (2), report_port (2), reset_design (2), set_drive (2);  
set_wire_load (2);
```


remove_scenario

Removes a scenario in multi scenario analysis.

SYNTAX

```
remove_scenario scenario list
```

ARGUMENTS

scenario list

A list of unique strings used to identify each scenario.

DESCRIPTION

The **remove_scenario** command removes the specified scenario from memory. To determine the current scenarios that exist, execute the following:

```
report_multi_scenario_design -scenarios.  
1
```

EXAMPLES

In the following example, two scenarios named scen1 and scen2 are removed.

```
pt_shell> remove_scenario {scen1 scen2}  
1
```

SEE ALSO

create_scenario(2), report_multi_scenario_design (2)

remove_setup_hold_pessimism_reduction

Removes the optimization constraints for setup-hold pessimism reduction.

SYNTAX

```
remove_setup_hold_pessimism_reduction  
[-setup_cutoff]  
[-hold_cutoff]
```

ARGUMENTS

```
-setup_cutoff  
    Reset setup_cutoff_slack value to negative INFINITY.  
  
-hold_cutoff  
    Reset hold_cutoff_slack value to negative INFINITY.
```

DESCRIPTION

Remove the optimization constraints for setup-hold pessimism reduction (SHPR). Option **setup_cutoff** reset the setup_cutoff slack to negative INFINITY. Option **hold_cutoff** reset the setup_cutoff slack to negative INFINITY. Command without options disables SHPR optimization.

EXAMPLES

The following example resets the setup_cutoff to negative INFINITY:

```
pt_shell> remove_setup_hold_pessimism_reduction -setup  
1
```

The following example resets the hold_cutoff to negative INFINITY:

```
pt_shell> remove_setup_hold_pessimism_reduction -hold  
1
```

The following example resets both hold_cutoff and setup_cutoff to negative INFINITY:

```
pt_shell> remove_setup_hold_pessimism_reduction -hold -setup  
1
```

The following example disable SHPR optimization:

```
pt_shell> remove_setup_hold_pessimism_reduction  
1
```

SEE ALSO

`set_setup_hold_pessimism_reduction` (2)

remove_si_aggressor_exclusion

Removes the exclusive groups set by the `set_si_aggressor_exclusion` command.

SYNTAX

```
int remove_si_aggressor_exclusion [-rise] [-fall] [-all] [anets]
list anets
```

ARGUMENTS

`-rise`

Removes the exclusive group set for the aggressor nets *anets* in the *rise* direction. If neither the *-rise* nor the *-fall* options are specified, the exclusive groups of the aggressor nets *anets* in both *-rise* and *-fall* directions are removed.

`-fall`

Removes the exclusive group set for the aggressor nets *anets* in the *fall* direction. If neither the *-rise* nor the *-fall* options are specified, the exclusive groups of the aggressor nets *anets* in both *-rise* and *-fall* directions are removed.

`-all`

Removes all the exclusive groups set on the design.

anets

Specifies the list of nets belonging to one exclusive group that are to be removed.

DESCRIPTION

The **remove_si_aggressor_exclusion** removes the effect of command **set_si_aggressor_exclusion** that was set on the aggressor nets. Only those groups that have been set can be removed. A warning message is issued if you are trying to remove a group that has not been set.

These exclusive commands are independent of parasitics, so they can be applied even before reading in parasitics.

Note that application of exclusive aggressors are not done incrementally, next `update_timing` and `update_noise` would be a full update.

You can use the **report_si_aggressor_exclusion** command to check whether the **remove_si_aggressor_exclusion** command was applied to the exclusive groups.

EXAMPLES

The following example shows how to remove the exclusion on nets *SCAN_LOGIC** for rise direction.

```
pt_shell> set_si_aggressor_exclusion [get_nets SCAN_LOGIC*] -rise
```

```
remove_si_aggressor_exclusion
```

```
1
pt_shell> remove_si_aggressor_exclusion [get_nets SCAN_LOGIC*]
1
```

The following example shows how to remove all exclusive groups set on the design.

```
pt_shell> remove_si_aggressor_exclusion -all
1
```

SEE ALSO

```
set_si_aggressor_exclusion (2), report_si_aggressor_exclusion (2),
report_delay_calculation (2), report_noise_calculation (2),
si_analysis_logical_correlation_mode (3), set_si_delay_analysis (2),
set_si_noise_analysis (2),
```

remove_si_delay_analysis

Removes the effect of the **set_si_delay_analysis** command.

SYNTAX

```
int remove_si_delay_analysis  
[-reselect rnets]  
[-ignore_arrival inets]  
[-victims vnets]  
[-aggressors anets]  
[-rise]  
[-fall]  
[-min]  
[-max]  
[-all]  
  
list rnets  
list inets  
list vnets  
list anets
```

ARGUMENTS

-reselect *rnets*

Removes the effect of using the **set_si_delay_analysis** command with its **-reselect** option. This **-reselect** option does not have control nets reselected based on reselection criteria. You cannot use this option with the **-ignore_arrival**, **-victims** and **-aggressors** options.

-ignore_arrival *inets*

Removes the effect of using the **set_si_delay_analysis** command with its **-ignore_arrival** option. You cannot use this option with the **-reselect**, **-victims** and **-aggressors** options.

-victims *vnets*

Removes the effect of using the **set_si_delay_analysis** command with its **-victims** option. When you use the **-victims** option with **-aggressors** option, the command restores the pair-wise relationship. However, when this option **-victims** is used to remove the effect set by the command **set_si_delay_analysis** with the option **-aggressors**, it does not remove any exclusion on the nets. You cannot use this **-victims** option with the **-reselect** or **-ignore_arrival** option.

-aggressors *anets*

Removes the effect of the **set_si_delay_analysis** command with its **-aggressors** option. When you use the **-aggressors** option with the **-victims** option, the command restores the pair-wise relationship. However, when this option **-aggressors** is used to remove the effect set by the command **set_si_delay_analysis** with the option **-victims**, it does not remove any exclusion on the nets. You cannot use this **-aggressors** option with the **-reselect** or **-ignore_arrival** option.

-rise
Removes the effect of **set_si_delay_analysis -exclude -rise** command.

-fall
Removes the effect of **set_si_delay_analysis -exclude -fall** command.

-min
Removes the effect of **set_si_delay_analysis -exclude -min** command.

-max
Removes the effect of **set_si_delay_analysis -exclude -max** command.

-all
Removes all the effects of **set_si_delay_analysis** command on all the nets.

DESCRIPTION

Removes the effect of the **set_si_delay_analysis** command.

The **set_si_delay_analysis** command allows you to exclude nets from crosstalk analysis, overriding net reselection methods, or setting some net as infinite window as both aggressor and victim . The **remove_si_delay_analysis** command removes such overrides.

remove_si_delay_analysis -reselect rnets removes the user-based reselection from *rnets*. It does not have any effect on nets that are reselected by the reselection criteria. So even after removing the user-forced reselection, the net can still be reselected due to the reselection criteria. If it is applied to a net that you have not already reselected, this command returns success without doing anything.

The **remove_si_delay_analysis** command returns a **1** if successful and a **0** if unsuccessful. If remove command is used to remove a effect that has not been set, warning message **XTALK-107** is issued.

To view the result of this command, use the **report_si_delay_analysis** command.

EXAMPLES

In the following example, all the nets named *CLK_NET_** are returned to their original state in crosstalk analysis.

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets CLK_NET_*]
1
```

```
pt_shell> remove_si_delay_analysis -victims [get_nets CLK_NET_*]
1
```

However in the following examples, none of the nets named *REG_NET_** are returned to their original state in crosstalk analysis.

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets REG_NET_*]
1
```

```
pt_shell> remove_si_delay_analysis -aggressors [get_nets REG_NET_*]
Warning: Cannot remove an effect that was not set on net(s) REG_NET_1. (XTALK-107)
Warning: Cannot remove an effect that was not set on net(s) REG_NET_2. (XTALK-107)
1
```

```
pt_shell> remove_si_delay_analysis -aggressors [get_nets REG_NET_*]
-victims [get_nets CLK_NET]
1
```

If there is a VIC_NET with aggressors AGG_NET_* and you had set pairwise exclusion on the nets with the list of its aggressors, the AGG_NET_* nets cannot be returned to their original state by using only **-aggressors** option. For example,

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets VIC_NET]
-aggressors [get_nets AGG_NET*]
1
```

```
pt_shell> remove_si_delay_analysis -aggressors [get_nets AGG_NET*]
1
```

In order to reset the pairwise exclusion on the nets VIC_NET and AGG_NET_*, the pairwise reset should be used as shown below

```
pt_shell> remove_si_delay_analysis -victims [get_nets VIC_NET]
-aggressors [get_nets AGG_NET*]
1
```

To verify if the exclusion was removed on the nets, use the **report_si_delay_analysis** with the **-excluded** option.

SEE ALSO

read_parasitics (2), **set_si_delay_analysis** (2); **report_si_delay_analysis** (2);
si_enable_analysis (3).

remove_si_delay_disable_statistical

Removes the effect of the **set_si_delay_disable_statistical** command.

SYNTAX

```
int remove_si_delay_disable_statistical  
dnets  
  
list dnets
```

ARGUMENTS

dnets

A list of nets for which the effect of the **set_si_delay_disable_statistical** command is removed.

DESCRIPTION

Removes the effect of the **set_si_delay_disable_statistical** command.

The **set_si_delay_disable_statistical** command allows you to disable the statistical analysis for *dnets* if selected in composite aggressor group for crosstalk analysis.

The **remove_si_delay_disable_statistical** command removes such effect.

EXAMPLES

The following example shows how to put net back into statistical analysis when considered as a composite aggressor.

```
pt_shell> remove_si_delay_disable_statistical [get_nets LOGIC1]  
1
```

SEE ALSO

set_si_delay_disable_statistical (2), **report_si_delay_analysis** (2).

remove_si_noise_analysis

Removes the effect of the **set_si_noise_analysis** command.

SYNTAX

```
int remove_si_noise_analysis  
[-ignore_arrival inets]  
[-victims vnets]  
[-aggressors anets]  
[-above]  
[-below]  
[-low]  
[-high]  
[-all]  
  
list inets  
list vnets  
list anets
```

ARGUMENTS

-ignore_arrival *inets*

Removes the effect of using the **set_si_noise_analysis** command with its **-ignore_arrival** option. You cannot use this option with the **-victims** or **-aggressors** option.

-victims *vnets*

Removes the effect of using the **set_si_noise_analysis** command with its **-victims** option. When this option **-victims** is used to remove the effect set by the command **set_si_noise_analysis** with the option **-aggressors**, it does not remove any exclusion on the nets. However, when you use the **-victims** option with **-aggressors** option, the command restores the pair-wise relationship. You cannot use this **-victims** option with the **-ignore_arrival** option.

-aggressors *anets*

Removes the effect of the **set_si_noise_analysis** command with its **-aggressors** option. When this option **-aggressors** is used to remove the effect set by the command **set_si_noise_analysis** with the option **-victims**, it does not remove any exclusion on the nets. However, when you use the **-aggressors** option with the **-victims** option, the command restores the pair-wise relationship. You cannot use this **-aggressors** option with the **-ignore_arrival** option.

-above

Removes the effect of **set_si_noise_analysis -exclude -above** command.

-below

Removes the effect of **set_si_noise_analysis -exclude -below** command.

-low

Removes the effect of **set_si_noise_analysis -exclude -low** command.

-high

Removes the effect of **set_si_noise_analysis -exclude -high** command.

-all

Removes all the effects of **set_si_noise_analysis** command on all the nets.

DESCRIPTION

Removes the effect of the **set_si_noise_analysis** command.

The **set_si_noise_analysis** command allows you to exclude nets from noise analysis, or set some net as infinite window as aggressor. The **remove_si_noise_analysis** command removes such overrides.

The **remove_si_noise_analysis** command returns a **1** if successful and a **0** if unsuccessful. If remove command is used to remove a effect that has not been set, warning message XTALK-107 is issued.

EXAMPLES

In the following example, all the nets named *CLK_NET_** are returned to their original state in noise analysis.

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets CLK_NET]
1
```

```
pt_shell> remove_si_noise_analysis -victims [get_nets CLK_NET*]
1
```

However in the following example, none of the nets named *REG_NET_** are returned to their original state in noise analysis.

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets REG_NET_*]
1
```

```
pt_shell> remove_si_noise_analysis -aggressors [get_nets REG_NET_*]
Warning: Cannot remove an effect that was not set on net(s) REG_NET_1. (XTALK-107)
Warning: Cannot remove an effect that was not set on net(s) REG_NET_2. (XTALK-107)
1
```

```
pt_shell> remove_si_noise_analysis -aggressors [get_nets REG_NET_*]
-victims [get_nets CLK_NET]
Warning: Cannot remove an effect that was not set on net(s) REG_NET_1
CLK_NET. (XTALK-107)
Warning: Cannot remove an effect that was not set on net(s) REG_NET_2
CLK_NET. (XTALK-107)
1
```

If there is a *VIC_NET* with aggressors *AGG_NET_** and you had set pairwise exclusion on the nets with the list of its aggressors, the *AGG_NET_** nets cannot be returned to their original state in noise analysis. For example,

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets VIC_NET]
```

```
-aggressors [get_attribute -class net [get_nets VIC_NET] aggressors]  
1
```

```
pt_shell> remove_si_noise_analysis -aggressors [get_nets AGG_NET*]  
Warning: Cannot remove an effect that was not set on net(s)  
AGG_NET_1. (XTALK-107)  
Warning: Cannot remove an effect that was not set on net(s)  
AGG_NET_2. (XTALK-107)  
1
```

In order to reset the pairwise exclusion on the nets VIC_NET and AGG_NET_*, the pairwise reset should be used as shown below

```
pt_shell> remove_si_noise_analysis -victims [get_nets VIC_NET]  
-aggressors [get_nets AGG_NET*]  
1
```

To verify if the exclusion was removed on the nets, use the **report_si_noise_analysis** with the **-excluded** option.

SEE ALSO

```
read_parasitics (2), set_si_noise_analysis (2), report_si_noise_analysis (2);  
si_enable_analysis (3).
```

remove_si_noise_disable_statistical

Removes the effect of **set_si_noise_disable_statistical** command.

SYNTAX

```
int remove_si_noise_disable_statistical  
dnets  
  
list dnets
```

ARGUMENTS

dnets
A list of nets for which the effect of command **set_si_noise_disable_statistical** is removed.

DESCRIPTION

Removes the effect of the **set_si_noise_disable_statistical** command.

The **set_si_noise_disable_statistical** command allows you to disable the statistical analysis for *dnets* if selected in composite aggressor group for noise analysis.

The **remove_si_noise_disable_statistical** command removes such effect.

EXAMPLES

The following example shows how to put net back into statistical analysis when considered as a composite aggressor.

```
pt_shell> remove_si_noise_disable_statistical [get_nets LOGIC1]  
1
```

SEE ALSO

set_si_noise_disable_statistical (2), **report_si_noise_analysis** (2).

remove_steady_state_resistance

Removes steady state resistance for a library pin or port.

SYNTAX

```
int remove_steady_state_resistance  
[-above]  
[-below]  
[-low]  
[-high]  
object_list  
  
list object_list
```

ARGUMENTS

-above
Removes the steady state resistance for above ground or power rail noise analysis region.

-below
Removes the steady state resistance for below ground or power rail noise analysis region.

-low
Removes the steady state resistance for ground rail noise.

-high
Removes the steady state resistance for power rail noise. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

This command removes the steady state resistance information set by the `set_steady_state_resistance` command.

EXAMPLES

This example removes steady state resistance information for the above the ground rail noise for pin A of library cell IV in the lsi_10k library:

```
pt_shell> remove_steady_state_resistance -above lsi_10k/IV/A
```

SEE ALSO

set_steady_state_resistance (2).

remove_user_attribute

Removes a user attribute from an object.

SYNTAX

```
string remove_user_attribute [-quiet] [-class class_name] object_spec attr_name  
string class_name  
list object_spec  
string attr_name
```

ARGUMENTS

-quiet

Does not report any messages.

-class *class_name*

If *object_spec* is a name, this is its class. Allowable values are design, port, cell, pin, net, lib, lib_cell, or lib_pin.

object_spec

Shows objects from which to remove the attribute. Each element in the list is either a collection or a pattern which combines with the *class_name* to find the objects.

attr_name

Provides the name of the attribute.

DESCRIPTION

The **remove_user_attribute** command removes attributes you set with **set_user_attribute**.

You cannot remove application attributes with this command. Each application attribute that can be removed has a command dedicated to it. For example, the **fanout_load** attribute is removed with the **remove_fanout_load** command.

EXAMPLES

This example defines an attribute 'X' for cells then sets the value on all cells in this level of the hierarchy. Finally, the example removes the attribute from one cell.

```
pt_shell> define_user_attribute -type int -class cell X  
pt_shell> set_user_attribute [get_cells *] X 30  
Set attribute 'X' on 'i1'  
Set attribute 'X' on 'i2'  
pt_shell> remove_user_attribute [get_cells i1] X  
Removed attribute 'X' from 'i1'
```

SEE ALSO

`collections` (2), `define_user_attribute` (2), `get_attribute` (2), `list_attributes` (2), `set_user_attribute` (2), `report_attribute` (2).

remove_user_sensitization

Report user sensitization of an instance or library arc for write_spice_deck output.

SYNTAX

```
int remove_user_sensitization  
[-analysis_type [rise | fall | high | low]]  
[-arc arcs_list]  
[-library library_name]  
[-design design_name]
```

```
list arcs_list  
string library_name  
string design_name
```

ARGUMENTS

```
[-analysis_type [rise | fall | high | low]]  
    Specifies the final state of the output pin of timing or library arc.
```

```
[-arc arcs_list]  
    Specifies a list timing or library arcs with their annotated user sensitization to be removed. You can get this argument with the get_timing_arcs or get_timing_lib_arc comments.
```

```
[-library library_name]  
    Specifies a list of input pins of the gate that contains the arc. It must includes the input pin of the arc in the arcs_list. The sensitization vectors/sequences are applied to these pins to achieve the desirable state indicated in the -analysis_type options.
```

```
[-design design_name]  
    Specifies the sensitization vectors/sequences to sensitize the arc. It is expressed as a list of 1, 0, r, and f. They stand for the states of logic one, logic zero, rising, and falling. The number of stats must be a multiple of the number of pin in the -event_pins options.
```

DESCRIPTION

It removes user sensitization of a library, a design, an instance arc collection, or a library arc collection.

EXAMPLES

Set, report and report of sensitization of a MUX with Falling S to rising X pin.

```
pt_shell> set arcs_1 [ get_timing_arcs -from inp_victim/S -to inp_victim/X ]  
pt_shell> set_user_sensitization -analysis_type rise -event_pin { D0 D1 S } \  
?      -event_states { r f 1 1 0 f } $arcs_1  
*****  
Report : set_user_sensitization
```

```

    -analysis_type rise
    -event_step 0.05
    -event_pins D0 D1 S
    -event_states R F 1 1 0 F
    timing_arc S -> X
    when = D0 * !(D1)
    sense = negative_unate
Design : SDN_MUX2_1_X_S
*****

```

```

pt_shell> report_user_sensitization -analysis_type rise -arc $arcs_1
User sensitization of timing arc inp_victim/S -> inp_victim/X
    Sense = negative_unate, when = (D0 * !(D1)), analysis_type = rise.
    Minimum transition separation time = 0.05 (LU).
    Event pin : inp_victim/D0 inp_victim/D1 inp_victim/S
    States : R F 1
             1 0 F
Information: 1 user sensitization reported. (SPICE-118)

pt_shell> remove_user_sensitization -analysis_type rise -arc $arcs_1
Information: 1 user sensitization removed. (SPICE-118)

```

SEE ALSO

```
write_spice_deck (2); set_user_sensitization (2); report_user_sensitization (2);
```

remove_variation

Removes one or more variations.

SYNTAX

```
int remove_variation  
[-all]  
[variation_list]  
collection variation_list
```

ARGUMENTS

-all
Remove all variations. At least one of *-all* and *variation_list* must be specified.

variation_list
Delete these variations.

DESCRIPTION

Removes the given variations completely and reset their associations with all of their timing objects.

EXAMPLES

The following example creates a variation, sets the variation on the current design, and then removes the variation.

```
pt_shell> create_variation -name Lfab -parameter_name len -type normal -  
values {0 1}]  
_sel22  
pt_shell> set_variation [get_variations Lfab]  
_sel23  
pt_shell> remove_variation [get_variations Lfab]  
1
```

SEE ALSO

`create_variation(2)`, `set_variation(2)`, `reset_variation(2)`.

remove_wire_load_min_block_size

Removes the minimum block size for automatic wire load selection.

SYNTAX

```
int remove_wire_load_min_block_size
```

ARGUMENTS

None.

DESCRIPTION

Removes the minimum block area for automatic wire load selection in the current design, set by **set_wire_load_min_block_size**.

EXAMPLES

The following example removes the previously-set minimum block size.

```
pt_shell> remove_wire_load_min_block_size
```

SEE ALSO

```
report_wire_load(2), set_wire_load_min_block_size(2),  
set_wire_load_selection_group(2); auto_wire_load_selection(3).
```

remove_wire_load_model

Removes wire load model from designs, hierarchical cells, or ports.

SYNTAX

```
string remove_wire_load_model  
[object_list]
```

Data Types

```
list object_list
```

ARGUMENTS

```
object_list  
    Lists ports, designs, or hierarchical cells. If this option is not specified,  
    the wire load model is removed from the current instance, or from the current  
    design if current instance is not set.
```

DESCRIPTION

Removes user-specified wire load model information on designs, ports, or hierarchical cells. The wire load model is used to calculate net capacitance, resistance, and area for designs, which are not placed and routed. You specify wire load models with the **set_wire_load_model** command, or by using automatic wire load selection.

To display wire load model settings for the current design or instance, use **report_wire_load**. To show wire load information for ports, use **report_port - wire_load**.

EXAMPLES

The following example removes wire load model settings from the current design and from all hierarchical cells in the design.

```
pt_shell> current_design TOP  
pt_shell> remove_wire_load_model  
pt_shell> remove_wire_load_model [get_cells -hier * -filter "is_hierarchical==true"]
```

The following example removes wire load model settings from port OUT2.

```
pt_shell> remove_wire_load_model [get_ports OUT2]
```

SEE ALSO

`report_port(2)`
`report_wire_load(2)`
`set_wire_load_model(2)`
`set_wire_load_selection_group(2)`
`report_design(2)`
`auto_wire_load_selection(3)`

remove_wire_load_selection_group

Removes wire load selection_group from current design.

SYNTAX

string **remove_wire_load_selection_group**

ARGUMENTS

None.

DESCRIPTION

Removes the wire load selection group setting from the current design. Both min and max selection group information is removed. The wire load selection group is specified with **set_wire_load_selection_group**. Refer to the **set_wire_load_selection_group** for details on this command.

To show the wire load selection group information for a design, use **report_design**.

EXAMPLES

This example removes the wire load selection group setting from the current design.

```
pt_shell> remove_wire_load_selection_group
```

SEE ALSO

report_design (2), **set_wire_load_selection_group** (2).

rename_cell

Change the name of a cell.

SYNTAX

```
int rename_cell cell new_name
list cell
string new_name
```

ARGUMENTS

cell

Specifies a cell to be renamed. Only one cell can be specified.

new_name

Specifies the new name for *cell*.

DESCRIPTION

The **rename_cell** command changes the name of a cell. Either a cell name that matches a single cell, or a collection of one cell, is passed in as the *cell* argument.

The **rename_cell** command fails if any of the following are true:

- PrimeTime cannot find *cell*.
- PrimeTime finds multiple cells that match *cell*.
- PrimeTime finds a leaf cell matching *new_name*.
- *new_name* is not at the same level of hierarchy as *cell*.

The **rename_cell** command is recorded for output with **write_changes**.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

rename_cell

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following examples, a cell is renamed in the current instance.

```
pt_shell> rename_cell cellA cellB
Information: Renamed cell 'cellA' to 'cellB' in design 'top'. (NED-008)
1
```

In the following examples, a cell is renamed at a level below the current instance. Currently, H1 and H1/H2 are instances of designs that have multiple instances. Therefore, they are uniquified as part of the editing operation.

```
pt_shell> rename_cell H1/H2/u1 H1/H2/cellC
Uniquifying 'H1/H2' (low) as 'low_0'.
Uniquifying 'H1' (inter) as 'inter_0'.
Information: Renamed cell 'u1 to 'cellC' in 'top/H1/H2'. (NED-008)
1
```

SEE ALSO

rename_design (2), **rename_net** (2), **write_changes** (2).

rename_design

Change the name of a design.

SYNTAX

```
int rename_design design new_name
list design
string new_name
```

ARGUMENTS

design
Specifies a design to be renamed. Only one design can be specified.

new_name
Specifies the new name for *design*.

DESCRIPTION

The **rename_design** command changes the name of a design. Either a design name which matches a single design, or a collection of one design, is passed in as the *design* argument. The command allows only simple design name changes for a single design. You cannot change the association of a design with its source file.

For the *design* to be renamed to *new_name*, there cannot be a design named *new_name* in the same file as *design* (see the Examples). The command may also fail if there are any instances of *design* in a linked design.

Renaming designs in PrimeTime should be done prior to any linking to avoid the case of instantiation conflicts.

Note that the **rename_design** command is not recorded for output with **write_changes**.

EXAMPLES

In the following example, design *top* is successfully renamed *top2*.

```
pt_shell> list_designs
Design Registry:
* top /home/designs/top.v:top
1
pt_shell> rename_design [get_designs top] top2
Renamed design 'top' to 'top2'
1
pt_shell> list_designs
Design Registry:
* top2 /home/designs/top.v:top2
1
```

In the following example, design *fbox* cannot be renamed *mbox* because there is already a design of that name in the same file. An attempt to rename it to *fbox2* fails because *fbox* is instantiated in a linked design.

```
pt_shell> list_designs
Design Registry:
    fbox          /home/designs/top.v:fbox
    mbox          /home/designs/top.v:mbox
    *L top        /home/designs/top.v:top
1
pt_shell> rename_design fbox mbox
Error: Could not rename design 'fbox' to 'mbox':
design 'mbox' already exists in file
    '/home/ajs/designs/top.v' (NED-020)
0
pt_shell> rename_design fbox fbox2
Error: Could not rename design 'fbox' to 'fbox2':
design 'fbox' is instantiated in design 'top' (NED-020)
0
```

SEE ALSO

list_designs (2), **link_design** (2), **write_changes** (2).

rename_net

Change the name of a net.

SYNTAX

```
int rename_net net new_name
list net
string new_name
```

ARGUMENTS

net

Specifies a net to be renamed. Only one net can be specified.

new_name

Specifies the new name for *net*.

DESCRIPTION

The **rename_net** command changes the name of a net. Either a net name that matches a single net, or a collection of one net, is passed in as the *net* argument.

The **rename_net** command fails if any of the following are true:

- PrimeTime cannot find *net*.
- PrimeTime finds multiple nets that match *net*.
- PrimeTime finds a net, port or hierarchical pin matching *new_name* in the same hierarchical block as *net*.
- *new_name* is not at the same level of hierarchy as *net*.

The **rename_net** command is recorded for output with **write_changes**.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and

relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following examples, a net is renamed in the current instance.

```
pt_shell> rename_net netA netB
Information: Renamed net netA' to 'netB' in design 'top'. (NED-009)
1
```

In the following examples, a net is renamed at a level below the current instance.

Currently, H1 and H1/H2 are instances of designs that have multiple instances. Therefore, they are uniquified as part of the editing operation.

```
pt_shell> rename_net H1/H2/w1 H1/H2/netC  
Uniquifying H1/H2 (low) as 'low_0'.  
Uniquifying 'H1' (inter) as 'inter_0'.  
Information: Renamed net 'w1' to 'netC' in 'top/H1/H2'. (NED-009)  
1
```

SEE ALSO

rename_design (2), **rename_cell** (2), **write_changes** (2).

report_activity_waveforms

Reports on activity analysis of VCD.

SYNTAX

```
int report_activity_waveforms
[-nosplit]
```

ARGUMENTS

-nosplit
Specifies that lines with overflow should not be split. This can be useful when the output is read by a script.

DESCRIPTION

After running the **write_activity_waveforms** command, you can use the **report_activity_waveforms** command to summarize the results and give the peak activity times for each block in the VCD file analyzed.

EXAMPLES

This example shows the form of the report.

```
pt_shell> write_activity_waveforms -output filename1 -vcd my_vcd.vcd -interval 10
pt_shell> report_activity_waveforms
```

```
*****
Report : Time-Based Switching Activity
Activity File: my_vcd.vcd
Version: B-2008.06-Dev-DEV
Date   : Wed Apr  9 17:54:06 2008
*****
```

Block Name le	# Signals	Peak Interval Start	Peak Interval End	Peak Togg Rate
TOP_level_of_my_design 0192	1383	0	2000	0.00
Second_level_of_my_design 0199	1335	0	2000	0.00

SEE ALSO

`write_activity_waveforms(2)`

report_alternative_lib_cells

Generates a report that contains data to aid in the selection of alternative library cells for a cell in the current design.

SYNTAX

string **report_alternative_lib_cells**

```
[-current_library]
[-libraries lib_spec]
[-delay_type delay_type]
[-all_pins]
[-weighted_design_cost]
[-total_design_cost]
[-significant_digits digits]
[-nosplit]
cell
```

Data Types

<i>cell</i>	string
<i>lib_spec</i>	list
<i>delay_type</i>	string
<i>digits</i>	int

ARGUMENTS

-current_library

If you specify this option, PrimeTime searches for library cells in the cell's current library only. You cannot specify this option with the *-libraries* option.

-libraries *lib_spec*

Specifies a list of libraries from which to select alternative library cells. The default is to select from all libraries in the link path. The *lib_spec* can be a list of library names, or collections of libraries loaded into PrimeTime; the latter can be obtained using the **get_libs** command. You cannot specify this option with the *-current_library* option.

-delay_type *delay_type*

Specifies the delay type to be reported for slack values. Allowed values are *max* (the default), *min*, *min_max*, *max_rise*, *max_fall*, *min_rise*, or *min_fall*. For the *min* and *max* values, the worst of rise and fall values is reported.

-all_pins

Indicates that the report is to include all pins. By default, the report includes the worst of the output pins.

-weighted_design_cost

Indicates that the report is to contain the weighted design cost (negative slack). The value is calculated as a weighted sum over the weighted cost

report_alternative_lib_cells

values calculated for all path groups. The cost values reported are the same as those reported by the **report_constraints** command for minimum delay (costs as a result of hold constraints) and maximum delay (costs as a result of setup constraints).

The minimum delay value is calculated as follows:

weighted min_delay/hold cost = summation over all path groups
(group weight * sum of all non-zero hold costs in group)

The maximum delay value is calculated as follows:

weighted max_delay/setup cost = summation over all path groups
(group weight * worst setup cost in group)

Maximum cost values are reported when *-delay_type max*, *max_rise*, or *max_fall* is specified, or when the default (*max*) is accepted. Minimum cost values are reported when *-delay_type min*, *min_rise*, or *min_fall* is specified. If *min_max* is specified, both minimum and maximum values are reported.

-total_design_cost

Indicates that the report is to contain the total design cost (negative slack). This option is similar to the *-weighted_design_cost* option, except that the total design cost can change in cases when the weighted maximum delay (setup) cost does not. Hence, this value can give some indication of the overall improvement in the design timing when the change does not improve the worst setup timing violation.

The total design cost is also computed as a weighted sum of the total costs computed for each path group. Only positive cost values are included in the calculation. The value is calculated as follows:

total design cost = summation over all path groups
(group weight * group total cost)

group total cost = summation over all group endpoints
having positive cost (cost value)

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported.

Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. Fixed-precision floating-point arithmetics is used for delay calculation; therefore, the actual precision might depend on the platform.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The *-nosplit* option prevents line-splitting and facilitates writing software to extract information from the report output.

cell

Specifies a cell for which a collection of alternative library cells is required.

DESCRIPTION

The **report_alternative_lib_cells** command generates a report that contains data to aid in the selection of alternative library cells for a cell in the current design. A report is produced that provides the slack values that would be obtained by replacing the library cell referenced by the cell with each alternative library cell that could be used to size the cell. A size cell operation is actually performed on the cell using each alternative library cell, followed by a timing analysis, and the slack values are obtained for each alternative. For each alternative library cell, the report lists the worst slack over all cell output pins.

You can refine the type of reported slack values using the *-delay_type* option. Use the *-all_pins* option to report the values for all cell pins (inputs, outputs, and inouts). Alternative library cells must satisfy the criteria used by the **size_cell** command to be considered valid alternatives.

If the *-current_library* option is specified, PrimeTime searches for library cells only in the cells' current libraries. If the *-libraries* option is specified, PrimeTime searches for library cells in the libraries contained only within the *lib_spec*.

Alternatively, if neither of the above options are specified, PrimeTime searches for the library cell in the following sources in this order:

- 1) The cell's current library.
- 2) The **link_path_per_instance** variable.
- 3) The **link_path** variable. If a library is found with a minimum library relationship (from the **set_min_library** command), the minimum library is automatically included.

To create a collection of library cells that can be used to "size" a specified cell, use the **get_alternative_lib_cells** command.

EXAMPLES

The following example illustrates generating a report of the alternatives library cells. The alternatives are taken from all libraries in the link path (the default). The default delay type, *max*, is used.

```
pt_shell> report_alternative_lib_cells AND2_0
```

```
*****
Report : alternative_lib_cells
        mux0
        -delay_type max
Design : swap0/0
Version: 2000.05-SI1
Date   : Fri Dec 17 14:37:22 1999
*****
```

Alternative	Slack
Library Cells	

AND2_2	-1.34(r)
AND2_1	-1.78(f)
AND2_0 *	-2.34(r)
AND2_3	-3.01(f)

*: The original library cell, not an alternative.

The following example generates a report that includes minimum and maximum slacks.

```
pt_shell> report_alternative_lib_cells AND2_0 -delay_type min_max
```

```
*****
Report : alternative_lib_cells
        mux0
        -delay_type min_max
Design : swap0/0
Version: 2000.05-SI1
Date   : Fri Dec 17 14:37:23 1999
*****
```

Alternative	Slack (min:max)
Library Cells	
AND2_1	1.34:8.00(r)
AND2_2	-1.01:-1.00(f)
AND2_0 *	2.00:-2.34(r)
AND2_3	3.34:-3.01(f)

*: The original library cell, not an alternative.

The following example generates a report that contains the weighted and total design costs. The alternative cells are to be chosen from all libraries in memory. The example illustrates that the weighted design cost is not affected by any of the alternatives; however, the total cost of the design can be improved by using cells AND2_2 or AND2_1 over the current cell AND2_0.

```
pt_shell> set libs_in_mem [get_libs *]
pt_shell> report_alternative_lib_cells AND2_0 -libraries $libs_in_mem
        -delay_type max -weighted_design_cost -total_design_cost
*****
Report : alternative_lib_cells
        mux0
        -delay_type max
Design : swap0/0
```

```
Version: 2000.05-SI1
Date   : Fri Dec 17 14:37:22 1999
*****
```

Alternative Library Cells	Slack	Weighted max_delay/setup Cost	Total Design Cost

AND2_2	1.01(r)	4.03	35.67
AND2_1	-1.34(f)	4.03	34.34
AND2_0 *	-2.34(r)	4.03	36.78
AND2_3	-3.01(f)	4.03	37.88

*: The original library cell, not an alternative.

SEE ALSO

```
get_alternative_lib_cells(2)
get_libs(2)
set_min_library(2)
size_cell(2)
link_path(3)
```

report_analysis_coverage

Generates a report about coverage of timing checks.

SYNTAX

```
string report_analysis_coverage  
[-status_details status_list]  
[-check_type check_type_list]  
[-exclude_untested untested_reason_list]  
[-sort_by sort_method]  
[-significant_digits digits]  
[-nosplit]  
[-pre_commands pre_command_string]  
[-post_commands post_command_string]
```

Data Types

<i>status_list</i>	list
<i>check_type_list</i>	list
<i>untested_reason_list</i>	list
<i>sort_method</i>	string
<i>digits</i>	int
<i>pre_command_string</i>	string
<i>post_command_string</i>	string

ARGUMENTS

-status_details *status_list*

Specifies a space-separated list of status names about which to show detail in the report. The allowed values are *untested*, *violated*, and *met*. By default, only summary information is shown. Use this option to see information about individual timing checks.

-check_type *check_type_list*

Specifies a list of check types to include in the report. The allowed values are *setup*, *hold*, *recovery*, *removal*, *nochange*, *min_period*, *min_pulse_width*, *clock_separation*, *clock_gating_setup*, *clock_gating_hold*, *max_skew*, *out_setup*, and *out_hold*.

Note: The check types *out_setup* and *out_hold* refer to the output setup and output hold constraints generated by the **set_output_delay** command.

-exclude_untested *untested_reason_list*

Specifies a space-separated list of reasons to exclude an untested check in the report. A check classified as untested is excluded from the report for any of the reasons specified as values to this argument. Hence, the coverage statistics are unaffected by these excluded constraints. The allowed values are as follows:

- *constant_disabled*: Paths to this check are disabled because of case analysis or a logic constant propagated through the design (for example, caused by a signal tied high or low).

- *mode_disabled*: A timing constraint is disabled because it requires a mode to be selected in mode analysis, and that mode is not selected.
- *user_disabled*: The timing check is explicitly disabled by the user.
- *no_paths*: The timing check had no paths found to it and as a result there were no arrival times.
- *false_paths*: All paths were false to a constrained pin.
- *no_endpoint_clock*: The timing check has no destination clock signal to latch the data.
- *no_startpoint_clock*: The timing check has no clock that launches the data at a startpoint latch.
- *no_constrained_clock*: There is no constrained clock for skew or clock separation checks.
- *no_ref_clock*: There is no reference clock for skew or clock separation checks.
- *no_clock*: A minimum pulse width or period width check has no clock.
- *unknown*: The reason is not listed above.

-sort_by *sort_method*

Specifies the method of sorting for the output of the detailed list. The allowed values are *slack*, *name*, and *check_type*. The default is *slack*, which sorts the output first by slack (untested is treated as negative infinity slack), then by name, then by type of check.

If you specify *name*, the output is sorted by pin name, then by slack, then by check type.

If you specify *check_type*, the output is sorted by type of check, then by slack, then by pin name.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported. The allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The *-nosplit* option prevents line-splitting and facilitates writing software to extract information from the report output.

-pre_commands *pre_command_string*

This option is available only if you invoke PrimeTime with the *-multi_scenario* option. This option allows you to specify a string of commands to be executed in the slave context before the execution of the merged reporting commands. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

`-post_commands post_command_string`

This option is available only if you invoke PrimeTime with the `-multi_scenario` option. This option allows you to specify a string of commands to be executed in the slave context after the execution of the Bmerged reporting commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

DESCRIPTION

Generates a report showing information about the coverage of timing checks in the current design, or current instance if it is defined. The default report is a summary of checks by type. For each type of check (for example, *setup*), the report shows the number and percentage of checks meeting and violating constraints, and those that are untested; the report does not show check types for which there are no checks. The report does not show unconstrained output ports; that is, those that have no output delay or *max_delay* or *min_delay*; however, the report does show constrained output ports.

You can see more information about the individual timing checks by using the `-status_details` option, which shows all checks with the corresponding status. Untested checks show information about the reason why they are untested, if the reason can be determined.

By default, the coverage statistics include constraints that have been disabled. Constraints could be disabled because of constant propagation (for logic constants in the design and case analysis), modes that are not enabled, or explicit disabling of timing arcs by the user. Such disabled constraints appear as untested in the coverage statistics. If you want to exclude some of these untested constraints from the coverage statistics, use the `-exclude_untested` option.

EXAMPLES

The following example shows the summary report.

```
pt_shell> report_analysis_coverage
*****
Report : analysis_coverage
Design : counter
Version: 1999.10
Date   : Mon Nov  1 13:40:04 1999
*****
```

Type of Check	Total	Met	Violated	Untested
setup	5	0 (0%)	3 (60%)	2 (40%)
hold	5	3 (60%)	0 (0%)	2 (40%)
All Checks	10	3 (30%)	3 (30%)	4 (40%)

The following example shows the detailed report of untested setup checks.

```
pt_shell> report_analysis_coverage -status_details {untested} -check_type {setup}
*****
Report : analysis_coverage
        -status_details {untested }
        -sort_by slack
        -check_type {setup }
Design : counter
Version: 1999.10
Date   : Mon Nov  1 13:40:04 1999
*****
```

Type of Check	Total	Met	Violated	Untested
setup	5	0 (0%)	3 (60%)	2 (40%)
All Checks	5	0 (0%)	3 (60%)	2 (40%)

Constrained Pin	Related Pin	Check Type	Slack	Reason
ffd/CR	CP	setup	untested	no_clock
ffd/D	CP	setup	untested	no_clock

The following example shows generation of a report that includes details of untested and violated checks, excludes timing checks if they are untested because of constant propagation, sorts by slack, and shows the *setup* and *out_setup* checks.

```
pt_shell> report_analysis_coverage -check_type {out_setup setup} \
-exclude_untested {constant_disabled} -status_details {violated untested}
*****
Report : analysis_coverage
        -status_details {untested violated }
        -exclude_untested {constant_disabled}
        -sort_by slack
        -check_type {setup out_setup}
Design : seq_case
Version: 1999.10
Date   : Mon Nov  1 13:40:04 1999
*****
```

Type of Check	Total	Met	Violated	Untested
setup	6	5 (83%)	1 (17%)	0 (0%)
out_setup	5	4 (80%)	0 (0%)	1 (20%)
All Checks	11	9 (82%)	1 (9%)	1 (9%)

Constrained Pin	Related Pin	Check Type	Slack	Reason

Q2 ff4/TI	CP	out_setup setup	untested -2.30	no_paths

SEE ALSO

```
check_timing(2)
current_instance(2)
report_constraint(2)
report_timing(2)
report_default_significant_digits(3)
```

report_annotated_check

Reports back-annotated timing checks.

SYNTAX

```
string report_annotated_check  
[-setup]  
[-hold]  
[-recovery]  
[-removal]  
[-nochange]  
[-width]  
[-period]  
[-max_skew]  
[-clock_separation]  
[-max_line num]  
[-list_annotated]  
[-list_not_annotated]  
[-constant_arcs]
```

Data Types

num int

ARGUMENTS

```
-setup            Reports setup timing checks.  
  
-hold            Reports hold timing checks.  
  
-recovery        Reports recovery timing checks.  
  
-removal        Reports removal timing checks.  
  
-nochange        Reports nochange timing checks.  
  
-width           Reports minimum pulse width timing checks.  
  
-period          Reports minimum period timing checks.  
  
-max_skew        Reports maximum skew timing checks.  
  
-clock_separation   Reports clock separation timing checks.
```

report_annotated_check

-max_line num
Provides maximum number of line for the *-list_** options.

-list_annotated
Lists timing arcs that are back-annotated. Use this option if no annotated checks are expected to identify the specified annotated checks.

-list_not_annotated
Lists timing arcs that are not back-annotated. Use this option if some annotated checks are missing to identify the missing annotated checks.

-constant_arcs
Keeps a separate count for the arcs that are disabled due to logic constants (but not case_analysis). With this option, the *-list_annotated* or *-list_not_annotated* option does not list the arc disabled due to logic constants. The total count at the end of the table includes the constant_arcs count.

DESCRIPTION

Provides a summary report of how many cell timing checks are annotated in the current design. The purpose of this command is to check if all timing checks of the design are annotated after reading an SDF file. You can use the *-list_not_annotated* option to identify which check timing arcs are not annotated.

EXAMPLES

The following are examples of annotated reports.

```
pt_shell> report_annotated_check
```

```
*****
report: annotated_check
*****
```

	Total	Annotated	NOT Annotated
cell setup arcs	2	0	2
cell hold arcs	2	0	2
cell recovery arcs	2	0	2
cell removal arcs	2	0	2
cell nochange arcs	2	0	2
cell min pulse width arcs	2	0	2
cell min period arcs	2	0	2
cell max skew arcs	2	0	2
cell clock separation arcs	2	0	2
	18	0	18

```
pt_shell> report_annotated_check -constant_arcs
```

```
*****
report: annotated_check
*****
```

	Total	Annotated	NOT Annotated
-----+-----+-----+-----			
cell setup arcs	2	0	1
constant arcs		0	1
cell hold arcs	2	0	1
constant arcs		0	1
cell recovery arcs	2	0	2
constant arcs		0	0
cell removal arcs	2	0	2
constant arcs		0	0
cell nochange arcs	2	0	2
constant arcs		0	0
cell min pulse width arcs	2	0	2
constant arcs		0	0
cell min period arcs	2	0	2
constant arcs		0	0
cell max skew arcs	2	0	2
constant arcs		0	0
cell clock separation arcs	2	0	2
constant arcs		0	0
-----+-----+-----+-----			
	18	0	18

SEE ALSO

```
read_sdf(2)
set_annotated_check(2)
remove_annotated_check(2)
reset_design(2)
```

report_annotated_delay

Reports back-annotated delays.

SYNTAX

```
string report_annotated_delay  
[-cell]  
[-net]  
[-from_in_ports]  
[-to_out_ports]  
[-max_line num]  
[-list_annotated]  
[-list_not_annotated]  
[-constant_arcs]  
[-crosstalk_delta]
```

Data Types

num int

ARGUMENTS

- cell**
Reports all data annotated on cells. By default, both cell and net annotated data are reported.
- net**
Reports all data annotated on nets. By default, both cell and net annotated data are reported.
- from_in_ports**
Includes the nets that start from input ports. By default, nets that start at input ports are not included in the list of nets reported for annotated delays. The reason is that SDF format specifies the nets starting at input ports of the design are not part of the SDF file because the delay of this net is dependent on the external environment.
- to_out_ports**
Includes the nets that end at output ports. By default, nets that end at output ports are not included in the list of nets reported for annotated delays. The reason is that SDF format specifies the nets ending at output ports of the design are not part of the SDF file because the delay of this net is dependent on the external environment.
- max_line *num***
Shows maximum number of lines reported for the *-list_** options.
- list_annotated**
Lists timing arcs that are back-annotated. Use this option to identify the specified annotated delays if no annotated delays are expected. This will list both connected and unconnected cells.

-list_not_annotated
 Lists timing arcs that are not back-annotated. Use this option to identify the missing annotated delays if some annotated delays are missing. This will list both connected and unconnected cells.

-constant_arcs
 Keeps a separate count for the arcs that are disabled due to logic constants (but not case_analysis). With this option, the *-list_annotated* or *-list_not_annotated* option does not list the arc disabled due to logic constants. The total count at the end of the table includes the *constant_arcs* count.

-crosstalk_delta
 Includes delta crosstalk delays that have been annotated through SDF in the report. A separate section is used to display the crosstalk specific information.

DESCRIPTION

Provides a summary report of how many cell and net delays are annotated in the *current design*. The purpose of this command is to check if all delays of the design are annotated after reading an SDF file. By default, net delay starting from input ports or ending at output ports are considered separately. The *-list_not_annotated* option is used to identify which delay arcs are not annotated. The *-crosstalk* option displays information about arcs that have been annotated with crosstalk values through SDF.

EXAMPLES

The following is an example of a default report showing annotated delays:

```
pt_shell> report_annotated_delay
```

```
*****
report: annotated_delay
*****
```

	Total	Annotated	NOT Annotated
cell arcs	19	19	0
cell arcs (unconnected)	1	1	0
internal net arcs	3	3	0
net arcs from primary inputs	11	11	0
net arcs to primary outputs	4	4	0
	38	38	0

The following is an example of a delay annotation report that includes a separate

```
report_annotated_delay
```


count for constant arcs.

```
pt_shell> report_annotated_delay -constant_arcs
```

```
*****  
report: annotated_delay  
*****
```

	Total	Annotated	NOT Annotated
-----+-----+-----+-----			
cell arcs	19	17	0
cell arcs (unconnected)	1	1	0
constant arcs		2	0
internal net arcs	3	3	0
constant arcs		0	0
net arcs from primary inputs	11	11	0
constant arcs		0	0
net arcs to primary outputs	4	4	0
constant arcs		0	0
-----+-----+-----+-----			
	38	38	0

SEE ALSO

```
read_sdf(2)  
remove_annotated_delay(2)  
reset_design(2)  
set_annotated_delay(2)
```

report_annotated_parasitics

Reports net parasitics back-annotated on the current design.

SYNTAX

```
string report_annotated_parasitics  
[-check]  
[-internal_nets]  
[-boundary_nets]  
[-driverless_nets]  
[-loadless_nets]  
[-pin_to_pin_nets]  
[-max_nets num]  
[-list_annotated]  
[-list_not_annotated]  
[-constant_arcs]  
[-variation]  
[-no_check]  
[net_list]
```

Data Types

num int

ARGUMENTS

-check

Indicates that the design is to be checked to verify that all annotated RC networks are complete. This option causes the **report_annotated_parasitics** command to check that all fanouts of each net connect through the RC network to each driver of the net. An error message reports nets with incomplete RC networks. This option is the default for PrimeTime version C-2009.06 and later.

-no_check

Indicates that the design is NOT to be checked to verify that all annotated RC networks are complete.

-internal_nets

Indicates that only parasitics annotated on internal nets (nets connected only to cell pins) are reported. By default, both boundary (port) net and internal nets are reported.

-boundary_nets

Indicates that only parasitics annotated on boundary (port) nets are reported. A connection to any port qualifies a net as a boundary net. By default, both boundary (port) net and internal nets are reported.

-pin_to_pin_nets

Indicates that only parasitics annotated on nets that have at least one global driver and at least one global load pin. By default, all nets are reported.

report_annotated_parasitics

-driverless_nets
 Indicates that only parasitics annotated on driverless nets (nets that do not have a global driver) are reported. By default, all nets are reported.

-loadless_nets
 Indicates that only parasitics annotated on loadless nets (nets that do not have a global load) are reported. Note that if there are nets that neither have a global driver nor a global load, those nets are counted as driverless nets and not as loadless nets for the report. By default, all nets are reported.

-variation
 Indicates that information about variation-aware parasitics should be printed. If this option is used without specifying the *-list_annotated* option, information about the global parasitic variation sources is printed. If this option is used along with *-list_annotated* option, the sensitivity factors for all RC elements are printed.

-max_nets num
 Specifies a maximum number of nets to report for the *-list_annotated* and *-list_not_annotated* options.

-list_annotated
 Indicates that back-annotated nets are to be listed. This option is exclusive to the *-list_not_annotated* option.

-list_not_annotated
 Indicates that nets that are not back-annotated are to be listed. This option is exclusive to the *-list_annotated* option.

-constant_arcs
 Keeps a separate count for nets connected to pins with arcs that are disabled due to logic constants (but not case analysis). With this option, the *-list_annotated* or *-list_not_annotated* option does not list the constant nets. The total count at the end of the table includes the count of constant nets, and a count of constant nets suppressed (due to *-max_nets*) is shown separately from the count of ordinary nets suppressed.

net_list
 Limits the report to the specified nets.

DESCRIPTION

Provides a report of nets annotated with parasitics in the current design. This command can also check the consistency of parasitics after reading ascii or binary parasitics. The report summarizes how many nets are annotated with reduced parasitics (pi models) or detailed parasitics (RC networks).

Parasitics are associate with global nets. This report counts nets without considering hierarchical crossings - only one segment per global net is reported.

The report clearly shows how many nets are back-annotated with lumped, pi and detailed RC networks. The nets that are not annotated are shown in a separate column. In addition, nets that do not have a global driver and nets that do not have

any global load pins are shown separately. If a net falls into both these categories, that is if a net does not have a driver and does not have a load, it is shown in driver-less nets row.

Details on which nets are annotated or not annotated can be displayed using either the `-list_annotated` or `-list_not_annotated` option. Listing annotated nets show a detailed description of RC networks. Listing unannotated nets show only a list of net names. By default, these reports only show 10 nets. To increase that limit, use the `-max_nets` option.

You can also report parasitics for a specific set of nets by using the `net_list` option. The `-list_annotated` and `-list_not_annotated` options restrict their scope to the nets in `net_list`.

When the **si_enable_analysis** variable is set, a column for coupled networks is added to the summary report that provides the number nets that have coupling capacitors.

If the `-check` option is used, and the command reports nets with incomplete parasitics, you can complete partially annotated nets using the **complete_net_parasitics** command.

EXAMPLES

The following is an example of an annotated parasitics report.

```
pt_shell> report_annotated_parasitics -check
```

```
*****
report: annotated_parasitics
      -check
      -internal_nets
      -boundary_nets
*****
```

Net Type	Total	Lumped	RC pi	RC network	Not Annotated
Internal nets					
- Pin to pin nets	23645	0	0	23644	1
- Driverless nets	3	0	0	0	3
- Loadless nets	1	0	0	0	1
Boundary/port nets					
- Pin to pin nets	34	0	0	34	0
- Driverless nets	0	0	0	0	0
- Loadless nets	0	0	0	0	0
	23683	0	0	23678	5

The following is an example of an annotated parasitics report for a coupled network.

```
report_annotated_parasitics
```

```
pt_shell> report_annotated_parasitics -check
```

```
*****
report: annotated_parasitics
      -check
      -internal_nets
      -boundary_nets
*****
```

Net Type	Total	Lumped	RC pi	RC network	Coupled network	Not Annotated
Internal nets						
- Pin to pin nets	23645	0	0	5453	18191	1
- Driverless nets	3	0	0	0	0	3
- Loadless nets	1	0	0	0	0	1
Boundary/port nets						
- Pin to pin nets	34	0	0	12	22	0
- Driverless nets	0	0	0	0	0	0
- Loadless nets	0	0	0	0	0	0
	23683	0	0	5465	18213	5

The following is an example of an annotated parasitics report, with PrimeTime SI analysis off, showing constant nets.

```
pt_shell> report_annotated_parasitics -check -constant_arcs
```

```
*****
report: annotated_parasitics
      -check
      -internal_nets
      -boundary_nets
      -constant_arcs
*****
```

Net Type	Total	Lumped	RC pi	RC network	Coupled network	Not Annotated
Internal nets						
- Pin to pin nets	23640	0	0	5453	18186	1
- Driverless nets	3	0	0	0	0	3
- Loadless nets	1	0	0	0	0	1
- Constant nets	5	0	0	0	5	0
Boundary/port nets						
- Pin to pin nets	6	0	0	12	22	0

- Driverless nets		0		0		0		0		0		0	
- Loadless nets		0		0		0		0		0		0	
- Constant nets		0		0		0		0		0		0	
-----+													
		23683		0		0		5465		18213		5	

SEE ALSO

[complete_net_parasitics\(2\)](#)
[read_parasitics\(2\)](#)
[remove_annotated_parasitics\(2\)](#)
[reset_design\(2\)](#)
[si_enable_analysis\(3\)](#)

report_annotated_power

Reports annotated power.

SYNTAX

```
int report_annotated_power
[-list_annotated]
```

ARGUMENTS

```
-list_annotated
    Indicates to list powers that are annotated.
```

DESCRIPTION

This command provides a summary report of how many powers are annotated in the current design and, if the *-list_annotated* option is specified, a list of cells with power values annotated on them.

EXAMPLES

The following example shows how power is annotated, removed, and reported.

```
pt_shell> set_annotated_power -int 1 -leak 0.01 u0/*
1
pt_shell> report_annotated_power -list_annotated
```

```
*****
Report : annotated_power
        -list_annotated
*****
```

Annotated cell powers:

-
- 1. u0/u0 (internal: 1 leakage: 0.01)
- 2. u0/u1 (internal: 1 leakage: 0.01)

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	1	2
	5	2	3

```
1
pt_shell> remove_annotated_power u0/u0
1
```

```
pt_shell> report_annotated_power
```

```
*****  
Report : annotated_power  
*****
```

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	0	3
	5	1	4

1

SEE ALSO

```
set_annotated_power(2)  
remove_annotated_power(2)
```


report_aocvm

Reports information about advanced on-chip variation (OCV) derate tables and coefficients. Also displays details of path-based and graph-based advanced OCV calculation.

SYNTAX

```
int report_aocvm  
[-early] [-late]  
[-rise] [-fall]  
[-voltage value]  
[-cell_delay] [-net_delay]  
[-list_annotated] [-list_not_annotated]  
[-coefficient]  
[-nosplit]  
[object_list]
```

Data Types

object_list list

ARGUMENTS

- early
Indicates that only early advanced OCV derate tables are shown on the objects specified in the *object_list*.
- late
Indicates that only late advanced OCV derate tables are shown on the objects specified in the *object_list*.
- rise
Indicates that only rise advanced OCV derate tables are shown on the objects specified in the *object_list*.
- fall
Indicates that only fall advanced OCV derate tables are shown on the objects specified in the *object_list*.
- voltage value
Indicates that only voltage value-specific advanced OCV derate tables are shown on the objects specified in the *object_list*.
- cell_delay
Indicates that only cell delay advanced OCV derate tables are shown on the objects specified in the *object_list*.
- net_delay
Indicates that only net delay advanced OCV derate tables are shown on the objects specified in the *object_list*.

-list_annotated
Indicates that leaf cells and global nets that are annotated with advanced OCV derate tables are listed.

-list_not_annotated
Indicates that leaf cells and global nets that are not annotated with advanced OCV derate tables are listed.

-coefficient
Indicates that random advanced OCV coefficients are shown. You can specify a collection of *lib_cell*, *lib_timing_arc*, and *cell* objects in the *object_list* to display coefficients annotated only on those objects.

-nosplit
Do not split lines when columns overflow.

object_list
Specifies either *timing_path* objects for which path-based advanced OCV metrics are to be reported; or *timing_arc* objects for which graph-based advanced OCV metrics are to be reported; or design objects on which advanced OCV derate tables and advanced OCV derate coefficients have been annotated.

DESCRIPTION

Displays the user-specified advanced OCV information. The type of information displayed by PrimeTime depends on the type of advanced OCV information annotated.

If the design has been annotated with advanced OCV derate tables using the **read_aocvm** command, the **report_aocvm** command outputs a summary showing the numbers of leaf cells and global nets annotated with advanced OCV derate tables. Lists of fully annotated, partially annotated, and not annotated leaf cells and global nets can be displayed using the *-list_annotated* and *-list_not_annotated* options. You can specify a collection of the design, library arcs, hierarchical cells, and nets in the *object_list* to display advanced OCV derate tables annotated only on those objects.

If the design has been annotated with advanced OCV derate coefficients using the **set_aocvm_coefficient** command, the **report_aocvm -coefficient** command displays these coefficients. You can specify a collection of library cells, library timing arcs, and cells in the *object_list* to display coefficients annotated only on those objects.

If a timing path is specified in the *object_list*, path-based metrics (distance, launch depth, and capture depth) for that path are displayed. Note that the depths shown in the report have been scaled by random advanced OCV coefficients, if they exist. Timing paths can be obtained using the **get_timing_paths** command.

If a timing arc is specified in the *object_list*, graph-based metrics for that arc are displayed. Note that the depths shown in the report have been scaled by random advanced OCV coefficients, if they exist. Timing arcs can be obtained using the **get_timing_arcs** command. Graph-based timing arc metrics are only displayed when graph-based advanced OCV analysis has been enabled by setting the **timing_aocvm_enable_analysis** variable to *true*.

EXAMPLES

In the following example, the design has been annotated with advanced OCV derate factors using the **read_aocvm** command.

```
pt_shell> report_aocvm
```

```
*****
Report : aocvm
Design : my_design
*****
```

	Total	Fully annotated	Partially annotated	Not annotated
Leaf cells	6	0	4	2
Nets	7	0	0	7
	13	0	4	9

1

The following example displays path-based metrics for the timing path specified in the *object_list*, for a design that has been annotated with advanced OCV derates.

```
pt_shell> report_aocvm [get_timing_paths -path_type full_clock_expanded]
```

```
*****
Report : aocvm
object_list
Design : my_design
*****
```

```
Startpoint: ffL (rising edge-triggered flip-flop clocked by clk2)
Endpoint: ffC (rising edge-triggered flip-flop clocked by clk2)
Path Group: clk2
```

Path metrics	Cell	Net
Distance	575.85	666.41
Launch depth	2.69	3.07
Capture depth	1.00	1.09

1

The following example displays graph-based metrics and advanced OCV derate factors

for the timing arc specified in the *object_list*. This report is only displayed when graph-based advanced OCV is enabled. The arc depth and distance displayed is the most pessimistic depth and distance for all possible paths through this arc.

```
pt_shell> report_aocvm [get_timing_arcs -from u2/A -to u2/Z]
```

```
*****
Report : aocvm
object_list
Design : my_design
*****
```

```
From pin: u2/A
To pin:   u2/Z
Arc type: cell (data network)
```

AOCVM arc metrics	Launch
Distance	702.14
Depth	3.00

AOCVM arc derates	Launch
Early rise	0.8705
Early fall	0.8508
Late rise	1.1357
Late fall	1.1566

1

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
remove_aocvm(2)
report_timing(2)
set_aocvm_coefficient(2)
timing_aocvm_enable_analysis(3)
```

report_attribute

Reports the attributes on one or more objects.

SYNTAX

```
string report_attribute  
[-class class_name]  
[-nosplit]  
[-application]  
object_spec
```

Data Types

<i>class_name</i>	string
<i>object_spec</i>	list

ARGUMENTS

-class *class_name*
If *object_spec* is a name, this is its class. Allowable values are *design*, *port*, *cell*, *pin*, *net*, *lib*, *lib_cell*, or *lib_pin*.

-nosplit
Does not split lines if column overflows.

-application
Lists application attributes as well as user-defined attributes.

object_spec
List of objects to report. Each element in the list is either a collection or a pattern that combines with the *class_name* value to find the objects.

DESCRIPTION

Generates a report of attributes on the specified objects. By default, only user-defined attributes are displayed. Using the *-application* option adds application attributes to the report. For application attributes that are of the type *collection*, the name of the first object in the collection is displayed.

EXAMPLES

This example defines an attribute 'X' for cells, sets the value on all cells in this level of the hierarchy, and reports the result.

```
pt_shell> define_user_attribute -type int -class cell X  
pt_shell> set_user_attribute [get_cells *] X 30  
Set attribute 'X' on 'i1'
```

```
Set attribute 'X' on 'i2'
pt_shell> report_attribute [get_cells *]
```

```
*****
Report : Attribute
Design : my_des
*****
```

Design	Object	Type	Attribute Name	Value
my_des	i1	int	X	30
my_des	i2	int	X	30

This example shows how application attributes can display.

```
pt_shell> report_attribute -application [get_designs my_des]
```

```
*****
Report : Attribute
Design : my_des
*****
```

Design	Object	Type	Attribute Name	Value
my_des	my_des	string	full_name	my_des
my_des	my_des	string	object_class	design
my_des	my_des	float	area	56.000000
my_des	my_des	string	tree_type_max	balanced_case
my_des	my_des	string	tree_type_min	balanced_case
my_des	my_des	float	wire_load_min_block_size	0.000000
my_des	my_des	string	wire_load_mode	top

SEE ALSO

```
collections(2)
define_user_attribute(2)
get_attribute(2)
list_attributes(2)
remove_user_attribute(2)
set_user_attribute(2)
```

report_bottleneck

Reports timing bottleneck information.

SYNTAX

```
string report_bottleneck
[-cost_type cost_type]
[-delay_type delay_type]
[-slack_lesser_than slack_limit]
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
[-through through_list]
[-rise_through through_list]
[-fall_through through_list]
[-max_cells cell_count]
[-max_paths path_count]
[-nworst_paths paths_per_endpoint]
[-group group_name]
[-significant_digits digits]
[-nosplit]
```

Data Types

<i>cost_type</i>	string
<i>delay_type</i>	string
<i>cell_count</i>	int
<i>from_list</i>	list
<i>rise_from_list</i>	list
<i>fall_from_list</i>	list
<i>to_list</i>	list
<i>rise_to_list</i>	list
<i>fall_to_list</i>	list
<i>through_list</i>	list
<i>rise_through_list</i>	list
<i>fall_through_list</i>	list
<i>paths_per_endpoint</i>	int
<i>path_count</i>	int
<i>slack_limit</i>	float
<i>group_name</i>	list
<i>digits</i>	int

ARGUMENTS

- cost_type *cost_type***
Specifies a cost type to use for computing the bottleneck cost. Valid values are:
- **path_count** (the default) - Uses the number of violating paths through the

cell.

- `path_cost` - Uses the total cost of violating paths through the cell.
- `fanout_endpoint_cost` - Uses the total cost of violating endpoints in the fanout of the cell.

`-delay_type delay_type`

Specifies whether to compute the bottleneck cost for setup (max) or hold (min) analysis. Valid values are max (the default) and min.

`-slack_less_than slack_limit`

Indicates that only those paths with a slack less than `slack_limit` are to be considered to find the bottleneck.

`-from from_list`

Specifies that only paths from the named pins, ports, nets, or startpoints clocked by named clocks are to be considered to find the bottleneck.

`-rise_from rise_from_list`

Same as the `-from` option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

`-fall_from fall_from_list`

Same as the `-from` option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

`-to to_list`

Specifies that only paths to the named pins, ports, nets, or endpoints clocked by named clocks are to be considered.

`-rise_to rise_to_list`

Same as the `-to` option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path.

`-fall_to fall_to_list`

Same as the `-to` option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

`-through through_list`

Indicates that only paths through the named pins, ports, nets or cells are to be considered to find the bottleneck.

You can use multiple `-through`, `-rise_through`, and `fall_through` options in a single command to specify paths that traverse multiple points in the design.

The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one *-through*, *-rise_through*, or *-fall_through* option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

-rise_through *rise_through_list*

Same as the *-through* option, except that the path must rise through the objects specified.

-fall_through *fall_through_list*

Same as the *-through* option, except that the path must fall through the objects specified.

-max_cells *cell_count*

Specifies the number of cells to report. The default is 20.

-max_paths *path_count*

Specifies the number of paths to be considered per path group. Allowed values are 1 to 2000000; the default is to use the *paths_per_endpoint* setting.

-nworst_paths *paths_per_endpoint*

Specifies the number of paths to be considered per endpoint. Allowed values are 1 to 2000000; the default is 100.

-group *group_name*

Indicates that only paths in *group_name* are to be considered.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in the fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The *-nosplit* option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

Produces a report showing information about timing bottlenecks in the current

design. A bottleneck is a common point in the design that contributes to multiple violations.

To report bottlenecks throughout the current design, arrival totals, and slacks must be available on all pins, not just at endpoints. To achieve this, the **timing_save_pin_arrival_and_slack** variable should be set to *true* when this command is issued. If the **timing_save_pin_arrival_and_slack** variable has not been set to *true*, the command automatically sets it to *true* and updates the design timing before the command executes.

If you intend to use this command it is recommended that you set the **timing_save_pin_arrival_and_slack** variable to *true* before the first timing update, thus preventing the cost of an additional timing update.

EXAMPLES

The following example reports the 20 worst bottleneck cells in the design based on the number of paths through the cell with slack less than 0.0.

```
pt_shell> report_bottleneck
```

```
*****
Report : bottleneck
        -max_cells 20
        -nworst_paths 100
Design : Example
Version: 1998.01
*****
```

Bottleneck Cost = Number of violating paths through cell

Cell	Reference	Bottleneck Cost

ipxr/ir1/i2/i0/u3	DFF1A	39656.00
ipxr/ir1/U14	INV2	39654.00
ipxr/ir1/U16	INV8	39654.00
ipxr/ir1/i2/i0/u4	DFF1A	31806.00
ipxr/ir1/U15	BUF8B	31804.00
ipxr/U1712	MUX2I	23999.00
ipxr/U558	INV4	23999.00
ipxr/U1370	NAN4	22485.00
ipxr/x01	INV2	22485.00
dmac/BufEn	BUF3	22485.00
dmac/U574	INV	22485.00
ipxr/U1335	NAN6CH	21844.00
ipxr/U564	MUX2I	20074.00
ipxr/U1694	MUX2A	19936.00
ipxr/U1559	BUF2C	14785.00
ipxr/U1484	MUX2I	14252.00
ipxr/U1486	MUX2I	12468.00
SccMOD/U929	OR3	12135.00
ipxr/U1493	MUX2I	11248.00
dmac/BiuSMOD/U879	NAN2	10949.00

SEE ALSO

`report_cell(2)`
`report_timing(2)`
`report_default_significant_digits(3)`

report_bus

Reports the bused ports or nets in the current instance or current design.

SYNTAX

```
string report_bus  
[-nosplit]
```

ARGUMENTS

-nosplit
Does not split lines if column overflows.

DESCRIPTION

Displays information about buses (pins or nets) in the current instance or current design. If the current instance is set, the report is generated for the design of that instance; otherwise, the report is generated for the current design.

EXAMPLES

The following command reports the buses in the design.

```
pt_shell> report_bus
```

```
*****  
Report : bus  
Design : new_design  
*****
```

Bussed Port	Dir	From	To	Width
acnt	out	15	0	16
baddreg	out	15	0	16
bwordreg	out	15	0	16
caddreg	out	15	0	16

SEE ALSO

report_port(2)

S.

report_case_analysis

Reports case analysis entries on ports and pins.

SYNTAX

```
string report_case_analysis  
[-all]  
[-nosplit]
```

ARGUMENTS

-all

Reports the pins upon which you have set case analysis values and reports the built-in constant pins of the design that are considered for startpoints of logic constant propagation. Logic constant propagation is performed by default from the constant pins of the design, unless the **disable_case_analysis** variable is set to *true*.

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line starting in the correct column.

DESCRIPTION

Reports case analysis entries on ports and pins that are specified with the **set_case_analysis** command. The report lists all pins and ports on which case analysis is specified. The list does not report where the logic constant values are propagated.

EXAMPLES

The following example specifies that pins *U1/U2/A* and *U1/U3/CI* are set to a constant logic 1.

```
pt_shell> set_case_analysis 1 {U1/U2/A U1/U3/CI}  
pt_shell> report_case_analysis
```

```
*****  
Report : case_analysis  
Design : middle  
Version: 1997.01-development  
Date   : Mon Jul 22 17:04:17 1996  
*****
```

Pin name	User case analysis value
U1/U2/A	1
U1/U3/CI	1

SEE ALSO

`remove_case_analysis(2)`
`set_case_analysis(2)`

m.

report_cell

Reports cell information.

SYNTAX

```
string report_cell  
[-connections [-verbose]]  
[-significant_digits digits]  
[-nosplit]  
[cell_names]
```

Data Types

<i>digits</i>	integer
<i>cell_names</i>	list

ARGUMENTS

-connections
Displays the pins and the nets to which they connect.

-verbose
Displays verbose connection information. For each input pin, displays the net and the driver pins on that net. For each output pin, displays the net and the load pins on that net. Use this option in conjunction with the **-connections** option.

-nosplit
Does not split lines if column overflows.

-significant_digits *digits*
Specifies the number of digits to the right of the decimal point that are to be reported. Allowed values are 0-13. The default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. Fixed-precision floating-point arithmetics is used for delay calculation; therefore, the actual precision might depend on the platform. For example, on SVR4 UNIX see FLT_DIG in limits.h. The upper bound on a meaningful value of the argument to **-significant_digits** is then $\text{FLT\_DIG} - \text{ceil}(\log_{10}(\text{fabs}(\text{any\_reported\_value})))$.

cell_names
Lists cells to report. If you do not specify this option, all cells in the current instance are listed.

DESCRIPTION

Displays information and statistics about cells in the current instance or current design. If you set for the current instance, the report is generated for the design of that instance. Otherwise, the report is generated for the current design.

Some information, such as whether the cell is in an ideal network or not, is

displayed after a timing update (as a result of manually executing the **update_timing** command function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to obtain information and statistics about cells without incurring the cost of a complete timing update.

EXAMPLES

The following example displays a report of the cells in the current design.

```
pt_shell> report_cell
```

```
*****
Report : cell
Design : middle
Version: Y-2006.06
Date   : Wed Mar  8 03:18:34 2006
*****
```

Attributes:

```
  b - black-box (unknown)
  h - hierarchical
  n - noncombinational
  u - contains unmapped logic
  A - abstracted timing model
  E - extracted timing model
  S - Stamp timing model
  Q - Quick timing model (QTM)
  I - ideal network
```

Cell	Reference	Library	Area	Attributes
i1	inter		6.00	h
low_i	low		2.00	h
n0_i	ND2	my_lib	1.00	
o_reg2	FD2	my_lib	9.00	n
Total 4 cells			18.00	

1

The following example gives a verbose report of the cell connections.

```
pt_shell> report_cell -verbose -connections
```

```
*****
Report : cell
        -connections
        -verbose
Design : middle
```

Version: Y-2006.06

Date : Wed Mar 8 03:18:44 2006

Connections for cell 'i1':

Reference: inter
Hierarchical: TRUE
Area: 6

Input Pins	Net	Net Driver Pins	Driver Pin Type
a	out_1	o_reg1/Q	Output Pin (FD2)
b	out_2	o_reg2/Q	Output Pin (FD2)
c	out_3	o_reg3/Q	Output Pin (FD2)
d	out_4	o_reg4/Q	Output Pin (FD2)

Output Pins	Net	Net Load Pins	Load Pin Type
z1	i1t1	low_i/n1/A	Input Pin (NR4)
z2	i1t2	low_i/n1/B	Input Pin (NR4)
z3	i1t3	low_i/n1/C	Input Pin (NR4)

Connections for cell 'low_i':

Reference: low
Hierarchical: TRUE
Area: 2

Input Pins	Net	Net Driver Pins	Driver Pin Type
a	i1t1	i1/low/n1/Z	Output Pin (NR4)
b	i1t2	i1/low2/n1/Z	Output Pin (NR4)
c	i1t3	i1/low3/n1/Z	Output Pin (NR4)
d	in2	in2	Input Port

Output Pins	Net	Net Load Pins	Load Pin Type
z	low	o_reg2/D	Input Pin (FD2)

Connections for cell 'n0_i':

Reference: ND2
Library: my_lib
Area: 1

Input Pins	Net	Net Driver Pins	Driver Pin Type
A	p_in	p_in	Input Port
B	p_in	p_in	Input Port

Output Pins	Net	Net Load Pins	Load Pin Type
Z	n0	n1_i/B	Input Pin (ND2)

Connections for cell 'o_reg2':

Reference: FD2

Library: my_lib
Area: 9

Input Pins	Net	Net Driver Pins	Driver Pin Type
-----	-----	-----	-----
D	low	low_i/n1/Z	Output Pin (NR4)
CP	CLOCK	CLOCK	Input Port
CD	OPER	OPER	Input Port
Output Pins	Net	Net Load Pins	Load Pin Type
-----	-----	-----	-----
Q	out_2	i1/low3/n1/B	Input Pin (NR4)
		out_2	Output Port
		i1/low/n1/B	Input Pin (NR4)
		i1/low2/n1/B	Input Pin (NR4)
QN	NET2QNX	i2/low3/n1/B	Input Pin (NR4)
		i2/low/n1/B	Input Pin (NR4)
		i2/low2/n1/B	Input Pin (NR4)

1

SEE ALSO

current_design(2)
current_instance(2)
report_design(2)
report_hierarchy(2)
report_net(2)
report_port(2)
report_reference(2)

report_clock

Reports clock-related information.

SYNTAX

```
string report_clock  
[-attributes]  
[-skew]  
[-groups]  
[-nosplit]  
[clock_names]
```

Data Types

clock_names list

ARGUMENTS

-attributes

Shows clock attributes and provides a list of all the clocks in the current design. The information for each clock includes source type, signal rise and fall times, and attributes. This report is shown by default.

-skew

Reports clock latency (source and network latency) and uncertainty information set on the design by the **set_clock_latency** and **set_clock_uncertainty** commands, respectively. Clock network latency information includes rise latency and fall latency. Clock source latency information includes rise latency and fall latency for early and late arrivals. Clock uncertainty information includes intraclock setup or hold uncertainty and interclock setup or hold uncertainty. This option also reports any fixed clock transition set by using the **set_clock_transition** command. This option only reports active clocks.

-groups

Shows the current setting of clock groups, including the list of active clocks in the current analysis scope and grouping of exclusive clocks and asynchronous clocks set by using the **set_clock_groups** command.

-nosplit

Specifies not to split lines if a column overflows. Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

clock_names

Lists the clocks that must be reported. Substitute the list you want for *clock_names*.

DESCRIPTION

Reports clock-related information. Displays all clock-related information on a design. The report contents are controlled by the options used. If you do not specify any options, the command displays the attributes report.

When a clock is created, it defaults to being active and is added to the active clocks list. The new clock is also added to the other clocks list if some clocks are defined to be exclusive or asynchronous with any other clocks in the design. When a clock is removed, it is removed from its related lists. To see if a clock is active or not, use the **report_clock** command with the *-attribute* option.

If a dynamic component of clock latency has been specified by the **set_clock_latency** command, the *-skew* option can be used to report what values have been set. In this case, all components of clock latency will be.

Some information, such as the waveform of a generated clock or propagated skew, is displayed after a timing update (as a result of manually executing an *update_timing* function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to define all clocks without incurring the cost of a complete timing update.

The report generated by using the **report_transitive_fanout** command with the *-clock_tree* option shows the fanout network of every clock source in the design.

To obtain a list of all clocks in the design, use the **all_clocks** command.

EXAMPLES

Here are some examples of the **report_clock** command:

```
pt_shell> create_clock -period 20.000000\  
[get_ports {CLK}]
```

```
pt_shell> report_clock
```

```
*****  
Report : clock  
Design : counter  
Version: 1997.01-development  
Date   : Fri Jul 26 09:41:39 1996  
*****
```

Attributes:

p - propagated_clock
G - Generated clock

Clock	Period	Waveform	Attrs	Sources
CLK	20.00	{0 10}		{CLK}

```
pt_shell> set_clock_latency 2.4 [get_clocks CLK]
```

```

pt_shell> set_clock_latency 0.17 -source -early [get_clocks CLK]
pt_shell> set_clock_latency 0.19 -source -late [get_clocks CLK]
pt_shell> set_clock_uncertainty 1.3 [get_clocks CLK]
pt_shell> set_clock_transition 1.7 [get_clocks CLK]
pt_shell> report_clock -skew

```

```

*****
Report : clock_skew
Design : counter
Version: 1997.01-development
Date   : Fri Jul 26 17:13:38 1996
*****

```

Object	Rise Delay	Fall Delay	Hold Uncertainty	Setup Uncertainty
CLK	2.40	2.40	1.3	1.3

Source Latency				
Object	Min Rise	Min Fall	Max Rise	Max Fall
CLK	0.17	0.17	0.19	0.19

Object	Rise Transition	Fall Transition
CLK	1.7	1.7

```

pt_shell> set_clock_latency -late -source -dynamic 0.5 4.5 [get_clocks clk]
pt_shell> set_clock_latency -early -source 2.5 -dynamic 0.5 [get_clocks clk]
pt_shell> report_clock -skew

```

```

*****
Report : clock_skew
Design : latency
Version: Z-2007.06-DEV
Date   : Tue Mar 20 08:14:26 2007
*****

```

Object	Min Condition Source Latency				Max Condition Source Latency				Re
	Early_r	Early_f	Late_r	Late_f	Early_r	Early_f	Late_r	Late_f	
l_clk									
clk (static)	2.00	2.00	4.00	4.00	2.00	2.00	4.00	4.00	--
clk (dynamic)	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	--
clk (total)	2.50	2.50	4.50	4.50	2.50	2.50	4.50	4.50	

```
pt_shell> set_clock_groups -ex -group {clk1 clk3}
pt_shell> set_clock_groups -asyn -name a1 -group clk2 -group clk2
pt_shell> set_active_clocks {clk1 clk2}
pt_shell> report_clock -groups
*****
Report : clock_groups
Design : test
Version: 2002.09-alpha
Date   : Thu May  9 13:13:51 2002
*****

Active clocks:
    clk1  clk2

Total exclusive groups: 1.
NAME : clk1_others
      -group {clk1 clk3 }
      -group {clk2 clk4 }

Total asynchronous groups: 1.
NAME : a1
      -group {clk2 }
      -group {clk4 }
```

SEE ALSO

```
all_clocks(2)
create_clock(2)
remove_clock_groups(2)
report_transitive_fanout(2)
set_active_clocks(2)
set_clock_groups(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
```


report_clock_gate_savings

Reports toggle savings on clock gates.

SYNTAX

```
int report_clock_gate_savings  
[-by_clock_gate]  
[-sequential]  
[-hierarchical]  
# [-by_clock]  
[-sort_by]  
[-clocks clock_list]  
[-nosplit]  
[object_list]
```

Data Types

clock_list	list
object_list	list

ARGUMENTS

-by_clock_gate

Specifies that the report is to report on individual clock gates in the design. The command uses toggle rates at the clk_in and clk_out pins of the clock gates to determine the fraction of toggles suppressed by the clock gate.

-sequential

Specifies that the report is to report on registers in the design. When the *-sequential* option is applied, the command uses toggle rates to determine the fraction of clock events at registers that result in a toggle on the Q pin. This fraction is a measure of how well clock gated a register is. Values near 25% are near optimal for datapath registers. Lower values indicate there might be more room for clock gating, as the clock is toggling often when the Q pin toggles little.

In addition to the Q/Clk toggle ratio, the Register Gating Efficiency is reported for each register. This measures the ratio between the toggle rate of the local (gated) clock, and the root clock specified with *create_clock*. High values indicate that most of the toggles from the root clock have been suppressed by clock gates along the clock tree from the root to the register. When the *-by_clock_gate* and *-sequential* options are used together, the command reports on register banks in the design. Registers are grouped into clusters by the clock gates that drive their clock pins. An additional cluster of ungated registers is also reported included in the report.

-hierarchical

Specifies that results are to be aggregated into hierarchical blocks, and each block is to be reported. Without this option, individual clock gates or registers are reported. The data is aggregated by averaging toggle savings over clock gates in a module.

The *-hierarchical* option cannot be used when both the *-by_clock_gate* and *-sequential* options are also used together.

.IP "-by_clock" # This option is the default unless the *-by_clock_gate* or *-sequential* option is used. # # Specifies to report on clocks in the design, and to # report overview information about the registers # that driven by those clocks. The report computes the average # toggle savings over registers driven by the clock, where # the toggle savings for a register is defined as the # ratio of the toggle rate at the clock pin of the register # to the toggle rate of the clock. The toggle savings # is the toggle savings of all the clock gates between # the clock and the register. When a register is # in several clock domains, the register is included with the # faster clock, so that registers are not included twice in the report.

-sort_by sort_method

Specifies the order in which to print design objects. The sort option is available with the *-sequential* and *-by_clock_gate* options, but not with the *-hierarchical* option. The following sorting modes are available:

<i>-by_clock_gate</i>	<i>-sequential</i>
name	name
clock_toggle_rate	clock_toggle_rate
toggle_savings	q_clk_ratio
	q_toggle_rate
	gating_efficiency

-clocks clock_list

Specifies that objects not in the clock domain of one of the given clocks should be excluded from the report.

-nosplit

Specifies that report lines should not be split when names are longer than the space provided in the report. This option can make it easier for scripts to parse the results, but might make it harder for humans to read the columns.

object_list

Specifies that the given cells are to be reported on only. If cells in the list are hierarchical, all clock gates (or registers, if the *-sequential* option is used) under the hierarchical cells are reported on. If the *-sequential* option is used without the *-by_clock_gate* option, any leaf cells in the list should be registers. If the *-by_clock_gate* option is used, or if both the *-by_clock_gate* and *-sequential* options are used, any leaf cells should be individual clock gates.

In the default report, the *object_list* option can also contain clocks, in which case only those clocks are reported on.

DESCRIPTION

This command uses toggle rates in the design to report on the toggle savings on the clock tree from clock gating in the design. To use the command, you should first apply switching activities before the command. In the time based mode, this can be done using the **read_vcd** command followed by the **update_power** command.

In the averaged power mode, apply switching activities by using the **read_saif**, **set_switching_activity**, or **read_vcd** command. In the averaged mode, partial switching activity annotation causes the tool to propagate switching activities.

report_clock_gate_savings

The usefulness of the **report_clock_gate_savings** command is largely dependent on the quality of the switching activities. To be useful, the activities should come from simulation. The testbench used from simulation should be designed to simulate the typical behavior of the design and should cover all modes that are part of the typical behavior. If a mode is left off, some part of the design might not be exercised.

The default report displays overview information about each clock and the registers that the clock drives. You can use this report to summarize the overall effectiveness of clock gating in the design. The report computes the average toggle savings over the registers driven by each clock, where here the toggle savings for a register is defined as the ratio of the toggle rate at the clock pin of the register to the toggle rate of the clock. A 20% toggle savings means that the toggle rate at clock pin of a register is 80% of the toggle rate of the main clock for the clock domain. When a register is in several clock domains, the register is only included with the domain of the faster clock, so that registers are not included twice in the report.

The *-by_clock_gate* report can help you determine which clock gates are the most useful in terms of saving toggles. It can also point you to locations where clock gates save few toggles. You are able to improve toggle savings if adjusting the RTL is an option, or it might save power to remove these clock gates. You should be careful when viewing individual clock gates with low toggle savings, as this situation may be a function of the testbench rather than the design itself. The report shows the toggle savings for each clock gate. The toggle savings for a clock gate is defined by the ratio of the toggle rate on the output clock pin of the clock gate, to the toggle rate on the input clock of the clock gate. a 20% toggle savings means that the clock gate suppressed 20% of the incoming toggles.

The *-sequential* reports helps to identify registers with many clk events but few Q toggles.

When the Q/Clk ratio for registers is significantly lower than 25%, there can be room to improve the clock gating for these registers by adding clock gating or increasing the fraction of time that clock gating suppresses clock toggles. A brief explanation of why 25% can be regarded as optimal is below.

In the case of a datapath, during cycles when the datapath is exercised, the clock gating is enabled, and the probability of Q toggling in a cycle is approximately 50%, assuming random and independent data is moving through the datapath. On cycles when the datapath is not being exercised, the clock gating should be disabled. In this idealized situation, the Q/clk toggle ratio should be 25%, since the clk toggles twice in a cycle, and the Q toggles on average once every other cycle (a 1:4 ratio).

There are some exceptions where registers can have Q/Clk ratios larger than 25%, such as low bits of counters where the Q should toggle every cycle when the counter is active. However, on the whole, most registers in the design should report (using the *-sequential* option) a Q/Clk ratio less than 25%.

More information can be gathered about individual clock gates and their associated registers when the *-by_clock_gate* and *-sequential* options are used together. In this case, Individual clock gates are reported, with each clock gate followed by a list of the registers driven by the clock gate. This report can be more useful than the *-sequential* report alone in helping find register banks with low Q/Clk ratios, where

it is possible to disable the clock gate more often. The report can also help find register banks that should be split; it might be that a portion of the registers driven by a clock gate have low Q/Clk ratios, and these registers might be able to have their own separate clock gate, which could allow disabling the clock more often for those registers.

The *-hierarchical* reports (which can be used with either the *-by_clock_gate* or *-sequential* option) can help to identify modules of interest. Since many register banks are too small for clock gating, the hierarchical sequential report might be less useful as the register banks of interest are overwhelmed in the aggregated data by the small register banks which are not clock gated.

USAGE

You can use the default report to view the overall effectiveness of clock gating in the design.

EXAMPLES

The following example shows the default report:

```
report_clock_gate_savings
*****
Report : Clock Gate Savings
        power_mode: Averaged
Design : top
*****

-----
Clock: clk
+ Clock Toggle Rate: 0.2
+ Number of Registers: 200
+ Number of Clock Gates: 29
+ Average Register Gating Efficiency (savings with respect to root clock): 48.1%
-----
```

Toggle Savings Distribution	Number of Registers	% of Registers
100%	8	4.0%
80% - 100%	13	6.5%
60% - 80%	71	35.5%
40% - 60%	71	35.5%
20% - 40%	11	5.5%
0%	24	12.0%

The following example shows the *-by_clock_gate* report on a particular module. There are two individual clock gates, with toggle rates shown.

```
pt_shell> report_clock_gate_savings -by_clock_gate [get_cell TOP/A]
```

```
*****
```

```
Report : Clock Gate Savings
```

```
Design : mydesign
```

```
        power_mode: Averaged
```

```
        -by_clock_gate
```

```
Version: B-2008.12
```

```
Date   : Thu Nov 20 14:08:20 2008
```

```
*****
```

```
-----
```

Clock Gate	Toggle Savings (%)	IN clk toggle rate	OUT clk toggle rate
TOP/A/U24	37.3%	0.5	0.3135
TOP/A/U25	13.3%	0.5	0.4335

```
-----
```

```
1
```

The following example shows the hierarchical report for clock gates.

```
pt_shell> report_clock_gate_savings -by_clock_gate -hierarchical
```

```
*****
```

```
Report : Clock Gate Savings
```

```
Design : mydesign
```

```
        power_mode: Averaged
```

```
        -hierarchical
```

```
Version: B-2008.12
```

```
Date   : Thu Nov 20 14:08:20 2008
```

```
*****
```

```
-----
```

Module	Number Clock Gates	Toggle Savings (%)
TOP	225	67%
/A	125	50%
//B	30	12%
/C	100	80%

```
-----
```

```
1
```

The following report shows individual registers (not register banks). The clk/q ratio of 4 is expected for a register clocked every cycle, with random data on the d pin of the register.

```
pt_shell> report_clock_gate_savings -sequential
```

```
*****
```

```
Report : Clock Gate Savings
```

```
Design : mydesign
```

```
power_mode: Averaged
```

```
-sequential
```

```
Version: B-2008.12
```

```
Date : Thu Nov 20 14:08:20 2008
```

```
*****
```

Register		Root Clock	Local Clock	Q	Q/	
Clk	Register	Toggle	Toggle	Toggle	Toggle	Gating
		Rate	Rate	Rate	Ratio	Effici

TOP/A/U34		0.2	0.1	0.025	25%	50%
TOP/A/U35		0.2	0.1	0.01	10%	50%

1						

The following example shows the sequential hierarchical report, which averages over registers in each hierarchical block.

```
pt_shell> report_clock_gate_savings -by_sequential -hierarchical
```

```
*****
```

```
Report : Clock Gate Savings
```

```
Design : mydesign
```

```
power_mode: Averaged
```

```
-sequential
```

```
-hierarchical
```

```
Version: B-2008.12
```

```
Date : Thu Nov 20 14:08:20 2008
```

```
*****
```

Module	Number of Registers	Q/Clk Toggle Ratio	Avg Register Gating Efficiency

TOP	2250	17.5%	40.6%
A	1250	15.6%	37%

B	300	23.3%	70%
C	1000	15.6%	45%

1

SEE ALSO

read_vcd(2)
read_saif(2)
set_switching_activity(2)
update_power(2)

report_clock_gating_check

Displays clock gating check information about specified pins.

SYNTAX

```
int report_clock_gating_check  
[-significant_digits digits]  
[-nosplit]  
[object_list]
```

Data Types

object_list list

ARGUMENTS

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. The *-nosplit* option prevents line splitting and facilitates writing software to extract information from the report output.

object_list

Specifies a list of cells, pins, clocks, or design objects for report.

DESCRIPTION

The **report_clock_gating_check** command displays the following information for the clock gating checks. Information displayed about the specified objects: Instance of the cell, enable pin, clock pin, setup/hold time for rise, setup/hold time for fall, user-specified high/low active waveform. The '*' marker in the high/low field means the user specified high/low overrides what PrimeTime inferred from logic of the cell. The attributes show where the clock gating check originally defined.

I

Automatically inferred by PrimeTime.

P

Power Compiler inserted the check.

L

Defined by library cell.

EXAMPLES

The following example displays how to report all clock gating check informations.

```
pt_shell> report_clock_gating_check
```

```
*****
Report : clock gating check
Design : flop
Version: 2001.08-SI1
Date   : Fri May 18 16:45:56 2001
*****
```

Cell	Enable	Clock	Rise		Fall		High/Low	Attr
			Setup	Hold	Setup	Hold		
mid	A	B	5.00	3.00	5.00	3.00	High	I
MX	A	S	5.00	3.00	5.00	3.00	High	I
MX	B	S	5.00	3.00	5.00	3.00	Low (*)	I

Note: * indicates user override of tool inferred controlling value
Attr: I:auto inferred, P:power compiler inserted, L:library cell defined

1

SEE ALSO

```
set_clock_gating_check(2)
report_constraint(2)
report_timing(2)
report_default_significant_digits(3)
```

report_clock_timing

Reports timing attributes of clock networks.

SYNTAX

```
string report_clock_timing
-type report_type
[-clock clock_list]
[-from_clock from_clock_list]
[-to_clock to_clock_list]
[-from from_list]
[-to to_list]
[-setup] | [-hold]
[-launch] | [-capture]
[-rise] | [-fall]
[-nworst worst_entries]
[-greater_than lower_limit]
[-lesser_than upper_limit]
[-slack_lesser_than slack_upper_limit]
[-include_uncertainty_in_skew]
[-verbose]
[-significant_digits digits]
[-show_clocks]
[-crosstalk_delta]
[-derate]
[-variation]
[-sort_by sort_attr]
[-nosplit]
[-pre_commands pre_command_string]
[-post_commands post_command_string]
```

Data Types

<i>report_type</i>	string
<i>clock_list</i>	list
<i>from_clock_list</i>	list
<i>to_clock_list</i>	list
<i>to_list</i>	list
<i>from_list</i>	list
<i>worst_entries</i>	int
<i>lower_limit</i>	float
<i>upper_limit</i>	float
<i>slack_lower_limitf</i>	float
<i>sort_attr</i>	string
<i>digits</i>	int
<i>pre_command_string</i>	string
<i>post_command_string</i>	string

ARGUMENTS

-type *report_type*

Specifies the type of report to be generated. Allowed values are as follows:

report_clock_timing

transition – To specify a transition time report.

latency – To specify a latency report.

skew – To specify a skew report; you cannot use the *-launch*, *-capture*, *-rise*, *-fall*, and *-lesser_than* options if you specify a skew report, and you can use the *-include_uncertainty_in_skew* option only in a skew, interclock_skew, or summary report.

interclock_skew – To specify an interclock skew report; you cannot use the *-launch*, *-capture*, *-rise*, *-fall*, and *-lesser_than* options if you specify an interclock skew report, and you can use the *-include_uncertainty_in_skew* option only in a skew, interclock_skew or summary report.

summary – To specify a summary report, which shows the worst instances of transition time, latency and skew over the clock networks or subnetworks of interest. If you specify a summary report, you can use only the *-clock*, *-to_list*, *-from_list*, *-include_uncertainty_in_skew*, *-nosplit*, and *-significant_digits* options.

For nonsummary reports, report entries are ordered with respect to the specified attribute of interest (transition time, latency, or skew). Note that all skews reported are "local" skews. For an explanation of local skew, see the DESCRIPTION section.

-clock clock_list

Specifies a list of clock networks to be used in the report. A subreport is typically produced for every clock in the *clock_list*, unless you additionally specify the *to_list*, *from_list*, or both options that has no network intersection with a given clock. In that case, PrimeTime drops these clocks from the *clock_list* and also issues a warning. By default, if you do not specify *clock_list*, all clocks in the design that have associated clock networks are used in the report.

-from_clock from_clock_list

Specifies a list of clock networks to be used as from-clocks in the current interclock skew report. This option may be used only in an interclock skew report. The report typically considers every clock in *from_clock_list*, unless you additionally specify the *from_list* option that has no network intersection with a given clock. In that case, PrimeTime drops these clocks from the *from_clock_list* and also issues a warning. By default, if you do not specify the *from_clock_list* option, all clocks in the design that have associated clock networks are used as from-clocks in the report.

-to_clock to_clock_list

Specifies a list of clock networks to be used as to-clocks in the current interclock skew report. This option may be used only in an interclock skew report. The report typically considers every clock in *to_clock_list*, unless you additionally specify a *to_list* that has no network intersection with a given clock. In that case, PrimeTime drops these clocks from the *to_clock_list* and also issues a warning. By default, if you do not specify the *to_clock_list* option, all clocks in the design that have associated clock networks are used as to-clocks in the report.

-to to_list

Specifies the list of sequential clock network pins or ports to consider as to-pins in the current report. If any named pin is not the clock pin of a sequential device, such as a sink pin, PrimeTime replaces that pin with all sequential clock pins in the transitive fanout of the named pin. If there are no sequential clock pins in the pin's transitive fanout, the pin is dropped

from the list with a warning message.

For skew reports, the from-to skew sense is defined by the pins in the *from_list* and *to_list* options respectively; if the *to_list* option is not specified, by default all sink pins in the given clock networks are used. Thus, specifying the *to_list* option reduces the topological scope of the report. For transition time and latency reports, the *from_list* and *to_list* options are concatenated and treated as a single list; if neither is specified, the *to_list* option is assumed to be populated with all sink pins in the given clock networks.

-from *from_list*

Specifies a list of sequential clock network pins or ports to consider as from-pins in the current report. If any named pin is not the clock pin of a sequential device, PrimeTime replaces that pin with all sequential clock pins in the transitive fanout of the named pin. If there are no sequential clock pins in the pin's transitive fanout, the pin is dropped from the list and a warning message is issued.

For skew reports, the from-to skew sense is defined by the pins in the *from_list* and *to_list* options respectively; if the *from_list* option is not specified, by default all sink pins in the given clock networks are used. Thus, specifying the *to_list* option reduces the topological scope of the report. For transition time and latency reports, the *from_list* and *to_list* options are concatenated and treated as a single list; if neither is specified, the *to_list* option is assumed to be populated with all sink pins in the given clock networks.

-setup

Indicates that only the setup data path is to be used in the report, and that the skews, latencies or transition times reported must correspond to those used by the **report_timing** command in the verification of setup constraints. The *-setup* and *-hold* options are mutually exclusive. If neither option is specified, the *-setup* option is assumed. You cannot use this option in summary reports.

-hold

Indicates that only the hold data path is to be used in the report, and that the skews, latencies, or transition times reported must correspond to those used by the **report_timing** command in the verification of hold constraints. The *-setup* and *-hold* options are mutually exclusive. If neither option is specified, the *-setup* option is assumed. You cannot use this option in summary reports.

-launch

Indicates that only pins launching data are to be used in the report, and that latencies or transition times reported must correspond to those used by the **report_timing** command for sequential device clock pins that are launching data. In skew reports, the role is implicit from the from-to sense indicated by the *from_list* and *to_list* options. In all other reports, the *-launch* and *-capture* options are mutually exclusive. If neither option is specified, the *-launch* option is assumed. You cannot use this option in summary or skew reports.

-capture

Indicates that only pins capturing data are to be used in the report, and that the latencies or transition times reported must correspond to those used

by the **report_timing** command for sequential device clock pins that are capturing data. In skew reports, the role is implicit from the *from_list* and *to_list* options. In all other reports, the *-launch* and *-capture* options are mutually exclusive. If neither option is specified, the *-launch* option is assumed. You cannot use this option in summary or skew reports.

-rise

Indicates that the active transition is a rising edge for sequential device clock pins in the current report. The *-rise* and *-fall* options are mutually exclusive; if neither option is specified, the active transition at a latch or flip-flop is deduced from the launch or capture role and the behavior of the sequential device itself. This option enables you to answer "what if" questions regarding latency and transition time at sequential device clock pins. You cannot use this option in summary or skew reports.

-fall

Indicates that the active transition is a falling edge for sequential device clock pins in the current report. *-rise* and *-fall* are mutually exclusive; if neither is specified, the active transition at a latch or flip-flop is deduced from the launch or capture role and the behavior of the sequential device itself. This option enables you to answer "what if" questions regarding latency and transition time at sequential device clock pins. You cannot use this option in summary or skew reports.

-nworst *worst_entries*

Specifies the number of worst report entries to be reported per clock domain; the default is 1. Entries are sorted with respect to the attribute of interest (transition time, latency or skew) specified with the *-type* option of the *report_type* command.

The worst entries are those most likely to cause a violation. For example, if you request a latency report using *-setup* and *-capture*, the smallest *worst_entries* are listed, sorted in ascending order, because small capture latencies for setup paths are more likely to lead to a violation than large capture latencies. By contrast, for skew reports, the worst entries always correspond to the largest skews over the specified domain. You cannot use this option in summary reports.

-greater_than *lower_limit*

Indicates that only those entries whose attribute value (latency, transition time or skew) is greater (more positive) than *lower_limit* are to be shown. You cannot use this option in summary reports.

-lesser_than *upper_limit*

Indicates that only those entries whose attribute value (latency, transition time or skew) is less (more negative) than *upper_limit* are to be shown. You cannot use this option in summary or skew reports.

-slack_lesser_than *slack_upper_limit*

Indicates that only those entries whose slack value is less (more negative) than *slack_upper_limit* are to be shown. For skew report entries slack is the worst slack over all paths launched by the from-pin and captured by the to-pin. For transition time or latency report entries, slack is defined as the worst slack of all paths either launched by a transition at the sink pin (if the *-launch* option is used) or captured by a transition at the sink pin (if

the *-capture* option is used).

Note that the use of this filter might greatly increase the run-time of this report. However, when used with the *-capture* option the effect on runtime should be small, since PrimeTime is able to make use of cached slack values to avoid expensive recomputations. You cannot use this option in summary reports.

-include_uncertainty_in_skew

Indicates that the user-defined uncertainty between the sequential devices in a launch/capture pair is to be considered a component of skew. You cannot use this option in summary or skew reports.

-verbose

Indicates that a more detailed report is to be generated; instead of a single line per pin, verbose reports trace the entire source-to-sink path through a clock network to show how the final reported attribute (skew, latency or transition time) was accumulated over the course of the path. You cannot use this option in summary reports.

-show_clocks

Indicates that the launching and capturing clocks are to be shown for every interclock skew entry in the report. This is useful if *from_clock_list* or *to_clock_list* contain more than one clock each. You cannot use this option in interclock skew reports.

-derate

Indicates that derate factors are to be shown in the report. By default derate factors are not displayed. Derate factors are only shown when the *-verbose* option is specified.

-variation

Indicates that the report is to be augmented with variation information - use of this switch is allowed only when detailed variation-aware clock analysis is enabled. Without *-variation*, only the quantile or mean value will be displayed, as specified using the **variation_derived_scalar_attribute_mode** variable. With *-variation*, the quantile, mean and either the standard deviation (for skew reports) or sensitivity (for latency or transition reports) is displayed for both verbose and non-verbose reports. In verbose reports, additional statistical info regarding individual path increments appear in new columns, namely the sensitivity, mean and increment, where the increment refers either to the quantile or the effective delay depending on the **variation_report_timing_increment_format** variable.

-sort_by

Indicates that the entries in the report are to be sorted with respect to the specified variational attribute. This switch can be used only with the *-summary* option and only when detailed variation-aware clock analysis is enabled. Note that this switch only controls the ranking of entries and not their display. Allowed values are as follows:
quantile - Rank entries using their quantile values.
mean - Rank entries using their mean values.
sensitivity - Rank entries using their sensitivity values. This options may not be used in a skew or interclock_skew report.

-crosstalk_delta

Indicates that delta delay and delta transition time are to be shown in verbose reports. The deltas are computed during crosstalk signal integrity analysis, or they can be annotated manually using the **set_annotated_delay -delta_only** and **set_annotated_transition -delta_only** commands. Note that the **-crosstalk_delta** option reports only the calculated or annotated deltas, it does not initiate crosstalk analysis. Only deltas on input pins are shown.

-nosplit

Prevents line-splitting and facilitates writing software to extract information from the report output. If this option is not used, most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-pre_commands *pre_command_string*

This option is available only if you invoke PrimeTime with the **-multi_scenario** option. This option allows you to specify a string of commands to be executed in the slave context before the execution of the merged reporting command. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

-post_commands *post_command_string*

This option is available only if you invoke PrimeTime with the **-multi_scenario** option. This option allows you to specify a string of commands to be executed in the slave context after the execution of the merged reporting commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

DESCRIPTION

This command generates a report of clock timing information for the current design.

There are several reporting types provided, which allow you to examine skew, latency and transition time attributes of a given clock network or sub-network at various levels of generality. By default, the report displays the values of these attributes only at sink pins (that is, the clock pins of sequential devices) of the clock network. You use the **-verbose** option to display source-to-sink path traces. If you specify several clock domains, the **report_clock_timing** command generates a separate subreport for each clock domain.

At the highest level of abstraction is the summary style report, which provides only a list of maxima and minima of the skew, latency, and transition time attributes over the given networks. At a lower level of abstraction are the transition, latency, and skew type reports, called list style reports, in which you can sort,

filter and display the worst set of sink pins in the given network with respect to a single attribute of interest. For skew reports, each report entry is a pair of sink pins and their relative skew. For transition time or latency reports, each entry corresponds to a single sink pin. The lowest level of abstraction is provided by verbose mode, which replaces every sink pin in a list style report by a corresponding source-to-sink path trace.

In both summary and list style reports, the rightmost column is an attributed column. Corresponding to each sink pin, the set of character symbols in this column indicates the following:

Symbol	Meaning
-----	-----
r	Rising transition
f	Falling transition
p	Propagated clock to this pin
i	Clock inversion to this pin
-	Launching transition
+	Capturing transition
m	Library clock insertion delay

In verbose mode, back-annotations on path elements in the timing path are indicated using a character symbol. For definitions of these character symbols, see the manual page of the **report_timing** command.

Skews reported by **report_clock_timing** are local skews only. Local skew exists from one sink pin to another only as long as their associated sequential devices are connected via a data path in the appropriate from-to sense. Note that even if all data paths between the sequential devices are false because of user-defined exceptions, the skew still qualifies as local skew if the devices are connected topologically.

If detailed variation-aware clock analysis is enabled using the **variation_analysis_mode** variable, the **report_clock_timing** command evaluates the attributes of latency, transition time and skew as statistical entities. By default, the quantile of these attributes is used for both ranking and display purposes. The ranking can be controlled using the **-sort_by** switch, while the display parameter can be changed from quantile to mean using the **variation_derived_scalar_attribute_mode** variable. To augment the display with additional statistical parameters of **report_clock_timing** attributes, see the **-variation** switch.

EXAMPLES

The following example shows a typical summary style report.

```
pt_shell> report_clock_timing -type summary
```

```
*****
Report : clock timing
        -type summary
        -nworst 1
```


Design : xor_testcase
Version: 2002.03-SI1
Date : Tue Nov 20 06:17:33 2001

Clock: CK1

Maximum setup launch latency: flop9/CP	3.08	rp-+
Minimum setup capture latency: or2_3/B	1.15	fpi-+
Minimum hold launch latency: or2_3/B	1.15	fpi-+
Maximum hold capture latency: flop9/CP	3.08	rp-+
Maximum active transition: or2_3/B	0.20	fpi-+
Minimum active transition: or2_3/B	0.09	fpi-+
Maximum setup skew: flop9/CP		rp-+
flop10/CP	0.87	rp-+
Maximum hold skew: flop9/CP		rp-+
flop10/CP	-0.21	rp-+

The following example displays the five worst setup skews in the clock network of CLK1, taking uncertainty into account.

```
pt_shell> report_clock_timing -clock CLK1 -type skew -setup \
-nworst 5 -include_uncertainty_in_skew
```

```
*****
Report : clock timing
        -type skew
        -nworst 5
        -setup
        -include_uncertainty_in_skew
Design : multi_domain
Version: 2002.03-SI1
Date : Tue Nov 20 06:49:19 2001
*****
```

Clock: CLK1

Clock Pin	Latency	Uncert	Skew	
f2_2/CP	6.11			rp-+
f2_1/CP	2.01	0.11	4.21	fp-+
l3_2/G	4.10			rpi-
f1_2/CP	1.00	0.22	3.32	rpi-+
l2_2/G	4.11			rp-
f1_2/CP	1.00	0.12	3.23	rpi-+
f2_2/CP	6.11			rp-+
l3_3/G	3.01	0.11	3.21	rp-
l1_3/G	5.11			rp-
f2_1/CP	2.01	0.11	3.21	fp-+

The following displays the five worst launching latencies for hold paths in the clock network of CLK2.

```
pt_shell> report_clock_timing -clock CLK2 -hold -launch -nworst 5 \
        -type latency
```

```
*****
Report : clock timing
        -type latency
        -launch
        -nworst 5
        -hold
Design : multi_domain
Version: 2002.03-SI1
Date   : Tue Nov 20 06:55:56 2001
*****
```

Clock: CLK2

Clock Pin	Trans	Source	Latency Network	Total	
f1_2/CP	0.04	0.00	1.00	1.00	rpi-+
f1_3/CP	0.00	0.01	1.00	1.01	fp-+
f2_1/CP	0.01	0.01	2.00	2.01	rp-+
f1_1/CP	0.00	0.00	3.00	3.00	rpi-+
f3_2/CP	0.00	0.00	3.00	3.00	fpi-+

The following example demonstrates how to request a verbose report showing the worst

local skew from f2_2/CP to any other sink pin.

```
pt_shell> report_clock_timing -type skew -verbose -from f2_2/CP
```

```
*****
```

Report : clock timing

-type skew

-verbose

-nworst 1

-setup

Design : multi_domain

Version: 2002.03-SI1

Date : Tue Nov 20 07:00:56 2001

```
*****
```

Clock: CLK1

Startpoint: f2_2 (rising edge-triggered flip-flop clocked by CLK1)

Endpoint: f2_1 (rising edge-triggered flip-flop clocked by CLK1)

Point	Trans	Incr	Path

clock source latency		0.11	0.11
clk2 (in)	0.00	0.00	0.11 r
az_1/Z (B1I)	0.09	1.00 H	1.11 r
az_2/Z (B1I)	0.13	1.00 H	2.11 r
bf2_2_1/Z (B1I)	0.02	1.00 H	3.11 r
if2_2_1/Z (IVA)	0.44	1.00 H	4.11 f
bf2_2_2/Z (B1I)	0.01	1.00 H	5.11 f
if2_2_2/Z (IVA)	0.13	1.00 H	6.11 r
f2_2/CP (FD1)	0.13	0.00	6.11 r
startpoint clock latency			6.11
clock source latency		0.01	0.01
clk2 (in)	0.00	0.00	0.01 r
az_1/Z (B1I)	0.02	1.00 H	1.01 r
az_3/Z (B1I)	0.01	1.00 H	2.01 r
f2_1/CP (FD1)	0.01	0.00	2.01 r
endpoint clock latency			2.01

startpoint clock latency			6.11
endpoint clock latency			-2.01

skew			4.10

The following example traces the two worst launch latencies for hold paths in the clock network of CLK1.

```
pt_shell> report_clock_timing -type latency -hold -verbose \
```

```
-nworst 2 -clock CLK1
```

```
*****
```

```
Report : clock timing
        -type latency
        -verbose
        -launch
        -nworst 2
        -hold
Design : multi_domain
Version: 2002.03-SI1
Date   : Tue Nov 20 07:14:28 2001
*****
```

Clock: CLK1

Endpoint: f1_2 (rising edge-triggered flip-flop clocked by CLK1')

Point	Trans	Incr	Path
clock source latency		0.00	0.00
clk1 (in)	0.00	0.00	0.00 f
if1_2_1/Z (IVA)	0.04	1.00 H	1.00 r
f1_2/CP (FD1)	0.04	0.00	1.00 r
total clock latency			1.00

Endpoint: f1_3 (rising edge-triggered flip-flop clocked by CLK1)

Point	Trans	Incr	Path
clock source latency		0.01	0.01
clk1 (in)	0.00	0.00	0.01 r
bf1_3_1/Z (B1I)	0.00	1.00 H	1.01 r
f1_3/CP (FD1)	0.00	0.00	1.01 r
total clock latency			1.01

The following example makes use of a slack filter to display the worst three skews between latches whose latch-to-latch paths are violating.

```
pt_shell> report_clock_timing -type skew -nworst 3 \
        -slack_lesser_than 0.0
```

```
*****
Report : clock timing
        -type skew
        -slack_lesser_than 0.00
        -nworst 3
        -setup
Design : multi_domain
Version: 2002.03-Beta2
Date   : Fri Dec 14 09:11:05 2001
*****
```

Clock: CLKA

Clock Pin	Latency	CRP	Skew	Slack

f2_2/CP	6.01				rp-+
f2_1/CP	2.11	-0.03	3.87	-1.49	rp-+
l2_2/G	4.01				rp-
f1_2/CP	1.10	-0.21	2.70	-6.64	rpi-+
l1_3/G	5.01				rp-
f2_1/CP	2.11	-0.32	2.58	-1.63	rp-+

The following requests the two largest setup skews for paths both launched and captured by any clock networks in the list *\$my_clocks*.

```
pt_shell> report_clock_timing -type interclock_skew \
          -from_clock $my_clocks -to_clock $my_clocks \
          -nworst 2 -include_uncertainty_in_skew -show_clocks
```

```
*****
```

```
Report : clock timing
-type interclock_skew
-nworst 2
-setup
-include_uncertainty_in_skew
-show_clocks
```

```
Design : my_design
Version: 2002.09-alpha
Date   : Thu May  2 10:55:20 2002
```

```
*****
```

```
Number of startpoint pins   : 907
Number of endpoint pins     : 907
Number of startpoint clocks : 5
Number of endpoint clocks   : 5
```

Clock Pin	Latency	Uncert	Skew	
orig_clk				
orig_if/ram_pdp1/FFB35/CK	1016.72			rp-+
dcd_ram/ram_clk				
dcd_ram/dcd_ram/RAM/CKRB	149.60	195.00	1062.12	rp-+
comp_clk				
comp_if/c_port_if/edge_trig_reg/CK	1400.42			rp-+
comp_clk				
comp_if/c_port_if/posi/iq_reg/CK	1159.09	199.00	440.34	rp-+

The following reports the largest two setup skews in the domain of CLKB in the context of a detailed variation-aware clock analysis, and requests the augmentation of the report with variation information.

```
pt_shell> report_clock_timing -type skew -clock CLKB -variation
```

```
*****
Report : clock timing
-type skew
-nworst 2
      -variation
Design : multi_domain
Version: Z-2006.12-Beta2
Date   : Thu Oct 19 10:14:46 2006
*****
```

Clock: CLKB

Clock Pin	Latency	Stddev	--- Skew --- Mean	Quant	
f2_2/CP	0.40221				rp-+
f2_1/CP	0.13533	0.02825	0.26688	0.34732	rp-+
12_1/CP	0.33036				rpi-+
12_2/CP	0.18638	0.02626	0.14398	0.20948	rp-+

SEE ALSO

```
report_clock(2)
report_timing(2)
set_clock_uncertainty(2)
report_default_significant_digits(3)
timing_remove_clock_reconvergence_pessimism(3)
variation_enable_analysis(3)
variation_analysis_mode(3)
variation_derived_scalar_attribute_mode(3)
```

report_constraint

Displays constraint-related information about a design.

SYNTAX

```
int report_constraint
[-all_violators]
[-verbose]
[-path_type format]
[-max_delay]
[-min_delay]
[-max_capacitance]
[-min_capacitance]
[-max_transition]
[-min_transition]
[-max_fanout]
[-min_fanout]
[-min_pulse_width]
[-min_period]
[-pulse_clock_min_width]
[-pulse_clock_max_width]
[-pulse_clock_min_transition]
[-pulse_clock_max_transition]
[-recovery]
[-removal]
[-max_skew]
[-clock_gating_setup]
[-clock_gating_hold]
[-clock_separation]
[-connection_class]
[-ignore_register_feedback feedback_slack_cutoff]
[-significant_digits digits]
[-nosplit]
[-pre_commands pre_command_string]
[-post_commands post_command_string]
[object_list]
```

Data Types

<i>format</i>	string
<i>digits</i>	int
<i>feedback_slack_cutoff</i>	float
<i>pre_command_string</i>	string
<i>post_command_string</i>	string
<i>object_list</i>	list

ARGUMENTS

-all_violators

Indicates that a summary is to be displayed showing the worst violation per endpoint of each violated design rule constraint in the current design. The -verbose option provides detailed information on each constraint violation.

Multiple violations for a given constraint are listed from the greatest to the least violator.

-verbose

Indicates that more detail is to be shown about constraint calculations.

-path_type *format*

Specifies the format for the path report. Allowed values are *slack_only* (the default), and *end*. This option has an effect only if the *-verbose* option is not used. If *slack_only* is specified, the report displays only endpoint slacks. If *end* is specified, the report has a column format that shows one line for each path, with only the endpoint path total, required-time, and slack.

-max_delay

Indicates that only maximum delay and setup information is to be displayed. In the default report, the worst violation per clock group is displayed. The default constraint report displays all timing and design rule constraints.

-min_delay

Indicates that only minimum delay and hold information is to be displayed. In the default report, the sum of all violations per clock group is displayed. The default constraint report displays all timing and design rule constraints.

-max_capacitance

Indicates that only maximum capacitance constraint information is to be displayed. The *-max_capacitance* option is a design rule used to limit total capacitance on a net. The *-max_capacitance* option displays the max_capacitance cost (the sum of all maximum capacitance violations). To see details about the worst violator, use the *-verbose* option in addition to the *-max_capacitance* option. To see details about all max_capacitance violations, use the *-all_violators* and *-verbose* options in addition to the *-max_capacitance* option. Note that max_capacitance constraint is not checked for load pin. The default constraint report displays all timing and design rule constraints.

-min_capacitance

Indicates that only minimum capacitance constraint information is to be displayed. The *-min_capacitance* option is a design rule used to limit total capacitance on a net. The default constraint report displays all timing and design rule constraints.

-max_transition

Indicates that only maximum transition constraint information is to be displayed. *-max_transition* is a design rule used to limit transition time on a ports and pins. The default constraint report displays all timing and design rule constraints. If the library uses the cmos2 delay model, maximum edge rate information is shown instead.

-min_transition

Indicates that only minimum transition constraint information is to be displayed. *-min_transition* is a design rule used to set a minimum transition time on a ports and pins. The default constraint report displays all timing and design rule constraints. If the library uses the cmos2 delay model,

maximum edge rate information is shown instead.

-max_fanout

Indicates that only maximum fanout constraint information is to be displayed. The *-max_fanout* option is a design rule used to limit fanout_load on a net. The default constraint report displays all timing and design rule constraints.

-min_fanout

Indicates that only minimum fanout constraint information is to be displayed. The *-min_fanout* option is a design rule used to set a minimum fanout_load on a net. The default constraint report displays all timing and design rule constraints.

-min_pulse_width

Indicates that only minimum pulse width constraint information is to be displayed. The *-min_pulse_width* option is a design rule used to set a minimum pulse width high or low at a clock pin or at pins or ports in the clock network. The default constraint report displays all timing and design rule constraints.

-min_period

Indicates that only minimum period constraint information is to be displayed. The *-min_period* option is a design rule used to set a minimum period on a clock signal. The default constraint report displays all timing and design rule constraints.

-pulse_clock_min_width

Indicates that only pulse clock minimum width constraint information is to be displayed.

-pulse_clock_max_width

Indicates that only pulse clock maximum width constraint information is to be displayed.

-pulse_clock_min_transition

Indicates that only pulse clock minimum transition constraint information is to be displayed.

-pulse_clock_max_transition

Indicates that only pulse clock maximum transition constraint information is to be displayed.

-recovery

Indicates that only recovery constraint information is to be displayed. The *-recovery* option is a timing constraint used to describe the minimum allowable time between the control pin transition to the inactive state, and the active edge of the synchronous clock signal. This time is from the control signal going inactive to the clock edge that latches data in. The asynchronous control signal must remain constant during this time, or an incorrect value may appear at the outputs. In the default report, the worst violation in the design is displayed. The default constraint report displays all timing and design rule constraints.

-removal

Indicates that only removal constraint information is to be displayed. The *-removal* option is a timing constraint used to describe the minimum allowable time between the clock pin inactive edge, while the asynchronous pin is active, to the inactive edge of the same asynchronous control pin. In the default report, the sum of all violations in the design is displayed. The default constraint report displays all timing and design rule constraints.

-max_skew

Indicates that only maximum skew constraint information is to be displayed. The *-max_skew* option is a timing constraint that checks the maximum separation time allowed between two clock signals. The default constraint report displays all timing and design rule constraint.

-clock_gating_setup

Indicates that only clock gating setup constraint information is to be displayed. The *-clock_gating_setup* option is a timing constraint used to set a minimum setup time between a clock and a signal controlling the gating of that clock. In the default report, the worst violation in the design is displayed. The default constraint report displays all timing and design rule constraints.

-clock_gating_hold

Indicates that only clock gating hold constraint information is to be displayed. The *-clock_gating_hold* option is a timing constraint used to set a minimum hold time between a clock and a signal controlling the gating of that clock. In the default report, the sum of all violations in the design is displayed. The default constraint report displays all timing and design rule constraints.

-clock_separation

Indicates that only clock separation constraint information is to be displayed. The *-clock_separation* option is a timing constraint that checks the minimum separation time allowed between two clock signals. The default constraint report displays all timing and design rule constraint.

-connection_class

Indicates to display only connection class constraint information. The *connection_class* constraint is displayed only if there is a *connection_class* violation.

-ignore_register_feedback *feedback_slack_cutoff*

Indicates to ignore any timing path that starts and ends at the same register and holds a value. This option applies to minimum delay as well as maximum delay reports. Only paths with slack less than the specified *feedback_slack_cutoff* are ignored. This option is applied as a filter to the paths after they are generated. Therefore, the number of paths generated may be less than the number specified with the *-nworst* and *-max_paths* options.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

`-pre_commands pre_command_string`

This option is available only if you invoke PrimeTime with the `-multi_scenario` option. This option allows you to specify a string of commands to be executed in the slave context before the execution of the merged reporting command. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

`-post_commands post_command_string`

This option is available only if you invoke PrimeTime with the `-multi_scenario` option. This option allows you to specify a string of commands to be executed in the slave context after the execution of the merged reporting commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

`-nosplit`

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. The `-nosplit` option prevents line splitting and facilitates writing software to extract information from the report output.

`object_list`

Specifies a list of pins and ports in the current design for which constraint related information is displayed. Note that object list can only be specified with the `max_transition`, `max_capacitance`, or both options. The `verbose` option can be used to report detailed constraint calculation.

DESCRIPTION

The **report_constraint** command displays the following information for the constraints on the current design: whether the constraint is violated or met, by how much the constraint value is violated or met, and the design object that is the worst violator.

The maximum delay information shows cost by path group. This includes violations of setup time on registers or ports with output delay as well as violations of **set_max_delay** commands. The total maximum delay cost is the sum of the weighted cost of each group. For information about creating path groups, see the **group_path** man page. To see the current path groups in the design, use the **report_path_group** command.

The minimum delay cost includes violation of hold time on registers or ports with output delay as well as violations of the **set_min_delay** command.

The **report_min_pulse_width** command displays information similar to that displayed by the **report_constraint -min_pulse_width** command.

The **report_pulse_clock_min_width** command displays information similar to that displayed by the **report_constraint -pulse_clock_min_width** command.

The **report_pulse_clock_max_width** command displays information similar to that displayed by the **report_constraint -pulse_clock_max_width** command.

The **report_pulse_clock_min_transition** command displays information similar to that

displayed by the **report_constraint -pulse_clock_min_transition** command.

The **report_pulse_clock_max_transition** command displays information similar to that displayed by the **report_constraint -pulse_clock_max_transition** command.

To enable pulse clock constraint checking, set the **timing_enable_pulse_clock_constraints** variable to *true*. This variable is set to *false* by default. The *object_list* can be optionally specified only with the *max_transition* and *max_capacitance* options. You have the flexibility of using the *verbose* option to report detailed constraint calculation for the objects specified in the object list.

EXAMPLES

The following example shows brief constraint information for the current design.

```
pt_shell> report_constraint
```

```
*****
Report : constraint
Design : counter
Version: 1997.01-development
Date   : Fri Aug  2 17:09:44 1996
*****
```

Group (max_delay/setup)	Cost	Weight	Weighted Cost

CLK	0.00	1.00	0.00
default	0.00	1.00	0.00

max_delay/setup			0.00
Constraint	Cost		

max_delay/setup	0.00	(MET)	
min_delay/hold	0.40	(VIOLATED)	
max_transition	0.00	(MET)	
max_fanout	0.00	(MET)	

The following example displays detailed constraint information for the current design.

```
pt_shell> report_constraint -verbose
```

```
*****
Report : constraint
        -verbose
```

```

Design : counter
Version: 1997.01-development
Date   : Fri Aug  2 17:12:27 1996
*****

```

```

Startpoint: ffb (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

```

Point	Incr	Path

clock CLK (rise edge)	0.00	0.00
startpoint clock skew (ideal)	0.00	0.00
startpoint clock uncertainty	0.00	0.00
ffb/CP (FD3)	0.00	0.00 r
ffb/QN (FD3)	2.42	2.42 r
w/Z (ND4)	0.59	3.01 f
q/Z (EO)	1.13	4.14 f
j/Z (AO2)	1.08	5.22 r
ffd/D (FDS2)	0.00	5.22 r
data arrival time		5.22
clock CLK (rise edge)	10.00	10.00
endpoint clock skew (ideal)	0.00	10.00
endpoint clock uncertainty	0.00	10.00
ffd/CP (FDS2)	0.00	10.00 r
library setup time	-0.90	9.10
data required time		9.10

data required time		9.10
data arrival time		-5.22

slack (MET)		3.88

```

Design: counter

```

max_area	30.00
- Current Area	78.00

Slack	-48.00 (VIOLATED)

The following example displays detailed information on only the worst violation per endpoint for those constraints that have violations.

```

pt_shell> report_constraint -all_violators -verbose

```

```

*****
Report : constraint

```

```

        -all_violators
        -verbose
Design : led
Version: 1997.01-development
Date   : Fri Aug  2 17:12:18 1996
*****

```

```

Startpoint: b (input port)
Endpoint: z5 (output port)
Path Group: default
Path Type: max

```

Point	Incr	Path

input external delay	0.00	0.00 r
b (in)	0.00	0.00 r
U5/Z (IV)	1.32	1.32 f
U3/Z (NR2)	3.35	4.67 r
U18/Z (AO6)	0.73	5.40 f
U22/Z (AO4)	1.42	6.82 r
z5 (out)	0.00	6.82 r
data arrival time		6.82
max_delay	6.50	6.50
output external delay	0.00	6.50
data required time		6.50

data required time		6.50
data arrival time		-6.82

slack (VIOLATED)		-0.32

```

Startpoint: c (input port)
Endpoint: z3 (output port)
Path Group: default
Path Type: max

```

Point	Incr	Path

input external delay	0.00	0.00 r
c (in)	0.00	0.00 r
U6/Z (IV)	1.34	1.34 f
U2/Z (NR2)	3.35	4.69 r
U15/Z (AO7)	0.87	5.56 f
U24/Z (AO3)	1.02	6.57 r
z3 (out)	0.00	6.57 r
data arrival time		6.57
max_delay	6.50	6.50
output external delay	0.00	6.50
data required time		6.50

data required time		6.50

```

data arrival time                                     -6.57
-----
slack (VIOLATED)                                     -0.07

Net: a

max_fanout                5.00
- Fanout                   7.00
-----
Slack                      -2.00  (VIOLATED)

```

The following example shows how to report all max transition violations.

```
pt_shell> report_constraint -max_transition -all_violators
```

```

*****
Report : constraint
        -all_violators
        -path slack_only
        -max_transition
Design : CKMX
Version: V-2004.06-pre-alpha
Date   : Mon Dec 22 17:12:00 2003
*****

max_transition

Pin                Required      Actual      Slack
Transition          Transition
-----
CK2                 0.10         10.00      -9.90 (VIOLATED)
m0/B                0.10         10.00      -9.90 (VIOLATED)
fc/D                0.10         1.75       -1.65 (VIOLATED)

```

The following example shows how to report all min-pulse width violations.

```
pt_shell> report_constraint -min_pulse_width -all_violators
```

```

*****
Report : constraint
        -all_violators
        -min_pulse_width
Design : led
Version: 1997.01-development
Date   : Fri Aug  2 17:12:18 1996
*****

```

sequential_clock_pulse_width

Pin	Required pulse width	Actual pulse width	Slack	
ff1/CP	10.20	10.00	-0.20	(VIOLATED)
ff2/CP	10.20	10.04	-0.16	(VIOLATED)
ff3/CP	2.10	2.00	-0.10	(VIOLATED)
ff3/CP	2.10	2.00	-0.10	(VIOLATED)
ff2/CP	10.00	9.96	-0.04	(VIOLATED)

clock_tree_pulse_width

Pin	Required pulse width	Actual pulse width	Slack	
nand1/Z	10.00	9.58	-0.42	(VIOLATED)
or1/B	10.00	9.58	-0.42	(VIOLATED)
nand1/A	10.20	10.00	-0.20	(VIOLATED)
or1/Z	10.20	10.04	-0.16	(VIOLATED)
or1/Z	10.00	9.96	-0.04	(VIOLATED)

The following example displays how to report all max_skew violations.

```
pt_shell> report_constraint -max_skew -all_violators
```

```
*****
```

```
Report : constraint
        -all_violators
        -path slack_only
        -max_skew
```

```
Design : ms_design
```

```
*****
```

max_skew				
Pin	Required Skew	Actual Skew	Slack	
ff3/c (r->f)	0.15	7.46	-7.31	(VIOLATED)
ff1/c (r->f)	0.15	5.47	-5.32	(VIOLATED)
ff2/c (f->f)	0.14	2.32	-2.18	(VIOLATED)
ff2/c (r->r)	0.12	1.18	-1.06	(VIOLATED)
ff4/c (f->f)	0.14	0.84	-0.70	(VIOLATED)
ff4/c (r->r)	0.12	0.42	-0.30	(VIOLATED)

SEE ALSO

create_clock(2)

report_constraint


```
group_path(2)
report_clock(2)
report_design(2)
report_min_pulse_width(2)
report_path_group(2)
report_timing(2)
report_pulse_clock_min_width(2)
report_pulse_clock_max_width(2)
report_pulse_clock_min_transition(2)
report_pulse_clock_max_transition(2)
set_max_delay(2)
report_default_significant_digits(3)
```

report_context

Reports the characterized timing context information.

SYNTAX

```
string report_context  
[-timing]  
[-environment]  
[-design_rules]  
[-constant_inputs]  
[-nosplit]  
cell_list
```

Data Types

cell_list list

ARGUMENTS

-timing
Reports timing information, such as clocks, input and output delays, and timing exceptions.

-environment
Reports environment related information, such as operating conditions (process, temperature and voltage), wire load model, capacitive loads on input and output pins, and driving cell information on input pins.

-design_rules
Reports characterized design rule information, such as *max_capacitance*, *max_transition*, and *max_fanout*.

-constant_inputs
Reports logic constants propagated to input pins.

-nosplit
Does not split lines if column overflows.

cell_list
Lists cells for which to report the context.

DESCRIPTION

Reports the timing context captured using the **characterize_context** command for the specified list of instances.

Options specify categories of context information to report. All information categories are reported if no option is specified.

EXAMPLES

The following command reports the timing-related context information for instance I1 and I2. The report shows clocks and their waveforms, point-to-point timing exception, input external delays, and output external delays.

The input and output delay information lists the minimum rise, minimum fall, maximum rise, and maximum fall delays. Input and output delay information lists the clock to which the delay is relative. If "(f)" follows the clock name, the delay is relative to the falling edge of the clock; otherwise the delay is relative to the rising edge. If "(l)" follows the clock name, input delay is considered as coming from a level-sensitive latch, or output delay is considered as going to a level-sensitive latch. The default is that the delay is relative to a rising edge-triggered device.

Multiple output delays are listed for a characterized output pin if it has more than one fanout or if the logic it drives terminates in more than one type of latches (that is, edge-triggered (falling or rising clock) or level-sensitive with falling or rising clock).

```
pt_shell> report_context -timing {I1 I2}
```

The following command reports the environment and design rule context information. The report shows design rule checking, wire load models, drive information of input pin, and capacitive load on input and output pins of I2.

```
pt_shell> report_context -env -design_rules I2
```

SEE ALSO

```
characterize_context(2)  
write_context(2)  
remove_context(2)
```

report_crpr

Reports the clock reconvergence pessimism (CRP) calculated between specified register clock pins or ports.

SYNTAX

```
int report_crpr -from from_latch_clock_pin  
-to to_latch_clock_pin  
[-from_clock from_clock]  
[-to_clock to_clock]  
[-setup | -hold]  
[-significant_digits digits]
```

```
string from_latch_clock_pin  
string to_latch_clock_pin  
string from_clock  
string to_clock  
integer digits
```

ARGUMENTS

-from *from_latch_clock_pin*
Specifies the clock pin of the launching sequential device or clock gating check to be reported. A constrained input port can also be used.

-to *to_latch_clock_pin*
Specifies the clock pin of the capturing sequential device or clock gating check to be reported. A constrained output port can also be used.

-from_clock *from_clock*
Specifies the clock that fans out to the launching sequential device.

-to_clock *to_clock*
Specifies the clock that fans out to the capturing sequential device.

-setup
Indicates that the CRP used for a setup data path is to be reported. The **-setup** and **-hold** options are mutually exclusive. If neither option is specified, **-setup** is assumed.

-hold
Indicates that the CRP used for a hold data path is to be reported. The **-setup** and **-hold** options are mutually exclusive. If neither option is specified, **-setup** is assumed.

-significant_digits *digits*
Specifies the number of digits to the right of the decimal point that are to

be reported. Allowed values are 0-13. If this option is not specified the number of significant digits is determined by the **report_default_significant_digits** variable, which has a default value of 2.

DESCRIPTION

Reports the clock reconvergence pessimism (CRP) calculated between specified register clock pins or ports. This command displays the following information about the calculation of the CRP between the two specified sequential elements.

- The name of the common pin in the clock network.
- The clock edges (rising or falling) that propagate through the common point to the launching and capturing registers.
- The value of the **timing_crpr_threshold_ps** variable.
- A tabulation of the arrival times for both clock edges at the common point and their associated CRP (**crp_rise** and **crp_fall**).
- The CRP used along with the reasoning behind the selection of the particular CRP (**crp_rise** or **crp_fall**) used for that report.
- The CRP value used is independent of the CRPR threshold, and therefore is referred to as the exact CRP. The **report_timing** and **report_clock_timing** commands report a CRP value that might be pessimistic by up to the value of the CRPR threshold. If the CRP value used by the **report_timing** or **report_clock_timing** command differs from the exact CRP, this value is also reported.

If the **timing_remove_clock_reconvergence_pessimism** variable is set to false, CRPR is inactive and this command does not work. To determine the current value of this variable, type the following command:

```
printvar timing_remove_clock_reconvergence_pessimism
```

The optional arguments, *from_clock* and *to_clock*, can be used to specify that the command should issue a CRP report for only the specified clocks. Otherwise, a report is issued for all clocks that fanout to each sequential device.

Two CRP values are calculated for each potential common point in the design:

- **crp_rise**, is calculated from rising arrival times at the common point.
 - **crp_fall**, is calculated from falling arrival times at the common point.
- The value of CRP used in the report is dependent on what clock edges propagate through the common point to the clock pins of the launching and capturing devices. The clock edges are included in the CRPR report as "Launching edge at common point", "Capturing edge at common point".

For example if a rising edge through the common point triggers the launching device and also triggers the capturing device then a rising CRP value (`crp_rise`) is used. Similar reasoning can be applied for a falling edge propagating through the common point and subsequently launching and capturing data leading to a falling CRP value being used (`crp_fall`). If a rising edge through the common point launches the data and a falling edge through the common point captures it, a mismatch in sense occurs. In this case, the **timing_clock_reconvergence_pessimism** variable comes into play. This variable allows the values: **normal** and **same_transition**. To determine the current value of this variable, type the following command:

```
printvar timing_clock_reconvergence_pessimism
```

If a mismatch occurs and the variable is set to **normal**, the minimum of `crp_rise` and `crp_fall` is used. If a mismatch occurs and the variable is set to **same_transition**, the CRP is reported as zero. This selection process is reported in the selection details section of the report.

If either clock edge is reported as being "RISING or FALLING", there is a non-unate path between the common point and that clock pin, such that botch edge directions result in active clock edges at the clock pin.

If dynamic information is enabled, two sets of arrival totals are tabulated. Dynamic information can result from SI delta delays, dynamic clock latencies or dynamic rail voltages. In this case, the two sets of arrival totals tabulated are those computed with and without dynamic information being available. The arrival totals computed without dynamic information are equivalent to those used in a PrimeTime analysis with no dynamic information available.

The additional set of arrivals result in four CRP values from which the CRP value used should be chosen. These four CRP values can be summarized as rise/fall CRP and rise/fall dynamic CRP. The rise/fall decision is made based on clock edges incident at the clock edge as before. The decision over whether or not to use dynamic CRP is based on the temporal separation of launching and capturing clock edges. It is only valid to use dynamic CRP if the clock edge launching and capturing data occurs at the same time.

In the case when the capturing sequential device is a level-sensitive latch, two CRP values are reported. The first corresponding to the opening edge of the latch (this CRP also appears in the timing report) and the other corresponding to the closing edge of the device. The selection details section also reports the decision making process behind both CRP values. Note that in this case, since the opening and closing clock edges at the latch will always be different there will always be a mismatch in clock edges for one of them.

EXAMPLES

The following example displays how to use the command to report the clock reconvergence pessimism calculated between the sequential elements, *ffa* and *ffd*, for a setup data path.

```
pt_shell> report_crpr -from ffa/CP -to ffd/CP -setup
```

```
report_crpr -from ffa/CP -to ffd/CP -setup  
*****
```

```

Report : CRP Calculation
Design : counter
Version: U-2003.03-Beta3
Date   : Wed Feb 12 15:44:20 2003
*****

```

```

Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)

```

```

Common Point: CLK <inside>
Common Clock: CLK
Launching edge at common point: RISING
Capturing edge at common point: RISING
CRPR threshold: 0.02

```

Arrival Times	Early	Late	CRP
Rise	3.00	19.00	16.00
Fall	5.00	23.00	18.00

Selection Details

Edge Match:	Match, using rise CRP
clock reconvergence pessimism	16.00

1

The following example displays a report where the launching device is a flip-flop and the capturing device is a latch. Note that in this case the mismatch in clock edges occurs for the CRP corresponding to the closing edge at the latch.

```
pt_shell> report_crpr -from reg1/CP -to lat2/G
```

```

*****
Report : CRP Calculation
Design : borrow
Version: U-2003.03-Beta3
Date   : Wed Feb 12 15:44:21 2003
*****

```

```

Startpoint: reg1 (rising edge-triggered flip-flop clocked by clk)
Endpoint: lat2 (positive level-sensitive latch clocked by clk)

```

```

Common Point: clk_buf/Z
Common Clock: clk
Launching edge at common point: RISING
Latch open edge at common point: RISING
CRPR threshold: 0.01

```

Arrival Times	Early	Late	CRP
Rise	2.1229	2.6529	0.5300

Fall	2.0868	2.7868	0.7000

Selection Details

Edge Match (opening): Match, using rise CRP
Edge Match (closing): Mismatch, using min(rise CRP, fall CRP)

clock reconvergence pessimism (open edge)	0.5300
clock reconvergence pessimism (close edge)	0.5300

1

The following example displays the case when both SI and CRPR are enabled. As outlined in a previous paragraph, two sets of arrival totals are computed in this case (for example, when both CRPR and SI are enabled). One set of arrival totals are computed with no SI information and another are computed considering SI information. To illustrate this, both a setup and hold report are displayed to highlight the case when CRP is calculated from arrival times without delta delays (setup) and the case when CRP is calculated from arrival times including delta delays (hold with the launching and capturing clock edge occurring at the same time).

pt_shell> **report_crpr -from seg1/u3/CK -to seg1/u9/CK**

```
*****
Report : CRP Calculation
Design : top
Version: U-2003.03-Beta3
Date   : Wed Feb 12 06:49:51 2003
*****
```

Startpoint: seg1/u3 (rising edge-triggered flip-flop clocked by CLK1)
Endpoint: seg1/u9 (rising edge-triggered flip-flop clocked by CLK1)

Common Point: seg1/u17_c/Y
Common Clock: CLK1
Launching edge at common point: RISING
Capturing edge at common point: RISING
CRPR threshold: 0.02

Arrival Times (Static)	Early	Late	CRP
Rise	0.3633	0.3746	0.0113
Fall	0.3871	0.3991	0.0120

Arrival Times (Dynamic)	Early	Late	CRP
Rise	0.2142	0.3746	0.1604
Fall	0.2296	0.3991	0.1695

Selection Details

Arrival Times: Static
Edge Match: Match, using rise CRP

clock reconvergence pessimism 0.0113

Range of accuracy of CRP in report_timing, due to value
of timing_crpr_threshold_ps: 0.0000 <= CRP <= 0.0113

1

pt_shell> **report_crpr -from seg1/u3/CK -to seg1/u9/CK -hold**

Report : CRP Calculation

Design : top

Version: U-2003.03-Beta3

Date : Wed Feb 12 06:49:51 2003

Startpoint: seg1/u3 (rising edge-triggered flip-flop clocked by CLK1)

Endpoint: seg1/u9 (rising edge-triggered flip-flop clocked by CLK1)

Common Point: seg1/u17_c/Y

Common Clock: CLK1

Launching edge at common point: RISING

Capturing edge at common point: RISING

CRPR threshold: 0.02

Arrival Times (Static)	Early	Late	CRP
Rise	0.3633	0.3746	0.0113
Fall	0.3871	0.3991	0.0120

Arrival Times (Dynamic)	Early	Late	CRP
Rise	0.2142	0.3746	0.1604
Fall	0.2296	0.3991	0.1695

Selection Details

Arrival Times: Static
Edge Match: Match, using rise CRP

clock reconvergence pessimism 0.1604

1

SEE ALSO

set_clock_latency (2)
set_rail_voltage (2)
timing_clock_reconvergence_pessimism (3)
timing_remove_clock_reconvergence_pessimism (3)
timing_crpr_threshold (3)
si_enable_analysis (3)

report_delay_calculation

Displays the actual calculation of a cell or net timing arc delay value.

SYNTAX

```
int report_delay_calculation [-min | -max]
    [-from_rise_transition value]
    [-from_fall_transition value]
    -from from_pin -to to_pin | -of_objects objects
    [-nosplit]
    [-thresholds]
    [-crosstalk]

string      from_pin
string      to_pin
float       value
collection objects
```

ARGUMENTS

-min

Indicates that minimum delay calculation is to be shown. The design must be in min/max mode.

-max

Indicates that maximum delay calculation is to be shown. This is the default if neither the **-min** nor **-max** option is specified.

-from_rise_transition value

Specifies a value to be used by the delay calculation for the from rise transition.

-from_fall_transition value

Specifies a value to be used by the delay calculation for the from fall transition.

-from *from_pin* -to *to_pin*

Specifies the start and end points of a timing arc within a design. For a cell timing arc, the pins must represent the input and output pins of a common leaf cell, which have a timing arc specified between them in the library. For a net timing arc, the pins must be a driver and a load on a common net. Port names are allowed in place of pin names for net arcs. You must use either the **-from/-to** combination or the **-of_objects** argument, but not both.

-of_objects *objects*

Specifies a collection of timing arcs (created with the **get_timing_arcs** command) on which to report. Arcs in the list are reported in order of from and to pins. You must use either the **-from/-to** combination or the **-of_objects** argument, but not both.

-nosplit

Prevents line-splitting and facilitates writing software to extract

information from the report output. Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

-thresholds

Reports the characterization thresholds that are used for delay calculation.

-crosstalk

Reports the crosstalk information for a net arc. The arc is specified by **-from_pin** and **-to_pin**. It is not permitted with **-of_objects** and user chosen transition time **-from_rise_transition** and **-from_fall_transition**. The crosstalk information is reported from the last **update_timing**.

DESCRIPTION

Provides detailed timing calculation information about the specified cell or net timing arc. This information is useful for debugging or verifying timing data in a technology library.

Both operating conditions and wire load models are considered when making delay calculations. Timing ranges as affected by the command **set_timing_derate** (similar to the dc_shell command **set_timing_range**) are not considered because they typically apply to an entire path.

The information reported is solely for the input arcs. Even if there are other arcs merging at the to-pin of an input arc, the transition time reported for the to-pin is the transition time that the input arc contributes to the merging point. As a result, the reported transition time may not match the transition time at the to-pin because the final transition time is the result of merging transition times contributed by all the input arcs. To get the final transition time at the to-pin, either all these input arcs should be reported or an attribute access to the to-pin should be used.

The **-crosstalk** option on net arc, reports the aggressor and victim information for both rise and fall. It prints coupling capacitance, the driving library cell, the clocks reaching the net (both victim and aggressor). The aggressors have "Switching Bump", which is used for prioritizing the aggressors, and also have aggressor attributes. These informations are already available as net or design attributes. For example, "Switching Bump" is available as net attribute **si_xtalk_bumps**. Considering the memory usage, only the worst case bumps for the net is stored, even though during the crosstalk delay calculation switching bump per net arc is used. "Switching Bump" is crosstalk delay bump ratio of VDD. The clocks gives insight into the constraints of the coupling cluster. For example, during the normal mode analysis the test clocks are supposed to be quiet. If there is no arrival windows, it reported as "No Arrival". Driving cell gives a idea relative strength of the victim and aggressor drivers. If the ports/pins have no driving cell it is reported as "No Lib Cell".

It also prints the other settings in design that could effect the crosstalk delay calculation, for example, logical correlation, composite aggressor setting, crosstalk delay analysis mode, and crosstalk delay effort level.

Attributes:

- A - aggressor is Active
- C - aggressor is a composite aggressor
- E - aggressor is screened due to user Exclusion
- I - aggressor has Infinite arrival with respect to the victim
- L - aggressor is screened due to Logical correlation
- N - aggressor does Not overlap for the worst case alignment
- S - aggressor is screened for Small bumps
- U - aggressor/victim RC calculation is skipped.
- X - aggressor is screened due to aggressor eXclusion

Active aggressor is an aggressor that contributes to the worst case alignment scenario. It is reported as 'A'.

In version X-2005.06 onwards, a new feature is added that analyzes the composite effect of a group of weak aggressors (including filtered ones) for better quality of SI analysis. These aggressors are reported as 'C'.

You can exclude an aggressor from the analysis by using the **case_analysis**, **set_si_delay_analysis -exclude** or **set_coupling_separation** commands. These aggressors are reported with attribute 'E'.

An aggressor has infinite window when it has no fixed timing relationship with the victim, for example, an aggressor clock is asynchronous to the victim clock. Also, if the victim or aggressor has **set_si_delay_analysis -ignore_arrival**, the aggressor gets the infinite window attribute 'I'.

When PrimeTime SI finds that the victim and aggressors cannot switch together due to the logical relationship (for example, the input and output net of a inverter, they cannot switch in the same direction at the same time), PrimeTime SI chooses the combination of aggressors that can switch together logically and set them to active. The aggressors that cannot switch due to logical relationship are screened with attribute 'L'. The logical relationship could be identified only for buffer/inverter chains.

Due to multiple aggressors and their possible multiple arrival windows there could be multiple alignments which could cause crosstalk delta. PrimeTime SI identifies and reports the crosstalk delta for only the worst case alignment. If the aggressor does not contribute to the worst case alignment scenario, it is screened, and the attribute is set to 'N'.

The switching bumps that are smaller compared to the electrical filtering criteria, are screened. The attribute of these aggressors are set to 'S'. When composite aggressor feature is applied, the SI effect of screened aggressors are considered, and 'S' is replaced by 'C'.

If the aggressor or victim RC calculation is skipped, the attribute is set to 'U'. The common cause of the RC calculation skipping on a net is there is no cell arc driving the net(victim or aggressor) or the driving model could not be created.

The **set_si_aggressor_exclusion** command can set aggressor exclusion relationship among aggressors (for example, the specified aggressors cannot switch in the same direction at the same time), PrimeTime SI chooses the combination of aggressors that can switch together based on alignment and sets them to active. The aggressors that cannot switch due to aggressor exclusion relationship are screened with attribute 'X'.

EXAMPLES

The following example shows the output on a cell arc connected to an RC network. RC delay calculation is performed.

```
pt_shell> report_delay_calculation -thresholds -from U3/CK -to U3/Q
```

```
*****
Report : delay_calculation
Design : test
Version: W-2004.12
Date   : Thu Nov  4 10:54:44 2004
*****
```

```
From pin: U3/CK
To pin:   U3/Q
Main Library Units: 1ns 0.001pF 1000kOhm
```

```
Library: 'slow'
Library Units: 1ns 1pF 1kOhm
Library Cell: 'SDFFX2'
arc sense: rising_edge
arc type: cell
```

Calculation	Rise	Rise	Fall	Fall	Slew	Rail	
Thresholds:	Delay	Slew	Delay	Slew	Derate	Voltage	Temp.
from-pin	50	10->90	50	90->10	1.000	1.080	-40.0
to-pin	50	10->90	50	90->10	1.000	1.080	-40.0

RC network on pin 'U3/Q' :

```
-----
Number of elements = 2 Capacitances + 1 Resistances
Total capacitance = 0.094381 pF
Total capacitance = 94.380999 (in library unit)
Total resistance  = 5.000000 Kohm
```

	Rise	Fall	
Input transition time	= 0.000000	0.000000	(in library unit)
Effective capacitance	= 0.069497	0.061939	(in pF)
Effective capacitance	= 69.496963	61.939442	(in library unit)
Drive resistance	= 2.020383	1.252620	(in Kohm)
Output transition time	= 0.656019	0.389840	(in library unit)
Cell delay	= 0.517020	0.356812	(in library unit)

The following example shows the output on a net arc. Wire load delay calculation is performed.

```
pt_shell> report_delay_calculation -min -thresholds -from U3/Q -to U4/A
```

```
*****
Report : delay_calculation
      -min
Design : test
Version: W-2004.12
Date   : Thu Nov  4 10:54:44 2004
*****
```

```
From pin: U3/Q
To pin:   U4/A
Main Library Units: 1ns 0.001pF 1000kOhm
```

```
arc sense: unate
arc type:  net
```

Calculation	Rise	Rise	Fall	Fall	Slew	Rail	
Thresholds:	Delay	Slew	Delay	Slew	Derate	Voltage	Temp.
from-pin	50	10->90	50	90->10	1.000	1.080	-40.0
to-pin	50	10->90	50	90->10	1.000	1.080	-40.0

RC delay calculation is being skipped because RC delay calculation was not used for the driver arcs.

```
Balanced case tree
RC delay: (r_wire/load_count) * (c_pin + c_wire/load_count)
rise:     (0.005 / 1) * (4.381 + (90 / 1))
fall:     (0.005 / 1) * (4.381 + (90 / 1))
```

```
Rise delay = 0.471905
Fall delay = 0.471905
```

The following example shows the output on a cell arc. Table lookup into cell the library tables is performed.

```
pt_shell> report_delay_calculation -thresholds -from U1/A -to U1/Y
```

```
*****
Report : delay_calculation
Design : test
Version: W-2004.12
Date   : Thu Nov  4 10:54:44 2004
*****
```

```
From pin: U1/A
To pin:   U1/Y
Main Library Units: 1ns 0.001pF 1000kOhm
```

```

Library: 'slow'
Library Units: 1ns 1pF 1kOhm
Library Cell: 'INVX2'
arc sense: negative_unate
arc type: cell

```

Calculation Thresholds:	Rise Delay	Rise Slew	Fall Delay	Fall Slew	Slew Derate	Rail Voltage	Temp.
from-pin	50	10->90	50	90->10	1.000	1.080	-40.0
to-pin	50	10->90	50	90->10	1.000	1.080	-40.0

RC delay calculation is being skipped because the driver from-pin is disabled.

Units: 1ns 1pF 1kOhm

Rise Delay

```

cell delay = 0.334919
Table is indexed by
  (X) input_pin_transition = 0
  (Y) output_net_total_cap = 0.094381
Relevant portion of lookup table:
      (X) 0.0420      (X) 0.0660
(Y) 0.0844      (Z) 0.3141      (Z) 0.3215
(Y) 0.1706      (Z) 0.6050      (Z) 0.6125

```

```

Z = A + B*X + C*Y + D*X*Y
A = 0.0168      B = 0.3052
C = 3.3703      D = 0.0362

```

```

Z = 0.334919
scaling result for operating conditions
multiplying by 1 gives 0.334919

```

Fall Delay

```

cell delay = 0.215786
Table is indexed by
  (X) input_pin_transition = 0
  (Y) output_net_total_cap = 0.094381
Relevant portion of lookup table:
      (X) 0.0420      (X) 0.0660
(Y) 0.0844      (Z) 0.2071      (Z) 0.2145
(Y) 0.1706      (Z) 0.3939      (Z) 0.4013

```

```

Z = A + B*X + C*Y + D*X*Y
A = 0.0115      B = 0.3079
C = 2.1650      D = 0.0082

```

```

Z = 0.215786
scaling result for operating conditions
multiplying by 1 gives 0.215786

```


Cell Delay
rise: 0.334919
fall: 0.215786

Rise delay = 0.334919
Fall delay = 0.215786

Units: 1ns 1pF 1kOhm

Transition rise
transition = 0.610174
Table is indexed by
(X) input_pin_transition = 0
(Y) output_net_total_cap = 0.094381
Relevant portion of lookup table:

	(X)	0.0420	(X)	0.0660	
(Y)	0.0844	(Z)	0.5478	(Z)	0.5478
(Y)	0.1706	(Z)	1.0854	(Z)	1.0854

$Z = A + B \cdot X + C \cdot Y + D \cdot X \cdot Y$
A = 0.0220 B = 0.0005
C = 6.2318 D = -0.0058

Z = 0.610174
scaling result for operating conditions
multiplying by 1 gives 0.610174

Transition fall
transition = 0.367266
Table is indexed by
(X) input_pin_transition = 0
(Y) output_net_total_cap = 0.094381
Relevant portion of lookup table:

	(X)	0.0420	(X)	0.0660	
(Y)	0.0844	(Z)	0.3297	(Z)	0.3297
(Y)	0.1706	(Z)	0.6536	(Z)	0.6536

$Z = A + B \cdot X + C \cdot Y + D \cdot X \cdot Y$
A = 0.0129 B = -0.0003
C = 3.7542 D = 0.0029

Z = 0.367266
scaling result for operating conditions
multiplying by 1 gives 0.367266

Rise transition = 0.610174
Fall transition = 0.367266

The following example shows the output on a net arc whose from-pin is a hierarchical pin. The delay of the net is assigned to the net arc whose from-pin is a leaf pin.

```
pt_shell> report_delay_calculation -from U1/clock_out -to U102/CP
```

```
*****
Report : delay_calculation
Design : test
Version: W-2004.12
Date   : Thu Nov  4 12:51:33 2004
*****
```

RC delay calculation is being skipped because the load to-pin is not an input pin.

```
From pin: U1/clock_out
To pin:   U102/CP
Main Library Units: 1ns 1pF 1kOhm
```

```
arc sense: unate
arc type:  net
```

```
Net arc from a hierarchical pin.
Arc rise delay = 0 (assigned)
Arc fall delay = 0 (assigned)
To_pin rise transition time = 49.6413 (copied from from_pin)
To_pin fall transition time = 35.2787 (copied from from_pin)
```

The following example shows the report for cell arc when the advanced mode of delay calculation is used. Advanced mode for driver model can be set using the **rc_driver_model_mode** variable. No additional option of **report_delay_calculation** is necessary to report details of the advanced mode of delay calculation.

```
pt_shell> report_delay_calculation -min -thresh -from sig_in0 -to sig_in0 -nosplit
```

```
*****
Report : delay_calculation
        -min
Design : eval_c
Version: Y-2006.06
Date   : Tue Mar 28 10:33:16 2006
*****
```

```
From port: sig_in0
To port:   sig_in0
Main Library Units: 1ns 1pF 1kOhm
```

```
Library: 'ports'
Library Units: 1ns 1pF 1kOhm
Library Cell: 'sig_in0'
arc sense: positive_unate
arc type:  cell
```

Calculation	Rise	Rise	Fall	Fall	Slew	Rail	
Thresholds:	Delay	Slew	Delay	Slew	Derate	Voltage	Temp.

```
report_delay_calculation
650
```

from-pin	50	30->70	50	70->30	0.400	1.100	125.0
to-pin	50	30->70	50	70->30	0.400	1.100	125.0

RC network on pin 'sig_in0' :

Number of elements = 8 Capacitances + 7 Resistances
Total capacitance = 0.003062 pF
Total capacitance = 0.003062 (in library unit)
Total resistance = 0.017983 Kohm

Advanced driver-modeling used for rise and fall.

	Rise	Fall	
Input transition time	= 0.100000	0.100000	(in library unit)
Effective capacitance	= 0.002625	0.002800	(in pF)
Effective capacitance	= 0.002625	0.002800	(in library unit)
Output transition time	= 0.060388	0.047040	(in library unit)
Cell delay	= 0.050937	0.045125	(in library unit)

The following example shows the report for net arc when the advanced mode of delay calculation is used. You can set the advanced mode for the receiver model by using the **rc_receiver_model_mode** variable. No additional option of **report_delay_calculation** is necessary to report details of the advanced mode of delay calculation. In addition to the existing information, the pin capacitances of the advanced receiver model and the transition values used to calculate the pin capacitances of the advanced receiver model are also reported.

pt_shell> **report_delay_calculation -max -thresh -from sig_in0 -to cell0/A -nosplit**

From port: sig_in0
To pin: cell0/A
Main Library Units: 1ns 1pF 1kOhm

arc sense: unate
arc type: net

Calculation	Rise	Rise	Fall	Fall	Slew	Rail	
Thresholds:	Delay	Slew	Delay	Slew	Derate	Voltage	Temp.
from-pin	50	30->70	50	70->30	0.400	1.100	125.0
to-pin	50	30->70	50	70->30	0.400	1.100	125.0

RC network on pin 'sig_in0' :

Number of elements = 8 Capacitances + 7 Resistances
Total capacitance = 0.003062 pF
Total capacitance = 0.003062 (in library unit)
Total resistance = 0.017983 Kohm

Advanced receiver-modeling used for rise and fall.

Advanced Receiver Model

	Rise	Fall	
Receiver model capacitance 1	= 0.001966	0.001888	(in library unit)
Receiver model capacitance 2	= 0.002328	0.002160	(in library unit)
Receiver model transition 1	= 0.059192	0.037666	(in library unit)
Receiver model transition 2	= 0.059192	0.037666	(in library unit)

	Rise	Fall	
Net delay	= 0.000024	0.000064	(in library unit)
Transition time	= 0.059192	0.037599	(in library unit)
From_pin transition time	= 0.059078	0.037666	(in library unit)
To_pin transition time	= 0.059192	0.037599	(in library unit)
Net slew degradation	= 0.000114	-0.000067	(in library unit)

The **define_scaling_lib_group** command allows you to define a group of libraries that PrimeTime interpolates between for voltage and/or temperature scaling. The following example shows the output on a net arc when scaling of the receiver model is done. The report indicates that scaling was done and also reports all the scaling libraries that were used for scaling the model.

```
pt_shell > report_delay_calculation -from I1/Z -to I2/B
```

```
From pin: I1/Z
To pin:   I2/B
Main Library Units: 1ns 0.001pF 1000kOhm
```

```
arc sense: unate
arc type:  net
```

```
RC network on pin 'I1/Z' :
```

```
-----
Number of elements = 2 Capacitances + 1 Resistances
Total capacitance  = 0.815687 pF
Total capacitance  = 815.686687 (in library unit)
Total resistance   = 8.323139 Kohm
```

```
Scaling library pin group used for rise and fall.
Scaling libraries used for receiver model : tc300c_0.85 tc300c_1.05
```

	Rise	Fall	
Net delay	= 2.406542	2.836394	(in library unit)
Transition time	= 5.312853	4.983530	(in library unit)
From_pin transition time	= 0.804747	0.328441	(in library unit)
To_pin transition time	= 5.312853	4.983530	(in library unit)
Net slew degradation	= 4.508106	4.655089	(in library unit)

The following run shows the output on a net arc connected to a port. RC delay calculation with crosstalk is performed.

pt_shell> **report_delay_calculation -crosstalk -from u2/Z -to o2**

Report : delay_calculation
Design : xtalk_test

From pin: u2/Z
To port: o2
Main Library Units: 1ns 1pF 1kOhm

arc sense: unate
arc type: net
Annotated max rise net delta delay: 0 arc delay: 0.0151697
Annotated max fall net delta delay: 0 arc delay: 0.0153037

RC network on pin 'u2/Z' :

Number of elements = 5 Capacitances + 4 Resistances
Total capacitance = 0.300000 pF
Total capacitance = 0.300000 (in library unit)
Total resistance = 0.100000 Kohm

	Rise	Fall	
Net delay	= 0.015170	0.015304	(in library unit)
Transition time	= 0.940437	0.542964	(in library unit)
From_pin transition time	= 0.940063	0.542372	(in library unit)
To_pin transition time	= 0.940437	0.542964	(in library unit)
Net slew degradation	= 0.000374	0.000592	(in library unit)

Annotated max rise delta transition: 0 pin transition: 0.940437
Annotated max fall delta transition: 0 pin transition: 0.542964

Reporting for Crosstalk:

Victim net name:	o2
Number of aggressors:	1
Number of effective (non-filtered) aggressors:	1
Net is reselected:	true
Victim driver rail voltage(VDD):	2.700000
si_xtalk_analysis_effort_level:	medium
si_xtalk_delay_analysis_mode:	all_paths
si_analysis_logical_correlation_mode:	true
Crosstalk composite aggressor mode:	disabled

Attributes:

- A - aggressor is Active
- C - aggressor is a composite aggressor
- E - aggressor is screened due to user Exclusion
- I - aggressor has Infinite arrival with respect to the victim
- L - aggressor is screened due to Logical correlation
- N - aggressor does Not overlap for the worst case alignment
- S - aggressor is screened for Small bumps
- U - aggressor/victim RC calculation is skipped
- X - aggressor is screened due to aggressor eXclusion

Victim is rising:					
	Victim	Coupling	Driver	Clocks	
	Net	Cap	Lib Cell		
	-----	-----	-----	-----	
	o2	0.200000	BF1T4_D	CLK2	
Aggressor					
tes	Switching Bump	Coupling	Driver	Clocks	Attribu
	Net	Cap	Lib Cell		
	(ratio of VDD)				
	-----	-----	-----	-----	-----
---	-----	-----	-----	-----	-----
	o1	0.200000	BF1T4_D	CLK1	N
	-				
Victim is falling:					
	Victim	Coupling	Driver	Clocks	
	Net	Cap	Lib Cell		
	-----	-----	-----	-----	
	o2	0.200000	BF1T4_D	CLK2	
Aggressor					
tes	Switching Bump	Coupling	Driver	Clocks	Attribu
	Net	Cap	Lib Cell		
	(ratio of VDD)				
	-----	-----	-----	-----	-----
---	-----	-----	-----	-----	-----
	o1	0.200000	BF1T4_D	CLK1	N
	-				

SEE ALSO

get_timing_arcs (2)
report_lib (2)
report_timing (2)
si_xtalk_analysis_effort_level (3)
si_xtalk_delay_analysis_mode (3)
si_analysis_logical_correlation_mode (3)
si_xtalk_composite_aggr_mode (3)
rc_driver_model_mode (3)
rc_receiver_model_mode (3)
define_scaling_lib_group (2)
report_lib_groups (2)
set_si_aggressor_exclusion (2)

report_design

Displays attributes on the **current_design**.

SYNTAX

```
int report_design [-nosplit]
```

ARGUMENTS

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

Lists information about the attributes on the **current_design**.

EXAMPLES

The following is an example of a design report.

```
pt_shell> report_design
```

```
*****
Report : design
Design : counter
Version: 1997.01-development
Date   : Fri Aug  2 17:12:27 1996
*****
```

Design Attribute	Value

Operating Conditions:	
operating_condition_max_name	WCCOM
process_max	1.50
temperature_max	70.00
voltage_max	4.75
tree_type_max	worst_case_tree
Wire Load:	
wire_load_mode	(use report_wire_load for more information)
wire_load_model_max	top
wire_load_selection_group_max	--
wire_load_min_block_size	--
max_capacitance	0
Design Rules:	
max_capacitance	0.5

min_capacitance	--
max_fanout	5.6
min_fanout	--
max_transition	0.8
min_transition	--
max_area	--
Timing Ranges:	
fastest_factor	--
slowest_factor	--

SEE ALSO

report_clock (2), **report_port** (2).

report_disable_timing

Reports disabled timing arcs in the current design.

SYNTAX

```
string report_disable_timing  
[-nosplit]  
[cells_or_ports]  
collection cells_or_ports
```

ARGUMENTS

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line starting in the correct column.

cells_or_ports

Limits disabled arc reporting to the specified list of cells or ports. Provide the list or collection of cells or ports as an argument to the command.

DESCRIPTION

Reports disabled timing arcs in the current design. Timing arcs can be disabled in several ways. You can disable a timing arc by using the **set_disable_timing** command. The following symbols are used to explain why a timing arc is disabled:

c : The propagation of case-analysis values
C : The presence of conditional arcs (arcs that have a when statement defined in the library)
d : The presence of default arcs (when the **timing_disable_cond_default_arcs** variable is set to true)
f : Disabling of false net-arcs
l : Loop breaking
L : db inherited loop breaking
p : The propagation of constant values
u : Arcs disabled by using the **set_disable_timing** command.

EXAMPLES

The following example shows that the timing arc of a cell named *U1/U2* from pin *A* to pin *Z* is disabled.

```
pt_shell> set_disable_timing {n2_i} -from A -to Z  
pt_shell> report_disable_timing  
*****  
Report : disable_timing  
Design : middle
```

Flags : c case-analysis
 C Conditional arc
 d default conditional arc
 f false net-arc
 l loop breaking
 L db inherited loop breaking
 p propagated constant
 u user-defined

Cell or Port	From	To	Sense	Flag	Reason
n2_i	A	Z	negative_unate	u	

1

The following example specifies that the timing arc of cell *ff4* from pin *CP* to pin *TE* and *TI* are disabled as a result of a case-analysis constant of 0 propagated to pin *TE* of cell *ff4*. Note that the actual case value is set on another pin *A* and the propagation path to *ff4/TE* is inverting.

```
pt_shell> set_case_analysis 0 {p_in in0}
pt_shell> report_disable_timing
```

Report : disable_timing
Design : middle

Flags : c case-analysis
 C Conditional arc
 d default conditional arc
 f false net-arc
 l loop breaking
 L db inherited loop breaking
 p propagated constant
 u user-defined

Cell or Port	From	To	Sense	Flag	Reason
o_reg4	CP	D	hold_clk_rise	c	D = 0
o_reg4	CP	D	setup_clk_rise	c	D = 0
o_reg4	CP	CD	recovery_rise_clk_rise		
				c	D = 0
p_out				c	p_out = 1

1

To view disabled arcs in specific cells, provide a list or collection of the required cells as an argument to the command. In the above example, to view disabled arcs for *o_reg4* instance, do the following:

```
pt_shell> report_disable_timing [get_cells { o_reg4 }]
```

Report : disable_timing
Design : middle

Flags : c case-analysis
 C Conditional arc
 d default conditional arc
 f false net-arc
 l loop breaking
 L db inherited loop breaking
 p propagated constant
 u user-defined

Cell or Port	From	To	Sense	Flag	Reason

o_reg4	CP	D	hold_clk_rise	c	D = 0
o_reg4	CP	D	setup_clk_rise	c	D = 0
o_reg4	CP	CD	recovery_rise_clk_rise		
				c	D = 0

1

SEE ALSO

remove_disable_timing (2), set_disable_timing (2).

report_distributed_hosts

Creates a detailed report on user-defined distributed hosts in the farm's pool

SYNTAX

Boolean **report_distributed_hosts**

DESCRIPTION

The **report_distributed_hosts** generates a report on user-defined distributed hosts in the farm's pool. The following information is displayed for each distributed processor.

Farm indicates the type of the farm from which the distributed processor was acquired, one of 'lsf', 'grd', 'generic' or 'now'.

Status indicates whether the distributed processor was launched by issuing 'successfully launched' on success and 'Not yet launched' when the distributed processor failed to launch or has not yet been launched.

Host Name indicates the name of the machine, grid name, or lsf name from which the distributed processor was acquired.

Command is the command used to launch the distributed processor.

EXAMPLES

In the following example a distributed processor is allocated from a workstation called platinum1. The processor is launched using the **create_distributed_farm** command. The **report_distributed_hosts** command then generates a report on the processor.

```
pt_shell> add_distributed_hosts -farm now -32bit platinum1
1
```

```
pt_shell> create_distributed_farm
```

```
1] Launched: rsh -n -l user platinum1 pt_shell -multi_scenario -32bit
-__zdpp_slave ptsrv2 16874 16859 1 /user/work PrimeTime 4106 4108 53205
      Status: Forking successful
      Stdout: your job
      Stderr: **<<EMPTY>>**
```

```
-----
Waiting for  1 (of  1) distributed hosts
-----
```

```
pt_shell> report_distributed_hosts
```

```
1] Farm      : now
   Status    : Successfully launched
   Host Name: platinum1
   Command   : pt_shell -multi_scenario -32bit
```

SEE ALSO

create_distributed_farm (2)

add_distributed_hosts (2)

report_driver_model

Displays the driver model for a library cell timing arc used to drive annotated parasitics.

SYNTAX

```
int report_driver_model  
-lib_cell lib_cell  
-from_pin from_pin  
-to_pin to_pin  
-rise_slew rise_slew  
-fall_slew fall_slew  
-capacitance capacitance  
  
string lib_cell  
string from_pin  
string to_pin  
float rise_slew  
float fall_slew  
double capacitance
```

ARGUMENTS

```
-lib_cell lib_cell  
    Specifies the name of the library cell for which the driver model is to be  
    computed. The name should be in the form library_name/cell_name.  
  
-from_pin from_pin  
    Specifies the start point of a timing arc through the lib_cell.  
  
-to_pin to_pin  
    Specifies the endpoint of a timing arc through the lib_cell.  
  
-rise_slew rise_slew  
    Specifies the rise time in library units on the from_pin.  
  
-fall_slew fall_slew  
    Specifies the fall time in library units on the from_pin.  
  
-capacitance capacitance  
    Specifies the load in library units on the to_pin.
```

DESCRIPTION

The **report_driver_model** command provides both the library data and the resulting driver model parameters for the specified library arc and conditions. This information is useful for validating library data and/or PrimeTime driver models used in delay calculation with annotated parasitics. For each timing arc between the specified pins, the delay, slew, and sensitivities (the changes in delay and slew resulting from a small change in load capacitance) are displayed. This information can be compared to simulation results using the same input slew and output load to

measure library interpolation errors. If the current design is in min-max mode, data for both operating conditions is displayed.

Slews are displayed without derating (They are shown as the time between the trip-points.). Use of the shell variable **rc_slew_derate_from_library** occurs only internally for this report when interacting with the library.

In addition to the library data, the driver model parameters used in delay calculation with annotated parasitics are displayed. Currently, PrimeTime supports only the simplified driver model (SDM) with three parameters: rd (the drive resistance through which a linear voltage ramp is applied), tz (the ramp start time relative to the arc input threshold time), and dt (the full-swing ramp duration).

The driver model is constructed to match the library delay, slew, and sensitivity for the specified input slew and output load. The cumulative relative error percentage in matching these values is displayed as an error. The check results (pass or FAIL) indicate whether or not this value is less than the value of the shell variable **rc_driver_model_max_error_pct**.

Problematic library data can prevent the construction of a driver model altogether. In this case, the driver model parameters will all be displayed as zero, and the check will show FAIL. The most common reason for this is due to the library data not indicating an increasing delay and/or slew for increasing output capacitance; this indicates a non-positive drive resistance. Another common reason is incorrect trip-point settings.

EXAMPLES

```
pt_shell> report_driver_model \  
? -lib_cell [get_lib_cell -of I0] \  
? -from A -to Y \  
? -rise 0.0384 -fall 0.0384 -cap 0.115194
```

```
*****  
Report : driver_model  
Driver : core/inv27:A-->Y  
Version: 2000.11  
Date   : Thu Aug  7 14:43:08 2000  
*****
```

Accuracy Settings:	Rise	Fall
slew_lower_threshold	= 30%	30%
slew_upper_threshold	= 70%	70%
input_threshold	= 50%	50%
output_threshold	= 50%	50%
slew_derate_from_library	=	1
driver_model_max_error	=	5%

```
Library Inputs:  (in library units)  
Rising input slew      = 0.038400  
Falling input slew     = 0.038400
```

Output load capacitance = 0.115194

Driver model for sense 'negative-unate':

(max)	Rise	Fall	
load	= 0.115194	0.115194	(pF)
delay	= 0.071811	0.044181	(ns)
slew	= 0.043008	0.026270	(ns without derate)
d-load	= 0.001152	0.001152	(pF)
d-delay	= 0.000392	0.000264	(ns)
d-slew	= 0.000332	0.000160	(ns without derate)
rd	= 0.406621	0.249222	(kohm)
tz	= -0.000323	0.000139	(ns)
dt	= 0.070798	0.043157	(ns)
error	= 0.853807	0.950170	(%)
check	= pass	pass	

SEE ALSO

`rc_driver_model_max_error_pct` (3), `rc_input_threshold_pct_fall` (3),
`rc_input_threshold_pct_rise` (3), `rc_output_threshold_pct_fall` (3),
`rc_output_threshold_pct_rise` (3), `rc_slew_derate_from_library` (3),
`rc_slew_lower_threshold_pct_fall` (3), `rc_slew_lower_threshold_pct_rise` (3),
`rc_slew_upper_threshold_pct_fall` (3), `rc_slew_upper_threshold_pct_rise` (3).

report_etm_arc

Reports the data and clock paths traversed while extracting a particular timing arc.

SYNTAX

```
string report_etm_arc  
[-from from_object]  
[-rise_from rise_from_object]  
[-fall_from fall_from_object]  
[-to to_object]  
[-rise_to rise_to_object]  
[-fall_to fall_to_object]  
[-arc_type arc_type]  
[-include path_type_list]  
[-context_borrow] | [-latch_level levels]  
[-library_cell]  
[-etm_report er_file_name]  
[-netlist_report nr_file_name]  
[-significant_digits digits]
```

```
list from_object  
list rise_from_object  
list fall_from_object  
list to_object  
list rise_to_object  
list fall_to_object  
list arc_type  
list path_type_list  
int levels  
string er_file_name  
string nr_file_name  
int digits
```

ARGUMENTS

-from *from_object*
Specifies a port or a clock from which the arc of interest originates. The sense at the starting point can be either rising or falling. Substitute one of the following valid values for *from_object*: **-fall_from**, **-from**, or **-rise_from**.

-rise_from *rise_from_object*
Specifies a port or a clock from which the arc of interest originates. The sense at the startpoint must be rising.

-fall_from *fall_from_object*
Specifies a port or a clock from which the arc of interest originates. The sense at the startpoint must be falling.

-to *to_object*
Specifies a port or a clock at which the arc of interest terminates. The sense at the end point can be either rising or falling. Substitute one of the

following valid values for *to_object*: **-fall_to.**, **-rise_to**, or **-to**.

-rise_to *rise_to_object*

Specifies a port or a clock at which the arc of interest terminates. The sense at the endpoint must be rising.

-fall_to *fall_to_object*

Specifies a port or a clock at which the arc of interest terminates. The sense at the endpoint must be falling.

-arc_type *arc_type*

Specifies the type of arc to be reported between the start and endpoint. Currently, *report_etm_arc* can only accept one *arc_type* value. Substitute one of the following valid values for *arc_type*: **clock_gating_hold**, **clock_gating_setup**, **hold**, **max_combo_delay**, **max_seq_delay**, **min_seq_delay**, **min_combo_delay**, **recovery**, **removal**, or **setup**.

-include *path_type_list*

Specifies whether the clock path and/or netlist path should also be reported. Substitute one or more of the following valid values for *path_type_list*: **clock_path** and **netlist_path**. Specifying **clock_path** prints a complete clock path in the generated report. Specifying **netlist_path** prints in the report a corresponding netlist path that was used by model validation. Specifying both **clock_path** and **netlist_path** prints a complete ETM Report (consisting of data and clock path) followed by a complete Netlist Report (consisting of data and clock path).

-context_borrow

Specifies that PrimeTime is to determine the latches on an interface that borrow, based on input port arrival times and clock definitions specified at the time of model extraction. This option is one of three that specify the model extraction environment in which the debugging has to be done: - **context_borrow**, **-latch_level**, and **-library_cell**. The **-context_borrow** and **-latch_level** options are mutually exclusive.

-latch_level *levels*

Specifies the number of levels of latch borrowing that are to occur at the interface of a design. This option is one of three that specify the model extraction environment in which the debugging has to be done: - **context_borrow**, **-latch_level**, and **-library_cell**. The **-context_borrow** and **-latch_level** options are mutually exclusive.

-library_cell

Specifies that the model is to be generated as a library cell. This option eliminates boundary nets, and the boundary parasitics become part of the model. This option is one of three that specify the model extraction environment in which the debugging has to be done: **-context_borrow**, **-latch_level**, and **-library_cell**.

-etm_report *er_file_name*

Specifies that the ETM report must be printed to a file named *er_file_name*.

-netlist_report *nr_file_name*

Specifies that the Netlist report must be printed to a file named *nr_file_name*.

report_etm_arc

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Substitute one of the following valid values for *digits*: an integer from **0** to **13**. The default is **2**. You can use this option to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

DESCRIPTION

Reports the data and clock paths traversed while extracting a particular timing arc. It is used to debug a discrepancy between an ETM and a netlist reported by the **write_interface_timing** and **compare_interface_timing** model validation commands.

By default, it reports only the data path traversed by an ETM while extracting a timing arc. You can use the **-include clock_path** option to report the clock path traversed. You can use the **-include netlist_path** option to compare the path traversed by model extraction to the path traversed by the PrimeTime timing engine. For the purpose of comparing ETM paths to netlist paths, using the **-include netlist_path** option is recommended over using the **report_timing** command.

You must use the same model extraction options and environment variables when generating a model as when debugging the model. This ensures that the same arc is extracted during model extraction and debugging. Hence, the values of environment variables used by the **extract_model** command are also valid for the **report_etm_arc** command. The **-context_borrow**, **-latch_level**, and **-library_cell** options specify how to perform extraction.

EXAMPLES

The following example generates a **report_etm_arc** command from a discrepancy shown by model validation. If the **compare_interface_timing** command yields a failure as:

```
RBUS_RnotW(r) ADC_SCLK_IN(r) setup 17.18 18.24 1.06 FAIL
```

```
pt_shell> report_etm_arc -rise_from RBUS_RnotW \  
-rise_to [get_clock ADC_SCLK_IN] -arc_type setup \  
-include {clock_path netlist_path}
```

The following example generates a **report_etm_arc** command for a clock to output path. From the following model validation discrepancy:

```
ADC_SCLK_IN(r) ADC_LRCLK_OUT(f) FALL 2.04 2.71 0.67 FAIL
```

```
pt_shell> report_etm_arc \  
-rise_from [get_clock ADC_SCLK_IN] \  
-fall_to ADC_LRCLK_OUT -arc_type min_seq_delay \  
-include {clock_path netlist_path}
```

SEE ALSO

`compare_interface_timing` (2), `extract_model` (2), `report_timing` (2),
`write_interface_timing` (2).

report_exceptions

Generates a report of timing exceptions.

SYNTAX

Boolean **report_exceptions**

```
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
[-ignored]
[-nosplit]
```

```
list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
```

ARGUMENTS

Note: Options marked with asterisks (*) in the SYNTAX can be used more than once in the same command.

-from *from_list*

Specifies a list of clocks, ports, cells, and pins in the current design. The report is to include only paths that start at the objects in the *from_list*. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-from** option. The **-from**, **-rise_from**, and **-fall_from** options are mutually exclusive.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions

along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through *through_list*

Specifies a list of pins or ports. The report is to include only paths that go through the pins or ports on *through_list*. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-through** option. You can use this option more than once in the same command.

-rise_through *rise_through_list*

Specifies a list of pins or ports (the same as the **-through** option) except that the paths must have a rising transition at the through points. You can use this option more than once in the same command.

-fall_through *fall_through_list*

Specifies a list of pins or ports (the same as the **-through** option) except that the paths must have a falling transition at the through points. You can use this option more than once in the same command.

-to *to_list*

Specifies a list of clocks, ports, cells, and pins in the current design. The report is to include only paths that end at the objects in the *to_list* objects. Using this option limits the report to information that was set using a path-based command (for example, **set_multicycle_path**) with the **-to** option. The **-to**, **-rise_to**, and **-fall_to** options are mutually exclusive.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-ignored

Indicates that the report is to lists path timing exceptions that are set on the current design, but are completely ignored. For example, a false path might be specified from port A to port Z1, but if there is no timing path between those points, the path is ignored. Use **-from** or **-to** to limit the report to certain paths. Ignored exceptions on objects are also included in the report; for example, an exception placed by **max_time_borrow** on a cell other than a level-sensitive latch.

-nosplit

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new

line, starting in the correct column.

DESCRIPTION

Produces a report showing information about timing exceptions for the current design.

PrimeTime accepts a timing exception if it alters the constraints of the path. If path constraints are different in the absence of a timing exception, the exception alters the path constraint.

The `report_exceptions` command attempts to identify reasons an exception is ignored, fully or partially, and classifies these into one of the following: invalid startpoints, invalid endpoints, non-existent paths, or overridden paths. This is reported as flags in the Ignored column. When `-ignored` is used, the reported flags will indicate why an exception is fully ignored. Otherwise, the reported flags will indicate why a subset of the exception specified paths were ignored.

If a path is unconstrained (that is, if it has a required time of infinity), timing exceptions do not change the path constraints. Applying a timing exception to an unconstrained path does not change the path constraint. A path is unconstrained if it has an infinite required time.

To remove timing exceptions from specified paths, use **`reset_path`**. **`reset_design`** removes all attributes from the design, including timing exceptions.

The number of timing exception objects reported in the *from*, *through*, or *to* sets are limited by the value of the **`collection_result_display_limit`** variable. As in collection result display, setting the variable to a negative value would print the complete set. Any non-negative value would truncate any set whose size is greater than the value; the truncated output is designated by an ellipsis. It is important to note that for negative settings, the behavior matches object collection query echoing the entire object set. It is recommended to avoid negative settings when reporting large numbers of exceptions with sizable object sets as this may adversely affect the performance of the `report_exceptions` command.

EXAMPLES

The following example lists all timing exceptions set on the design.

```
pt_shell> report_exceptions
*****
Report : exceptions
Design : counter
Version: 2002.09
Date   : Tue Mar 19 12:00:00 2002
*****

Reasons:
  f - invalid startpoint(s)
  t - invalid endpoint(s)
  p - non-existent paths
  o - overridden paths
```

From	To	Setup	Hold	Ignored
RESET	*	FALSE	FALSE	
*	CO	max=4.5	min_rise=2,min_fall=2.5	
ffa	ffb	cycles=2	-	

The following example shows rise/fall qualified timing exceptions set on the design.

pt_shell> **report_exceptions**

From	To	Setup	Hold	Ignored
RESET	*	FALSE	FALSE	
*	CO(r)	max=4.5	min_rise=2,min_fall=2.5	
ffa(r)	ffb(f)	cycles=2	-	

The following example lists all ignored timing exceptions set on the design.

pt_shell> **report_exceptions -ignored**

Report : exceptions

-ignored

Design : counter

Version: 2002.09

Date : Tue Mar 19 12:00:00 2002

Reasons:

f - invalid startpoint(s)

t - invalid endpoint(s)

p - non-existent paths

o - overridden paths

From	To	Setup	Hold	Ignored
QA	*	max=10	-	f
*	A	cycles=2	-	t
*	B	FALSE	FALSE	t

Object	Type	Attributes
CLK	clock	max_time_borrow=2.3

The following example lists some user-entered compressed exceptions.

pt_shell> **report_exceptions**

Report : exceptions

Design : counter

Version: 2002.09

Date : Tue Mar 19 12:00:00 2002

Reasons:

report_exceptions

f - invalid startpoint(s)
t - invalid endpoint(s)
p - non-existent paths
o - overridden paths

From	Through	To	Setup	Hold	Ignored

{ ffa/CP ffb/CP }	*	{ ffa/D ffb/D ffc/D }	max=2		p

SEE ALSO

`current_design(2), set_false_path(2), set_max_delay(2), set_max_time_borrow (2), set_min_delay(2), set_multicycle_path(2), collection_result_display_limit(3).`

report_global_slack

Displays slack of specified pins or ports.

SYNTAX

```
int report_global_slack  
[-significant_digits digits]  
[-nosplit]  
[-min] [-max]  
[-rise] [-fall]  
port_pin_list
```

"Data Types

port_pin_list list

ARGUMENTS

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

-min

Displays min slack.

-max

Displays max slack.

-rise

Displays rise slack.

-fall

Displays fall slack.

port_pin_list

Specifies a list of pins or ports to report. By default, the report contains all pins and ports in the current design. Pins of hierarchical cells are not reported.

DESCRIPTION

The **report_global_slack** command displays the slack of specified pins and ports. By

report_global_slack

default all pins except those of hierarchical cells and ports of the design are reported.

In order to report slacks throughout the current design arrival totals and slacks must be available on all pins, not just at endpoints. To achieve this, the variable **timing_save_pin_arrival_and_slack** should be set to **true** before this command is issued. If **timing_save_pin_arrival_and_slack** has not been set to TRUE then the command will set it to TRUE and will update the design timing before the command executes. If you intend to use this command it is recommended that you set the **timing_save_pin_arrival_and_slack** variable to **true** before the first timing update, thus preventing the cost of an additional timing update.

The *max* and *min* options are mutually exclusive. The *rise* and *fall* options are also mutually exclusive. You can choose any combination from the *max* or *min* option and *rise* or *fall* option to report a particular slack value. If only the *max* option, but not the *rise* or *fall* option, is specified, both the *max_rise* and *max_fall* slack values are reported. If only the *min* option, but not the *rise* or *fall* option is specified, both the *min_rise* and *min_fall* slack values are reported. If only the *rise* option, but not the *max* or *min* option is specified, both the *max_rise* and *min_rise* slack values are reported. If only the *fall* option, but not the *max* or *min* option is specified, both the *max_fall* and *min_fall* slack values are reported. If neither the *rise* and *fall* options nor the *max* and *min* options are specified, the *max_rise*, *max_fall*, *min_rise*, and *min_fall* slack values are reported. If slack attribute is not available at a pin or port, * is printed.

Note that this command expects that the design has already been timed with the **timing_save_pin_arrival_and_slack** variable set to *true*. If the **timing_save_pin_arrival_and_slack** variable is *false*, the **report_global_slack** command can potentially take a longer runtime.

EXAMPLES

The following example generates a report of all slack violations in the current design.

```
pt_shell> report_global_slack -significant_digits 4 -nosplit
```

```
*****
Report : report_global_slack
Design : tc1
*****
```

Max_Rise	Max_Fall	Min_Rise	Min_Fall	Point
-1.4869	-1.6330	1.5669	1.7130	ff2/D
-1.4869	-1.6330	1.5669	1.7130	in2/Z
-1.6330	-1.4869	1.7130	1.5669	in2/A
-1.6330	-1.4869	1.7130	1.5669	in1/Z

SEE ALSO

`report_timing(2)`

report_hierarchy

Reports the reference hierarchy of the current_instance or current_design.

SYNTAX

```
string report_hierarchy [-full] [-noleaf] [-nosplit]
```

ARGUMENTS

-full

Displays the full hierarchy. By default, components of submodules in multiple locations in a hierarchy are listed only once. An ellipsis (...) indicates the contents of a previously displayed module.

-noleaf

Does not display the leaf library cells.

-nosplit

Does not split lines if column overflows.

DESCRIPTION

Displays the indented reference hierarchy of the current instance or the current design. If the current instance is set, the report is generated relative to that instance; otherwise, the report is generated for the current design.

EXAMPLES

The following command reports the full hierarchy of the design.

```
pt_shell> report_hierarchy -full

*****
Report : hierarchy
Design : middle
*****

      FD2                tech_lib
      ND2                tech_lib
      inter
          low
              NR4        tech_lib
      low
          NR4            tech_lib
```

SEE ALSO

report_reference (2), **report_cell** (2).

report_host_usage

Creates a detailed report that describes all host options that you have created.

SYNTAX

```
int report_host_usage  
[-verbose]  
[-nosplit]  
[host_option_names]
```

host_option_names list

ARGUMENTS

-verbose

Indicates you want to generate a more detailed report.

-nosplit

Do not split lines when columns overflow.

host_option_names

A list of names of the host options to report. The host options must have been defined using the **set_hosts_options** command. If no host option names are specified, then all host options are reported.

DESCRIPTION

The **report_host_usage** command generates a report showing the values specified for each host options set.

Calling the **report_host_usage** command before any hosts have been started reports the following basic information.

- Options Name

The name of the host options set.

- Host

The host on which the processes will be launched. If the processes are to be launched on a managed compute resource then >>farm<< will be shown.

- 32Bit

The process launched is 32 bit or not.

- Num Processes

report_host_usage

The number of processes to be launched.

If the `-verbose` option is specified, the following additional information is reported.

- Action

The actions that can be performed for the host options. `SUBMIT` : Command to submit a process for execution.

- Command

The user provided command to perform the associated action.

Once the hosts have been started using the `start_hosts` command, then calling the `report_host_usage` command will additionally show the current "real time" details of the processes launched by default.

- #

The internal instance number of the process. When each slave instance is launched by the `start_hosts` command it is assigned an instance number. This is to allow you to relate the process to the output of the `start_hosts` command.

- Host Name

The name of the actual host on which the process is running.

- Job ID

If the options specified access a managed resource (e.g. LSF, SGE etc) then this is the ID of the job submitted to the farm otherwise it is the ID of the process that launched the slave PrimeTime process. If this value is -1 then the job ID either could not be determine or the process was shutdown in which case the ID is no longer valid.

- Process ID

This is the ID of the slave PrimeTime process. If this value is -1 then the process ID either could not be determined or the process was shutdown in which case the ID is no longer valid.

- Status

This is the status of the slave process. The possible values are

`LAUNCHED` : The process is launched but not ready for use yet.

ONLINE : The process is online and ready for use.
 SHUTDOWN : The process was shut down.
 FATAled : The process abnormally terminated.
 MISSALIGNED : The master a slave processes are different versions
 of PrimeTime and the slave process is not usable.

EXAMPLES

In the following example, the master process is running on the host named ptopt018 using a 64-bit binary and specifies five different sets of host options. The first host options, named "my_opts1", specifies 1 process with the same architecture as the master on the same host as the master. The second host options, named "my_opts2", specifies 2 processes using the 32-bit binary on the same host as the master but explicitly mentions the name of the master's host. The third host options, named "my_opts3", specifies 4 processes (with the same architecture as the master) on the host named ptopt010 using rsh to connect to ptopt010. The fourth host options named "my_opts4", specifies 5 processes (with the same architecture as the master) on an LSF farm requesting 500MB of memory on the compute servers. The fifth host options, named "my_opts5", specifies 2 processes (with the same architecture as the master) on a Grid farm requiring the jobs to be launched using the project named bnormal.

Note: The host name of the host for my_opts1 is inferred as localhost (i.e. the master). The host name for my_opts1 and my_opts2 are surrounded by "***", indicating that the processes will be launched directly on the host on which the master process is running. The host name for all managed resources is ">>farm<<".

Call the **set_host_options** command to specify the host options.

```

pt_shell> set_host_options -name my_opts1
1
pt_shell> set_host_options -name my_opts2 -32bit -num_processes 1 \
ptopt018
1
pt_shell> set_host_options -name my_opts3 -num_processes 1 \
-submit_command "/usr/bin/rsh -n" ptopt010
1
pt_shell> set_host_options -name my_opts4 -num_processes 1 \
-submit_command "[sh which bsub] -R rusage\[mem=500\]"
1
pt_shell> set_host_options -name my_opts5 -num_processes 1 \
-submit_command "[sh which qsub] -b y -P bnormal"
1
  
```

Call the **report_host_usage** command with the **-verbose** option before starting the hosts to report the host options created by the **set_host_options** command. Note: The submit command for my_opts1 and my_opts2 are empty because the processes will be directly launched on the masters host.

```

pt_shell> report_host_usage -verbose
*****
  
```



```
Report : host_usage
        -verbose
Version: D-2009.12-alpha-DEV
Date   : Wed Jul 29 09:07:42 2009
*****
```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	Action	Command
my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

Call the **start_hosts** command to start the slave processes. Notice that each instance launched is assigned a number e.g. 1] Launched ...

```
pt_shell> start_hosts
1] Launched : ...
    ...
    Status  : Forking successful
    Stdout  : Process group is (7199)
    Stderr  : **<<EMPTY>>**

2] Launched : ...
    ...
    Status  : Forking successful
    Stdout  : Process group is (7233)
    Stderr  : **<<EMPTY>>**

3] Launched : ...
    ...
    Status  : Forking successful
    Stdout  : Process group is (7269)
    Stderr  : **<<EMPTY>>**

4] Launched : ...
    ...
    Status  : Forking successful
    Stdout  : Process group is (8481)
    Stderr  : **<<EMPTY>>**
```

```

5] Launched : ...
    ...
    Status : Forking successful
    Stdout : Process group is (8558)
    Stderr : **<<EMPTY>>**

6] Launched : ...
    ...
    Status : Forking successful
    Stdout : Process group is (8629)
    Stderr : **<<EMPTY>>**

7] Launched : ...
    ...
    Status : Forking successful
    Stdout : Process group is (8716)
    Stderr : **<<EMPTY>>**

8] Launched : ...
    ...
    Status : Forking successful
    Stdout : Job <627124> is submitted to default queue <normal>.
    Stderr : **<<EMPTY>>**

9] Launched : ...
    ...
    Status : Forking successful
    Stdout : Job <627125> is submitted to default queue <normal>.
    Stderr : **<<EMPTY>>**

10] Launched : ...
    ...
    Status : Forking successful
    Stdout : Job <627126> is submitted to default queue <normal>.
    Stderr : **<<EMPTY>>**

11] Launched : ...
    ...
    Status : Forking successful
    Stdout : Job <627127> is submitted to default queue <normal>.
    Stderr : **<<EMPTY>>**

12] Launched : ...
    ...
    Status : Forking successful
    Stdout : Job <627128> is submitted to default queue <normal>.
    Stderr : **<<EMPTY>>**

13] Launched : ...
    ...
    Status : Forking successful
    Stdout : Your job 8998610 ("pt_shell") has been submitted
    Stderr : **<<EMPTY>>**

```

```

14] Launched : ...
      ...
      Status  : Forking successful
      Stdout   : Your job 8998611 ("pt_shell") has been submitted
      Stderr   : **<<EMPTY>>**

```

```

-----
Distributed farm creation timeout : 21600 seconds
Waiting for 14 (of 14) distributed hosts (Wed Jul 29 09:07:07 2009)
Waiting for  9 (of 14) distributed hosts (Wed Jul 29 09:07:17 2009)
Waiting for  0 (of 14) distributed hosts (Wed Jul 29 09:07:27 2009)
-----

```

1

Call the **report_host_usage** command with the **-verbose** option after starting the hosts to report the host options created by the **set_host_options** command, as well as the real time data associated with the processes. Notice that number of the instance launched is reported with the real time data.

```

pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:07:42 2009
*****

```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	#	Host Name	Job ID	Process ID	Status
my_opts1	1	ptopt018	7199	7199	ONLINE
my_opts2	2	ptopt018	7233	7233	ONLINE
	3	ptopt018	7269	7269	ONLINE
my_opts3	4	ptopt010	8481	8481	ONLINE
	5	ptopt010	8558	8558	ONLINE
	6	ptopt010	8629	8629	ONLINE
	7	ptopt010	8716	8716	ONLINE
my_opts4	8	maddog108	627124	7718	ONLINE
	9	maddog108	627125	7715	ONLINE
	10	maddog108	627126	7717	ONLINE
	11	maddog108	627127	7716	ONLINE

	12	maddog108	627128	7714	ONLINE
my_opts5	13	peopf11	8998610	30080	ONLINE
	14	peopf1	8998611	11076	ONLINE

Options Name	Action	Command

my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

SEE ALSO

set_host_options (2)
remove_host_options (2)
start_hosts (2)
stop_hosts (2)

report_ideal_network

Displays information about ports, pins, nets, and cells on ideal networks in the current design.

SYNTAX

```
int report_ideal_network
    [-net]
    [-cell]
    [-load_pin]
    [-timing]
    [object_list]
```

list *object_list*

ARGUMENTS

-net

Display all nets in the specified ideal networks. By default, nets are displayed for all ideal networks.

-cell

Display all cells in the specified ideal networks. By default, cells are displayed for all ideal networks.

-load_pin

Specifies to display load pins (boundary pins) of the specified ideal networks along with any ideal timing set on them using the **set_ideal_latency** and **set_ideal_transition** commands. By default, load pins are displayed for all ideal networks.

-timing

Specifies to display all internal pins (non-source and non-boundary pins) in the specified ideal networks that have ideal timing set on them using the **set_ideal_latency** and **set_ideal_transition** commands. By default, internal pins with timing are displayed for all ideal networks.

object_list

Defines a list of source ports or source pins of the specified ideal networks to be displayed. By default, the source pins and source ports for all ideal networks in the current design are displayed.

DESCRIPTION

Displays information about the ideal networks in the current design. If no arguments are specified, all ideal network sources in the current design are displayed. If you specify a list of source pins or ports, the command displays information about the ideal networks emanating from these objects.

The "Latency" and "Transition" columns show the minimum and maximum values set for

both rising and falling edges. You set these values with the **set_ideal_latency** and **set_ideal_transition** commands.

EXAMPLES

The following example sets up an ideal network, applies ideal latency and ideal transition values and shows the output of the ideal network report.

```
pt_shell> set_ideal_network {p_in in1}
pt_shell> set_ideal_latency -min 1.12 in1
pt_shell> set_ideal_transition -max 9.87 in1
pt_shell> set_ideal_latency -min 1.34 n0_i/Z
pt_shell> set_ideal_transition -rise 5.62 n3_i/B
pt_shell> report_ideal_network -timing -load_pin -cell -net
```

```
*****
```

```
Report : ideal_network
```

```
-timing
```

```
-load_pin
```

```
-net
```

```
-cell
```

```
Design : middle
```

```
Version: Y-2006.06
```

```
Date : Fri Feb 24 10:03:04 2006
```

```
*****
```

Source ports and pins	Latency				Transition		
	Rise		Fall		Rise		Fall
	min	max	min	max	min	max	min
max	-----						

middle/in1	1.12	--	1.12	--	--	9.87	--
9.87							
middle/p_in	--	--	--	--	--	--	--
--							
Internal pins with ideal timing	Latency				Transition		
	Rise		Fall		Rise		Fall
	min	max	min	max	min	max	min
max	-----						

n3_i/B	--	--	--	--	5.62	5.62	--
--							
n0_i/Z	1.34	--	1.34	--	--	--	--
--							
Boundary pins	Latency				Transition		
	Rise		Fall		Rise		Fall
	min	max	min	max	min	max	min
max	-----						

```

-----
o_reg3/D          --      --      --      --      --      --      --
--
middle/p_out      --      --      --      --      --      --      --
--

Nets
-----
-----
in1
p_in
p_out
n2
n1
n0

Cells
-----
-----
n3_i
n2_i
n1_i
n0_i

1

```

The following example shows the output of the ideal network report on specified objects.

```

pt_shell> report_ideal_network -timing -load_pin -cell -net {in1}

*****
Report : ideal_network
        -timing
        -load_pin
        -net
        -cell
Design : middle
Version: Y-2006.06
Date   : Fri Feb 24 10:05:40 2006
*****

Source ports          Latency          Transition
and pins              Rise              Fall              Rise              Fall
                     min    max         min    max         min    max         min
max
-----
middle/in1            1.12   --         1.12   --         --     9.87         --
9.87

Boundary pins         Latency          Transition
                     Rise              Fall              Rise              Fall
                     min    max         min    max         min    max         min

```

```

max
-----
o_reg3/D      --      --      --      --      --      --
--
Nets
-----
in1
1

```

SEE ALSO

```

remove_ideal_network (2),
report_constraint (2),
set_ideal_latency (2),
set_ideal_network (2),
set_ideal_transition (2).

```


report_lib

Reports library information.

SYNTAX

```
string report_lib [-timing_arcs] [-power_arcs] [-nosplit] library_name  
[lib_cell_list]  
list library_name  
list lib_cell_list
```

ARGUMENTS

-timing_arcs

Displays timing arc information. Indicates to list all cell timing arcs.

-power_arcs

Displays power arc information. Indicates to list all cell power arcs.

-nosplit

Prevents splitting lines if column overflows. Some of the information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

library_name

A pattern matching a single library or a collection containing one library. The library must have been read using **read_db**.

lib_cell_list

Displays a list of library cells about which information is reported. The default is to report information about all cells in the technology library. Each element in this list is either a collection of library cells or a pattern which matches library cells in *library_name*.

DESCRIPTION

A library report displays library information including a list of operating conditions, wire load models, and available library cells. The **-timing_arcs** option lists detailed timing arc information for each library cell, while the **-power_arcs** option lists detailed power arc information. Please note that the **-power_arcs** option and the **-timing_arcs** option are currently mutually exclusive. **report_lib** will also indicate if the library is a max library in a max/min relationship to a min library created by **set_min_library**.

To generate a report, the specified library must be loaded into pt_shell.

EXAMPLES

This example shows the default library report.

```
pt_shell> read_db tech_lib.db
pt_shell> report_lib tech_lib
```

```
*****
Report : library
Library: tech_lib
*****
```

```
Time Unit           : 1 ns
Capacitance Unit    : 1 pF
Default Wire Load Mode : top
Default Wire Load Model : Not specified.
Default Wire Selection Group: Not specified.
```

Operating Conditions:

Name	Process	Temp	Voltage	Tree Type
BCCOM	0.60	0.00	5.25	best_case
TYPICAL	1.00	25.00	5.00	balanced_case
WCCOM	1.50	70.00	4.75	worst_case

Wire Load Models:

```
Name
-----
small_wl
medium_wl
large_wl
```

Library Cells:

```
Attributes:
b - black box (function unknown)
```

```
Lib Cell  Attributes
-----
AN2
BUFA
BUFB
FF
IV
LTCH
MUX21
OR2
```

In this example, timing arc information is shown for a list of library cells.

```
pt_shell> report_lib -timing_arcs tech_lib {AN2 OR2}
```

```
*****
Report : library
```

```

        -timing_arcs
Library: tech_lib
*****

```

Library Cells:

```

Attributes:
  b - black box (function unknown)

```

Lib Cell	Attributes	Arc		Arc Pins		When
		#	Sense/Type	From	To	
AN2		0	unate	A	Z	
		1	unate	B	Z	
OR2		0	unate	A	Z	
		1	unate	B	Z	

In the example below, power arc information is shown for a list of library cells.

```

pt_shell> report_lib -power_arcs tech_lib {AN2 OR2}

```

```

*****
Report : library
        -power_arcs
Library: tech_lib
*****

```

```

Time Unit           : 1 ns
Capacitance Unit    : 1 pF
Pulling Resistance Unit : 10ohm
Voltage Unit         : 1 V
Current Unit         : 1e-06 A
Leakage Power Unit   : 1e-06 W

```

Scaling Factors:

```

*Global*
-----
k_process_rise_propagation : 0.000000
k_volt_rise_propagation    : 0.000000
k_temp_rise_propagation    : 0.000000
k_process_fall_propagation : 0.000000
k_volt_fall_propagation    : 0.000000
k_temp_fall_propagation    : 0.000000
k_process_rise_transition  : 0.000000
k_volt_rise_transition     : 0.000000
k_temp_rise_transition     : 0.000000
k_process_fall_transition  : 0.000000
k_volt_fall_transition     : 0.000000
k_temp_fall_transition     : 0.000000
k_process_cell_rise       : 0.000000
k_volt_cell_rise          : 0.000000

```

k_temp_cell_rise	:	0.000000
k_process_cell_fall	:	0.000000
k_volt_cell_fall	:	0.000000
k_temp_cell_fall	:	0.000000
k_process_internal_power	:	0.000000
k_volt_internal_power	:	0.000000
k_temp_internal_power	:	0.000000
k_process_cell_leakage_power	:	0.000000
k_volt_cell_leakage_power	:	0.000000
k_temp_cell_leakage_power	:	0.000000

AN2_factors

k_process_rise_propagation	:	0.000000
k_volt_rise_propagation	:	0.000000
k_temp_rise_propagation	:	0.000000
k_process_fall_propagation	:	0.000000
k_volt_fall_propagation	:	0.000000
k_temp_fall_propagation	:	0.000000
k_process_rise_transition	:	0.000000
k_volt_rise_transition	:	0.000000
k_temp_rise_transition	:	0.000000
k_process_fall_transition	:	0.000000
k_volt_fall_transition	:	0.000000
k_temp_fall_transition	:	0.000000
k_process_cell_rise	:	1.000000
k_volt_cell_rise	:	-0.440000
k_temp_cell_rise	:	0.002120
k_process_cell_fall	:	1.000000
k_volt_cell_fall	:	-0.440000
k_temp_cell_fall	:	0.002120
k_process_internal_power	:	0.000000
k_volt_internal_power	:	0.000000
k_temp_internal_power	:	0.000000
k_process_cell_leakage_power	:	0.000000
k_volt_cell_leakage_power	:	0.000000
k_temp_cell_leakage_power	:	0.000000

OR2_factors

k_process_rise_propagation	:	0.000000
k_volt_rise_propagation	:	0.000000
k_temp_rise_propagation	:	0.000000
k_process_fall_propagation	:	0.000000
k_volt_fall_propagation	:	0.000000
k_temp_fall_propagation	:	0.000000
k_process_rise_transition	:	0.000000
k_volt_rise_transition	:	0.000000
k_temp_rise_transition	:	0.000000
k_process_fall_transition	:	0.000000
k_volt_fall_transition	:	0.000000
k_temp_fall_transition	:	0.000000
k_process_cell_rise	:	1.000000
k_volt_cell_rise	:	-0.440000
k_temp_cell_rise	:	0.002120

k_process_cell_fall	:	1.000000
k_volt_cell_fall	:	-0.440000
k_temp_cell_fall	:	0.002120
k_process_internal_power	:	0.000000
k_volt_internal_power	:	0.000000
k_temp_internal_power	:	0.000000
k_process_cell_leakage_power	:	0.000000
k_volt_cell_leakage_power	:	0.000000
k_temp_cell_leakage_power	:	0.000000

Operating Conditions:

Name	Process	Temp	Voltage	Tree Type
<lib_default>	1.00	27.00	2.50	balanced_case
B	0.63	-30.00	2.80	balanced_case
BB	0.63	-30.00	2.80	best_case
BG	0.63	-30.00	2.75	balanced_case
TYPICAL	1.00	27.00	2.50	balanced_case
W	1.37	125.00	2.20	balanced_case
WGSM	1.37	85.00	2.25	balanced_case
WW	1.37	125.00	2.20	worst_case

Input Voltages:

No input_voltage groups specified.

Output Voltages:

No output_voltage groups specified.

Wire Loading Model:

No wire loading specified.

Wire Loading Model Mode: top.

Delay Threshold Trip-Points

input_threshold_pct_rise	:	NOT SPECIFIED
output_threshold_pct_rise	:	NOT SPECIFIED
input_threshold_pct_fall	:	NOT SPECIFIED
output_threshold_pct_fall	:	NOT SPECIFIED

Slew Threshold Trip-Points

slew_lower_threshold_pct_rise	:	NOT SPECIFIED
slew_upper_threshold_pct_rise	:	NOT SPECIFIED
slew_lower_threshold_pct_fall	:	NOT SPECIFIED
slew_upper_threshold_pct_fall	:	NOT SPECIFIED
slew_derate_from_lib	:	NOT SPECIFIED

Power Supply Group:

No power supply group specified.

default_cell_leakage_power : 0
default_leakage_power_density : NOT SPECIFIED

Power Information:

Attributes:

- a - average power specification
- i - internal power
- l - leakage power
- rf - rise and fall power specification

Cell	Attributes	#	Power Toggling	pin	Source of path	When

--						
AN2	1	0				
	i,rf	1	A			!B
	i,rf	2	B			!A
	i,rf	3	Z		A	B
	i,rf	4	Z		B	A
	i,a	5	Z			
OR2	1	0				
	i,rf	1	B			A
	i,rf	2	A			B
	i,rf	3	Z		B	!A
	i,rf	4	Z		A	!B
	i,a	5	Z			

SEE ALSO

read_db (2), **set_min_library** (2).

report_lib_groups

Generates a report of library groups.

SYNTAX

```
int report_lib_groups  
[-scaling]  
[-show]  
[-nosplit]  
[show_list]
```

Data Types

show_list list

ARGUMENTS

-scaling

Reports the list of library groups defined for scaling. Scaling library groups can be defined using the **define_scaling_lib_group** command.

-show

Reports the list of options in the *show_list* option. Specify this option if you want detailed information for the libraries in the scaling group, such as voltage and temperature. If this option is used, you must also specify the *show_list* option.

-no_split

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

show_list

Reports the specific options that need to be reported. The *show_list* option can only contain options from the list {temperature, voltage, extended_name}. You can use any combination of these options. The *show_list* option cannot be empty. If the -show option is specified, you must also specify a non-empty *show_list*.

DESCRIPTION

Generates a report of the scaling library groups specified in the design.

The *scaling* option allows you to view all the scaling library groups specified using the **define_scaling_lib_group** command.

You can obtain detailed information about the libraries by using the -show option. The -scaling option is required.

EXAMPLES

The following example shows the scaling relationships that exist in the design.

```
pt_shell> report_lib_groups -scaling
*****
Report : lib_groups
        -scaling
Design : test
Version: Y-2006.06-Alpha-DEV
Date   : Wed Mar  8 11:22:51 2006
*****
```

Group	Library

Group 1	
	lib_0.95
	lib_0.85
1	

The following example shows how you can use the *-show* option to obtain more detailed library information.

```
pt_shell> report_lib_groups -scaling -show {voltage temperature}
*****
Report : lib_groups
        -scaling
        -show
Design : test
Version: Y-2006.06
Date   : Wed Mar  8 11:50:28 2006
*****
```

Group	Library	Temperature	Voltage

Group 1			
	lib_0.95	125.00	0.95
	lib_0.85	125.00	0.85
1			

SEE ALSO

```
define_scaling_lib_group(2)
report_lib(2)
```


report_min_pulse_width

Displays minimum pulse width check information about specified pins or ports.

SYNTAX

```
int report_min_pulse_width
    [-all_violators]
    [-significant_digits digits]
    [-nosplit]
    [-path_type format]
    [-input_pins]
    [-derate]
    [port_pin_list]
```

```
list port_pin_list
```

ARGUMENTS

-all_violators

Indicates that only violating minimum pulse width checks are to be reported.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line splitting and facilitates writing software to extract information from the report output.

-path_type *format*

Specifies the format of the path report and how the clock path is displayed. Allowed values are: *summary* (the default), which generates a report with a column format that shows one line for each path and shows only the required pulse width, actual pulse width and slack; *short*, which displays only start and end points in the clock path; *full_clock*, which displays full clock paths; and *full_clock_expanded*, which displays full clock paths including all master clocks of a generated clock.

-input_pins

Indicates that input pins are to be shown in the path report. The default is to show only output pins.

-derate

Indicates that derate factors are to be shown in the path report. The default is to show no derate factors. This will also show the derate factor specified on the required pulse width if one has been specified by **set_timing_derate**.

Note that this option applies only when *-path_type* is set to *full_clock* or *full_clock_expanded*.

`port_pin_list`

Specifies a list of pins or ports to report. By default, the report contains all pins and ports in the current design.

DESCRIPTION

The **report_min_pulse_width** command displays the following information for the minimum pulse width check: required pulse width, actual pulse width, and by how much the constraint is violated or met (slack). This information is listed in order of decreasing slack starting with the worst violator.

If the **timing_remove_clock_reconvergence_pessimism** variable is set to true, clock reconvergence pessimism is removed from the slack on the reports.

More detailed minimum pulse width calculations are shown using the **-path_type** and **-input_pins** options.

Minimum pulse width violations can also be reported with the **report_constraint** command using the **-min_pulse_width** option.

Note that if the variable **timing_enable_pulse_clock_constraints** is set to **TRUE** (default value), **report_min_pulse_width** does not report pulse width checks in the pulse clock networks. To report pulse width checks in the pulse clock network use **report_pulse_clock_min_width**. To check pulse width in the pulse clock network using **report_min_pulse_width** command, set **timing_enable_pulse_clock_constraints** to **FALSE**.

EXAMPLES

The following example generates a report of all minimum pulse width violations in the current design.

```
pt_shell> report_min_pulse_width
```

```
*****
Report : min pulse width
        -path_type summary
Design : middle
Version: Z-2006.12-Beta4-DEV
Date   : Wed Nov  8 09:06:01 2006
*****
```

```
sequential_clock_pulse_width
```

Pin	Required pulse width	Actual pulse width	Slack	
ff1/CP	10.20	10.00	-0.20	(VIOLATED)
ff2/CP	10.20	10.04	-0.16	(VIOLATED)

ff3/CP	2.10	2.00	-0.10	(VIOLATED)
ff3/CP	2.10	2.00	-0.10	(VIOLATED)
ff2/CP	10.00	9.96	-0.04	(VIOLATED)

clock_tree_pulse_width

Pin	Required pulse width	Actual pulse width	Slack	
nand1/Z	10.00	9.58	-0.42	(VIOLATED)
or1/B	10.00	9.58	-0.42	(VIOLATED)
nand1/A	10.20	10.00	-0.20	(VIOLATED)
or1/Z	10.20	10.04	-0.16	(VIOLATED)
or1/Z	10.00	9.96	-0.04	(VIOLATED)

SEE ALSO

```
remove_min_pulse_width (2), report_constraint (2), report_timing (2),
set_min_pulse_width (2); report_default_significant_digits (3),
timing_remove_clock_reconvergence_pessimism (3), set_pulse_clock_min_width (2),
report_pulse_clock_min_width (2), timing_enable_pulse_clock_constraints (2).
```

report_mode

Displays a report of modes for specified cells or the design

SYNTAX

```
string report_mode [-type cell | design] [-nosplit] [instance_list]  
list instance_list
```

ARGUMENTS

-type design | cell

Indicates the type of mode to be reported. This option has the following mutually-exclusive valid values: **design** and **cell**. The **cell** value specifies that cell modes are to be reported. Cell modes are defined on library cells in the library. The **design** value specifies that design modes are to be reported. Design modes are user specified using **define_design_mode_group** and **map_design_mode**. If the **-type** option is omitted from the command then this is equivalent to specifying **-type cell**

-nosplit

Most of the mode information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

instance_list

Specifies a list of instances whose modes are to be reported. By default, the report includes all cells that have modes. This option is only valid if **-type** has a cell value

DESCRIPTION

Reports cell modes or design modes. When reporting cell modes, the report specifies whether the cell mode is enabled or disabled and also specifies the reason as to why the mode is enabled or disabled. There are four possible reasons as to why a mode can be enabled or disabled.

cell - Indicates that the cell mode has been set directly using **set_mode -type cell**.

design - Indicates that the cell mode has been set due to the activation of a design mode using the **set_mode -type design** command. The design mode must have previously been mapped to the cell mode using **map_design_mode**. The report also specifies the name of the activated design mode. To view more details on the design mode, use **report_mode -type design**.

cond - Indicates that the cell mode has been set due to evaluation of the mode condition during constant propagation.

default - Indicates that the cell mode has not been set by any of the above reasons

When reporting design modes the report displays the design modes specified on the current design and for each design mode, reports the cell modes and moded pins associated with that design mode.

EXAMPLES

The following example reports cell mode information for the design top_design. Two cells have modes, Uram1 and Uram2 are instances of the library cell RAM2_core. rw is a cell mode group defined on the library cell RAM2_core. This cell mode group has two cell modes read and write. No modes are currently active so all modes are enabled and the reason is given as default.

```
pt_shell> report_mode
```

```
*****
Report : mode
Design : top_design
*****
```

Cell	Mode(Group)	Status	Reason

Uram1/core(RAM2_core)	read(rw)	ENABLED	default
	write(rw)	ENABLED	default

Uram2/core(RAM2_core)	read(rw)	ENABLED	default
	write(rw)	ENABLED	default

The following example shows a report of modes specified for the two RAMs that have modes in the design. Ram Uram1 is set in mode read, and Ram Uram2 is set in mode write. Thus, all timing arcs of RAM Uram1 associated with mode read are enabled , and all timing arcs associated with mode write are disabled.

```
pt_shell> set_mode read Uram1/core
pt_shell> set_mode write Uram2/core
pt_shell> report_mode
```

```
*****
Report : mode
Design : top_design
*****
```

Cell	Mode(Group)	Status	Reason

Uram1/core(RAM2_core)	read(rw)	ENABLED	cell
	write(rw)	disabled	cell

Uram2/core(RAM2_core)	read(rw)	disabled	cell
	write(rw)	ENABLED	cell

The following example reports that two design modes have been specified using the **set_mode -type design** command. For each design mode, it reports any mapping of cell modes, as specified by the **map_design_mode** command. The report also displays any paths that are design mode dependent, as specified by the **map_design_mode -from** or **map_design_mode -to** commands.

```
pt_shell> map_design_mode {read,latching} {U2/core,U1/core} read
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF3/D}
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF4/D}
pt_shell> map_design_mode {write,transparent} {U2/core,U1/core} write
pt_shell> report_mode -type design
```

```
*****
Report : design_mode
Design : misc
Version: v4.0a-development
Date   : Fri Mar 29 13:29:05 1996
*****
```

```
-----
Design Mode Group : 'default'
Design Mode : 'read' (is current design mode)
```

Cell	Mode (Group)
U2/core	latching(output_latch) read(rw)
U1/core	latching(output_latch) read(rw)
From Pin	
INFF1/CLK	
INFF1/CLK	
To Pin	
INFF3/D	
INFF4/D	

```
-----
Design Mode Group : 'default'
Design Mode : 'write'
```

Cell	Mode (Config)
U2/core	transparent(output_latch) write(rw)

```
U1/core                                transparent(output_latch)
                                       write(rw)
```

SEE ALSO

```
map_design_mode (2), define_design_mode_group (2), remove_design_mode (2),
reset_mode (2), set_mode (2).
```

report_multi_scenario_design

Creates a detailed report on user defined multi-scenario objects and attributes.

SYNTAX

```
Boolean report_multi_scenario_design  
  [-scenario]  
  [-session]  
  [-license]
```

ARGUMENTS

[-scenario]

Reports in detail on the current scenarios that have been defined. Details reported include the name of the scenario, the files used to generate the image for the scenario (i.e. common and specific data files), the scenario used to generate the baseline image for the scenario, the location of the current image, and the focus of the scenario. For detailed description of the settings of the focus field, see the description section below.

[-session]

Reports in detail on the current session. This option reports the number of scenarios defined in the current session. Three fields are reported for each scenario in the current session: The command focus of the scenario, the name of the scenario, and the scenario which provides the image to the scenario.

[-licenses]

Reports in detail on license usage and limits set.

DESCRIPTION

The **report_multi_scenario_design** command generates a report on user defined multi-scenario analysis. The level of detail reported depends on the options specified. By selecting no options, **report_multi_scenario_design** will issue a short summary of current user defined objects and settings. The summary reports on whether or not a session has been defined. The number of defined scenarios and licenses in use are also reported. The report generated can report in detail on the following topics: scenarios, sessions and licenses.

The **-scenario** option generates a report on all the scenarios that exist in the master process. The report shows several pieces of information about each scenario.

- The name of the scenario.
- The files used to generate the images for the scenario (i.e. common and specific data files)
- The scenario used to generate the baseline image for the scenario.

- The location of the scenario's current image.

- The focus of the scenario.

The focus will have one of three possible settings.

- "Out of focus" means the scenario is not in the current_session.

- "In current session" means the scenario is in the current_session but it is not in command focus. i.e. will not receive commands from the master.

- "In current session and command focus" means the scenario is in the current session and in command focus i.e. it will receive commands from the master.

See the **current_scenario** command for changing command focus.

The -session option reports the scenarios defined in the current session. Along with the number of scenarios, some scenario specific data is reported. The focus field of the scenario shows whether the scenario is just in the session (Session), or whether it is also in command focus (Command). Also reported is the scenario which provides the image to each scenario.

The -license option shows the numbers of licenses checked out for each feature and the user defined limits for each feature.

EXAMPLES

In the following example several scenarios are created and the current session is set. Then the **report_multi_scenario_design** is used to examine the multi scenario information available to the master.

```
pt_shell>create_scenario -name scen1 -common_data {com_s1.tcl} -
specific_data {spec_s1.tcl spec_s2.tcl spec_s3.tcl}
1
pt_shell>create_scenario -name scen2 -common_data {com_s2.tcl} -
specific_data {spec_s2.tcl}
1
pt_shell>create_scenario -name scen3 -common_data {com_s3.tcl} -
specific_data {spec_s3.tcl}
1
pt_shell>create_scenario -name scen4 -common_data {com_s3.tcl} -
specific_data {spec_s3.tcl}
1
pt_shell> current_session {scen1 scen2 scen3}
1
pt_shell> report_multi_scenario_design
```

1

```
*****
Report : multi_scenario_design
Version: W-2004.12-Beta3
Date   : Mon Nov  8 08:59:13 2004
*****
```

Summary

```
Current session           : Defined
Number of defined scenarios : 4
Number of scenarios in command focus : 3
```

1

SEE ALSO

```
create_scenario (2), current_session (2), current_scenario (2),
set_multi_scenario_license_limit (2)
```

report_name_mapping

Report the name mapping rules.

SYNTAX

```
string report_name_mapping  
[-nosplit]
```

ARGUMENTS

-nosplit
Indicates to prevent line splitting if column overflows.

DESCRIPTION

The **report_name_mapping** command reports the user-defined name mapping rules that has been created by `set_rtl_to_gate_name` commands.

EXAMPLES

In the following example, all user-defined name mapping rules are reported.

```
pp_shell> fset_rtl_to_gate_name -rtl a_net -gate n272  
1  
pp_shell> fset_rtl_to_gate_name -rtl a_pin -gate U3/D0  
1  
pp_shell> fset_rtl_to_gate_name -rtl a_cell -gate U3  
1  
pp_shell> report_name_mapping  
*****  
Report : report_name_mapping  
Design : mac  
Version: Z-2006.12-Beta-1-DEV  
Date   : Thu Sep 21 10:11:59 2006  
*****
```

RTL Name	Gate Level Name	Type
a_cell	U3	cell
a_pin	U3/D0	pin
a_net	n272	net

SEE ALSO

set_rtl_to_gate_name (2)

report_net

Generates a report of net information.

SYNTAX

```
string report_net  
[-connections [-verbose]]  
[-significant_digits digits]  
[-segments]  
[-nosplit]  
[net_names]
```

```
list net_names
```

ARGUMENTS

-connections

Indicates that the report is to show net connection information.

-verbose

Must be used with the **-connections** option. Indicates that the report is to show verbose connection information.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. Fixed-precision floating-point arithmetics is used for delay calculation; therefore, the actual precision might depend on the platform. For example, on SVR4 UNIX see FLT_DIG in limits.h. The upper bound on a meaningful value of the argument to **-significant_digits** is then $\text{FLT\_DIG} - \text{ceil}(\log_{10}(\text{fabs}(\text{any\_reported\_value})))$.

-segments

Indicates that the report is to show all global segments for each net requested. Global net segments are all those physically connected across all hierarchical boundaries.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

net_names

Specifies a list of nets to be reported. The default is to report all nets in the current instance or current design.

DESCRIPTION

The command displays information about the nets in the design of the current instance, or in the current design. If **current_instance** is set, the report is generated for the design of that instance; otherwise, the report is generated for the current design. Attributes, such as annotated resistance and annotated capacitance, are displayed.

If **-connections** is used, PrimeTime lists the leaf cell pins connected to each net.

EXAMPLES

The following example reports all nets in the design.

```
pt_shell> report_net
```

```
*****
Report : net
Design : top
Version: 1997.01-development
Date   : Wed Aug  7 09:28:05 1996
*****
```

Attributes:

c - annotated capacitance
r - annotated resistance

Net	Fanout	Fanin	Cap	Resistance	Pins	Attributes
a	2	1	2.00	0.00	3	
b	1	1	1.00	0.00	2	
c	1	1	1.00	0.00	2	
d	1	1	1.00	0.00	2	
e	1	1	1.00	0.00	2	
en	20	1	25.00	0.00	21	
f	1	1	1.00	0.00	2	
g	1	1	1.00	0.00	2	
h	1	1	1.00	0.00	2	
i1	1	1	1.00	0.00	2	
i2	1	1	1.00	0.00	2	
i3	1	1	1.00	0.00	2	
i4	1	1	1.00	0.00	2	
j	1	1	1.00	0.00	2	
k	1	1	1.00	0.00	2	
l	1	1	1.00	0.00	2	
m	2	1	2.00	0.00	3	
n	1	1	1.00	0.00	2	
o	1	1	1.00	0.00	2	
o1	1	1	0.00	0.00	2	
o2	1	1	0.00	0.00	2	
p	1	1	1.00	0.00	2	
q	2	1	2.00	0.00	3	
r	1	1	1.00	0.00	2	

s	1	1	1.00	0.00	2
t	1	1	1.00	0.00	2
u	1	1	1.00	0.00	2
w	1	1	1.00	0.00	2
y	1	1	1.00	0.00	2
z	1	1	2.00	0.00	2

Total 30 nets	52	30	56.00	0.00	82
Maximum	20	1	25.00	0.00	21
Average	1.73	1.00	1.87	0.00	2.73

The following example displays a verbose connection report for net z.

```
pt_shell> report_net -verbose -connections z
```

```
*****
```

```
Report : net
```

```
    -connections
```

```
    -verbose
```

```
Design : top
```

```
Version: 1997.01-development
```

```
Date   : Wed Aug  7 09:30:38 1996
```

```
*****
```

```
Connections for net 'z':
```

```
  pin capacitance:    2
```

```
  wire capacitance:   0
```

```
  total capacitance:  2
```

```
  wire resistance:    0
```

```
  number of drivers:  1
```

```
  number of loads:    1
```

```
  number of pins:     2
```

Driver Pins	Type	Pin Cap
-----	-----	-----
z/Z	Output Pin (NOR2)	0

Load Pins	Type	Pin Cap
-----	-----	-----
o2/B	Input Pin (BUF)	2

SEE ALSO

current_design (2), **current_instance** (2), **report_cell** (2), **report_design** (2).

report_noise

Reports noise analysis information.

SYNTAX

```
int report_noise
[-above]
[-below]
[-low]
[-high]
[-nworst_pins pin_count]
[-significant_digits digits]
[-slack_type slack_type]
[-slack_lesser_than slack_limit]
[-all_violators]
[-data_pins]
[-clock_pins]
[-async_pins]
[-verbose]
[-nosplit]
[object_list]

list object_list
```

ARGUMENTS

-above

Performs the reporting only above the rails. If this option is combined with **-low**, it reports for the noise bumps above the low rail. If it is combined with **-high**, it reports the noise bumps above the high rail. Otherwise, it reports the noise bumps above the high rail and above the low rail.

-below

Performs the reporting only below the rails. If this option is combined with **-low**, it reports for the noise bumps below the low rail. If it is combined with **-high**, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps below the high rail and below the low rail.

-low

Performs the reporting only for the low rail. If this option is combined with **-above**, it reports the noise bumps above the low rail. If it is combined with **-below**, it reports the noise bumps below the low rail. Otherwise, it reports the noise bumps for both below and above the low rail.

-high

Performs the reporting only for the high rail. If this option is combined with **-above**, it reports the noise bumps above the high rail. If it is combined with **-below**, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps for both below and above the high rail.

-nworst_pins *pin_count*

Specifies the number of load pins to be reported. Any number greater than 1

is accepted; the default value is 1.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-slack_type *slack_type*

Specifies the type of slack to be used. Valid values are *area*, *height*, and *area_percent*. A *slack_type* of *area* reports slack as the voltage margin multiplied by the noise bump width. The voltage margin is defined by the noise bump height and noise immunity curves or DC noise margin. This setting is the default. A *slack_type* of *height* reports noise slack as the voltage margin. A *slack_type* of *area_percent* reports noise slack as the percentage of the noise constraint area. The noise constraint area is computed by multiplying the noise height constraint by the noise bump width.

-slack_lesser_than *slack_limit*

Indicates that only those pins with a slack less (more negative) than *slack_limit* are to be shown.

-all_violators

Indicates that only violating pins (negative slack) are to be shown. This option cannot be used with the **-slack_lesser_than** option. If this option is used with the **-nworst_pins** option, the number of violating pins will be limited by that value.

-data_pins

Indicates that the reporting is done only on pins that are data pins of sequential cells. The effect is similar to preselect the data pins using **all_registers -data_pins** and pass the resulting collection to the **report_noise** command.

-clock_pins

Indicates that the reporting is done only on pins that are clock pins of sequential cells. The effect is similar to preselect the clock pins using **all_registers -clock_pins** and pass the resulting collection to the **report_noise** command.

-async_pins

Indicates that the reporting is done only on the asynchronous pins of sequential cells. The effect is similar to preselect the asynchronous pins using **all_registers -async_pins** and pass the resulting collection to the **report_noise** command.

-verbose

Shows more details about the calculation of total noise on each load pin, including the individual contribution of each aggressor as well as noise bumps propagated from previous stages of the design.

-nosplit

If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

object_list

Specifies the load pins for which the noise reporting is performed. If no pin is specified, reporting is performed on the entire design.

DESCRIPTION

Provides a report of noise information for the current design.

By default, the non verbose report is generated which shows the total crosstalk noise bump height and width on the load pins. If the **-verbose** option is used, the individual contribution of each effective aggressor, and also the amount of propagated noise is also reported.

EXAMPLES

The following example generates a report for the worst total noise bump above the low rail in the current design.

```
pt_shell> report_noise -above -low
```

The following example generates a report for the five worst total noise bump above the low rail in the current design.

```
pt_shell> report_noise -above -low -nworst_pins 5
```

The following example generates a report for the worst total noise bump above the low rail and also the worst total noise bump below the low rail in the current design.

```
pt_shell> report_noise -low
```

The following example generates a report for the worst total noise bump for each of the four cases of above the low rail, below the low rail, above the high rail, and below the high rail in the current design.

```
pt_shell> report_noise -low -high
```

SEE ALSO

```
update_noise (2); report_default_significant_digits (3);
```

report_noise_calculation

Displays the actual calculation of noise information for the specified net arc.

SYNTAX

```
int report_noise_calculation
[-above]
[-below]
[-low]
[-high]
[-significant_digits digits]
[-nosplit]
-from from_pin
-to to_pin

int digits
list from
list to
```

ARGUMENTS

- above
Specifies a noise calculation for above ground or power rails noise analysis region.
- below
Specifies a noise calculation for below ground or power rails noise analysis region.
- low
Specifies a noise calculation for ground rail noise.
- high
Specifies a noise calculation for power rail noise.
- significant_digits
Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.
- nosplit
Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

-from *from_pin* -to *to_pin*

Specifies the start and end points of a net arc within a design. The *from_pin* must be a driver pin or port, and the *to_pin* must be the load pin or port of the same net.

DESCRIPTION

Displays details of the noise calculation for the specified net arc. This command uses the noise analysis settings for the design set by the command **set_noise_parameters**, and also performs a noise update if needed.

The report shows some general information about the victim, as well as details of noise bump and noise slack calculations.

The general information section includes the victim driver net, pin, noise composite aggressor mode and also any possible derating factors set by **set_noise_derate** command.

The noise calculation section includes the total noise bump as well as the estimated noise bumps induced from each of the aggressors. It is noted that individual aggressor noise bumps are reported only to show their relative contributions. The linear summation of all individual noise bumps may not match the total noise bump due to the nonlinear behavior of the coupled clusters. The aggressor attribute column shows more details about each of the effective aggressors. Aggressors which are filtered are not included in this report. The following is the list of possible attributes an aggressor can have:

Attributes: A - aggressor is active C - aggressor is a composite aggressor D - aggressor is analyzed with detailed engine E - aggressor is screened due to user exclusion G - aggressor is analyzed with gate level simulator I - aggressor has infinite window L - aggressor is screened due to logical correlation S - aggressor is screened due to small bump height X - aggressor is screened due to aggressor exclusion

Active aggressor is an aggressor which contributes to the worst case alignment scenario of all the aggressors.

Composite aggressor models the combined effect of a group of weak aggressors to improve runtime and memory capacity.

Aggressor is analyzed with detailed noise calculation engine, if the bump height that it induces is larger than the threshold set by variables **si_noise_effort_threshold_within_rails** or **si_noise_effort_threshold_beyond_rails**.

Aggressor can be screened from the analysis if it has no significant impact on the victim, for example when the aggressor is excluded by using the **case_analysis** or **set_si_delay_analysis** commands, or the bump height that it induces on the victim is very small.

Aggressor is analyzed using CCS Noise gate level simulation engine if the amplitude of the total noise glitch is significant compared to the lowest noise immunity of the load cells. In this mode, all aggressor drivers as well as the victim driver are analyzed using CCS Noise advanced engine and this attribute 'G' shows up on the total noise line.

An aggressor has infinite window when it has no fixed timing relationship with the victim, e.g. an aggressor which is driven by a clock asynchronous to the clock that drives the victim.

Aggressor has logical correlation if there is a common point in the transitive fanin of both aggressor and the victim, or two aggressors.

Aggressor can be excluded from the analysis by user by using the **case_analysis** or **set_si_delay_analysis** commands.

The noise slack calculation section shows the details of how the area slack and height slack is derived for the victim, and also what kind of constraint was used to derive the slack. The height slack is the difference of the actual bump height versus the bump height which causes the failure with the same width. The area slack is the area that needs to be added to the actual bump to cause a noise failure.

EXAMPLES

The following example specifies a noise calculation report for the driver pin buf2/ZN and load pin buf5/I for below the high rail noise region:

```
pt_shell> report_noise_calculation -below -high -from buf2/ZN -to buf5/I
```

```
*****
Report : noise_calculation
        -from buf2/ZN
        -to buf5/I
        -high
        -below
Design  : testcase
Version: Y-2006.06-Beta4-DEV
Date    : Wed May 10 17:24:09 2006
*****
```

Units: 1ns 1pF 1kOhm

```
Analysis mode      : report_at_source
Region            : below_high
Victim driver pin  : buf2/ZN
Victim driver library cell : mylib/INVD3
Victim net         : I2
Victim driver effective capacitance : 0.200827

Steady state resistance source : library set CCS noise iv curve
Driver voltage swing           : 1.080000
Noise derate height offset     : 0.000000
Noise derate height scale factor : 1.000000
Noise derate width scale factor : 1.000000
Noise effort threshold         : 0.000000
Noise composite aggressor mode : disabled
```

Noise calculations:

Attributes:

- A - aggressor is active
- C - aggressor is a composite aggressor
- D - aggressor is analyzed with detailed engine
- E - aggressor is screened due to user exclusion
- G - aggressor is analyzed with gate level simulator
- I - aggressor has infinite window
- L - aggressor is screened due to logical correlation
- S - aggressor is screened due to small bump height
- X - aggressor is screened due to aggressor exclusion

	Height	Width	Area	Aggressor Attributes

Aggressors:				
I1	0.300604	0.786466	0.118207	A I D
I3	0.300604	0.786466	0.118207	A I D
Total:	0.497124	0.919737	0.228612	G

Noise slack calculation:

Constraint type: library CCS noise immunity

	Height	Area

Required	0.581570	(0.581570 * 0.919737) -
Actual	0.497124	(0.497124 * 0.919737)

Slack	0.084445	0.077668

SEE ALSO

update_noise (2), **report_noise** (2), **set_noise_derate** (2), **set_noise_parameters** (2), **si_noise_composite_aggr_mode** (3), **set_si_aggressor_exclusion** (2).

report_noise_parameters

Reports status of the noise analysis parameters for the current design.

SYNTAX

```
int report_noise_parameters
```

DESCRIPTION

This command reports status of the parameters that are considered during the noise analysis. If this command is performed, the settings of the parameters for the noise analysis are reported: beyond the rail analysis, arrival times of aggressors, analysis effort level, and propagation of noise.

EXAMPLES

The following example shows the report format:

```
pt_shell> report_noise_parameters
report_noise_parameters
*****
Report   : noise_parameters
Design   : top
Version  : Y-2006.06-VA-STA-Beta1-DEV
Date     : Thu Jan 26 09:46:43 2006
*****

analysis effort      : low
ignore arrival       : true
include beyond Rails : true
enable propagation   : true
```

SEE ALSO

```
update_noise (2); report_noise (2); set_noise_parameters (2); reset_noise_parameters
(2); si_noise_effort_threshold_within_rails (3).
si_noise_effort_threshold_within_rails (3).
```

report_noise_violation_sources

Reports noise violation sources for failing endpoints.

SYNTAX

```
int report_noise_violation_sources  
[-above]  
[-below]  
[-low]  
[-high]  
[-nworst_endpoints pin_count]  
[-max_sources_per_endpoint pin_count]  
[-significant_digits digits]  
[-slack_type slack_type]  
[-verbose]  
[-nosplit]  
[object_list]
```

list *object_list*

ARGUMENTS

-above

Performs the reporting only above the rails. If this option is combined with **-low**, it reports for the noise bumps above the low rail. If it is combined with **-high**, it reports the noise bumps above the high rail. Otherwise, it reports the noise bumps above the high rail and above the low rail.

-below

Performs the reporting only below the rails. If this option is combined with **-low**, it reports for the noise bumps below the low rail. If it is combined with **-high**, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps below the high rail and below the low rail.

-low

Performs the reporting only for the low rail. If this option is combined with **-above**, it reports the noise bumps above the low rail. If it is combined with **-below**, it reports the noise bumps below the low rail. Otherwise, it reports the noise bumps for both below and above the low rail.

-high

Performs the reporting only for the high rail. If this option is combined with **-above**, it reports the noise bumps above the high rail. If it is combined with **-below**, it reports the noise bumps below the high rail. Otherwise, it reports the noise bumps for both below and above the high rail.

-nworst_endpoints *pin_count*

Specifies the number of endpoint pins to be reported. Any number greater than 1 is accepted; the default value is 1.

-max_sources_per_endpoint *pin_count*

Specifies the number of violation source pins per endpoint to be reported.

Any number greater than 1 is accepted; the default value is 1.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-slack_type *slack_type*

Specifies the type of slack to be used. Valid values are *area*, *height*, and *area_percent*. A *slack_type* of *area* reports slack as the voltage margin multiplied by the noise bump width. The voltage margin is defined by the noise bump height and noise immunity curves or DC noise margin. This setting is the default. A *slack_type* of *height* reports noise slack as the voltage margin. A *slack_type* of *area_percent* reports noise slack as the percentage of the noise constraint area. The noise constraint area is computed by multiplying the noise height constraint by the noise bump width. The default is *area*.

-verbose

Shows details about pins in fan-in cone of the logic from violation sources to endpoint. Pins are ordered from the endpoint to sources and leveled by the distance from the endpoint. The worst slacks are shown on the top.

-nosplit

If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

object_list

Specifies the load pins for which the noise reporting is performed. If no pin is specified, reporting is performed on the entire design.

DESCRIPTION

Provides a report of noise violation source information for failing noise endpoints of the current design.

A noise endpoint is one of the followings: - Data pin, clock pin, or asynchronous pin of flip-flops - Input pin of level sensitive latch - Output port - Load pin of gates where noise exceeds the threshold

A failing endpoint is an endpoint with negative slack value. A violation source is the source of violation that causes a failure in the endpoint.

This command reports the violation sources of the specified or the *nworst* endpoints.

By default, the non verbose report is generated which shows the total crosstalk noise bump height and width as well as slack on the load pins.

EXAMPLES

The following example generates a report for the worst violation source of the worst endpoint above the low rail in the current design.

```
pt_shell> report_noise_violation_sources -above -low
```

The following example generates a report for the worst violation source of five worst endpoints above the low rail in the current design.

```
pt_shell> report_noise_violation_sources -above -low -nworst_pins 5
```

The following example generates a report for the worst violation source above the low rail and also the worst violation source below the low rail in the current design.

```
pt_shell> report_noise -low
```

The following example generates a report for the worst violation source for each of the four cases of above the low rail, below the low rail, above the high rail, and below the high rail in the current design.

```
pt_shell> report_noise -low -high
```

SEE ALSO

```
update_noise (2); set_noise_parameters (2); report_noise (2);  
report_default_significant_digits (3);
```

report_path_group

Reports path_group information.

SYNTAX

```
string report_path_group [-nosplit] [path_group_names]  
list  path_group_names
```

ARGUMENTS

-nosplit

Does not split lines if column overflows.

path_group_names

Indicates that the path group information for the *path_group_names* specified be displayed.

DESCRIPTION

Produces a report showing information about path groups in the **current_design**. The report includes path groups automatically created by **create_clock** and groups manually created with the **group_path** command. Path groups are used to affect the calculation of Maximum Delay cost during optimization. The cost of each group is displayed using **report_constraint**.

EXAMPLES

The following command displays all the path groups in the current design.

```
pt_shell> report_path_group
```

```
*****  
Report : path_group  
Design : test  
Version: A-2007.12-DEV  
Date   : Sun Jul 29 12:03:16 2007  
*****
```

Path_Group	Weight	From	Through	To
****default**	1.00	-	-	-
C1	1.00	*	*	C1
C2	1.00	*	*	C2
CLK	1.00	*	*	CLK

SEE ALSO

`create_clock` (2), `current_design` (2), `group_path` (2), `report_constraint` (2),
`reset_design` (2).

report_port

Displays port information within the design.

SYNTAX

```
string report_port [-verbose]  
[-design_rule]  
[-drive]  
[-input_delay]  
[-output_delay]  
[-wire_load]  
[-nosplit]  
[port_names]
```

```
list port_names
```

ARGUMENTS

-verbose

Indicates that the port report includes all port information. By default, only a summary section is displayed that lists all ports and their direction.

-design_rule

Reports only port design rule information, including maxCap, manLoad, and maxFanout.

-drive

Reports only drive resistance, input transition time, and driving cell information for only input and inout ports.

-input_delay

Reports only the port input delay information you set.

-output_delay

Reports only the port output delay information you set.

-wire_load

Reports only the port wire load information.

-nosplit

Prevents line splitting if column overflows. Most design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

port_names

Displays information on these ports in the current design. Each element in this list is either a collection of ports or a pattern matching the port names.

DESCRIPTION

Displays information about ports in the current design. By default, the command produces a brief report including all ports in the design.

The input and output delay information lists the minimum rise, minimum fall, maximum rise, maximum fall delays, and the clock to which the delay is relative. If "(f)" follows the clock name, the delay is relative to the falling edge of the clock; otherwise the delay is relative to the rising edge. If "(l)" follows the clock name, input delay is considered to be coming from a level-sensitive latch, or output delay is considered to be going to a level-sensitive latch. By default, the delay is relative to a rising edge-triggered device.

Some information, such as whether the port is in an ideal network or not, is displayed after a timing update (as a result of manually executing an `update_timing` function or experiencing an implicit timing update as a result of executing other commands). This behavior is intended to help you to obtain information and statistics about ports without incurring the cost of a complete timing update.

EXAMPLES

This example shows the default port report.

```
pt_shell> report_port
```

```
*****
Report : port
Design : middle
Version: Y-2006.06
Date   : Wed Mar  8 07:26:43 2006
*****
```

Attributes:

I - ideal network

Port	Dir	Pin Cap	Wire Cap	Attributes
CLOCK	in	0.0000	0.0000	
OPER	in	0.0000	0.0000	
in0	in	0.0000	0.0000	
in1	in	0.0000	0.0000	
in2	in	0.0000	0.0000	
out_1	out	0.0000	0.0000	
out_2	out	0.0000	0.0000	
out_3	out	0.0000	0.0000	
out_4	out	0.0000	0.0000	
p_in	in	0.0000	0.0000	
p_out	out	0.0000	0.0000	

1

SEE ALSO

`report_design` (2), `report_hierarchy` (2), `report_net` (2), `report_cell` (2),
`report_reference` (2).

report_power

Generate power reports.

SYNTAX

```
int report_power
[-net_power]
[-cell_power]
[-leaf]
[-include_boundary_nets]
[-include_estimated_clock_network]
[-sort_by sort_method]
[-nworst number]
[-power_greater_than threshold]
[-hierarchy]
[-levels level]
[-clocks clock_list]
[-groups group_list]
[object_list]
[-verbose]
[-nosplit]
[-derate]
```

```
int number
int level
string sort_method
float threshold
list clock_list
list group_list
list object_list
float quantile
```

ARGUMENTS

-net_power
Indicates to generate net-based power report.

-cell_power
Indicates to generate cell-based power report.

-leaf
Indicates that the power report should traverse the hierarchy and report nets or cells at lower-levels (as if the design's hierarchy were flat). The default is to report objects at only the current level of hierarchy. When used in combination with the **-levels** option, the lower levels are traversed only to the specified depth.

-include_boundary_nets
Indicates that the switching power of primary input nets is to be counted when generating the power report; the default is to exclude boundary input nets.

`-include_estimated_clock_network`
 Indicates that the clock network power estimated by `estimate_clock_network_power` is to be included in the power report.

`-sort_by sort_method`
 Specifies the sorting mode for the net or cell order in the power report.

`-nworst number`
 Indicates that the report is to be filtered so that it displays only the top-most *number* of nets or cells sorted by a certain type of `sort_method`.

`-power_greater_than threshold`
 Indicates that only nets or cells with total power value equal to or greater than *threshold* are to be reported.

`-hierarchy`
 Indicates to generate hierarchy-based power report.

`-levels level`
 Specifies the number of levels of hierarchies to be displayed in the hierarchy-based power report. Also specifies the number of levels to traverse for the net and cell based report when the **-leafP option is used**.

`-clocks clock_list`
 Select only nets and/or cells that belong to the clock domains specified by `clock_list` and generate report for the selected objects.

`-groups group_list`
 Select only nets and/or cells that belong to the power groups specified by `group_list` and generate report for the selected objects.

`object_list`
 Specifies a list of cells and/or nets to be displayed in the cell-based and/or net-based power report.

`-verbose`
 Indicates to report some verbose information, mainly in the header of the report, such as operating conditions, wire load models used to calculate power, and power unit information, etc.

`-nosplit`
 Indicates to prevent line-splitting or section-breaking. By default, if the information for a given field exceeds its fixed column width, the next field begins on a new line, starting in the correct column (line-splitting). Also by default, if the information for one line exceeds the 80 character limit, the report is broken into 2 sections, each containing part of the information (section-breaking).

`-derate`
 Indicates that effective power derate factors will be included in the cell reports. It does not affect other types of reports.

DESCRIPTION

This command is to generate a power report. It can be used in either non-distributed power analysis or distributed power analysis. In the distributed power analysis, the command in the master has no any option and only generates a merged power report for top design. The following description is for the normal power analysis (non-distributed power analysis or usage in slaves of distributed power analysis).

Generates power reports for the design or a specific hierarchy. The specific hierarchy is determined by the `current_instance` command. Before generating the report, power information is updated if necessary.

Four types of power reports can be generated: summary report, cell-based report, net-based report, and hierarchy-based report.

Summary Power Report

When none of the `-cell_power`, `-net_power`, and `-hierarchy` options are specified, a summary power report will be generated. The summary power report displays internal, leakage, switching and total power, as well as peak power, peak time, glitching power, and X transition power, if available, for the current instance, or the current design if the current instance is the top hierarchy of the current design. The summary power report also reports power for each power group. By default all instances are placed in only 1 power group; so the sum of the predefined groups power equals the total power consumption of the design.

There are 7 predefined power groups:

- * `io_pad`: cells defined as part of the `pad_cell` group in the library.
- * `memory`: cells defined as part of the `memory` group in the library.
- * `black_box`: cells with no functional description in the library.
- * `clock_network`: cells in the `clock_network` excluding `io_pad` cells.
- * `register`: latches and flip flops driven by the clock network excluding `io_pads` and `black_boxes`.
- * `combinational`: non-sequential cells with a functional description.
- * `sequential`: latches and flip flops clocked by signals other than those in the clock network.

See section `Power Group Power Reporting`.

If the variable `"power_report_leakage_breakdowns"` is set to `TRUE`, the total leakage breakdowns will also be printed out.

Cell-based Power Report

When option `-cell_power` is specified, cell-based power is to be reported. The cell-based power report has two sections. Section one displays cell internal power, cell leakage power, switching power of nets driven by the cell, total power, and percentage of total power compared with the overall power of all cells displayed in the report. Section two reports peak power, peak time, glitching power and X transition power. Section two only shows up when there are peak power, glitching power, and/or X transition power. If option `-nosplit` is specified, the two section will be merged together. Peak power for a cell is only available if the waveform is created for this cell (See man page of `update_power` command).

By default, only cells (including hierarchical cells) in current hierarchy are displayed. The power information displayed for a hierarchical cell is the total effect of all leaf cells the hierarchical cell contains. If `object_list` is specified, only cells in the `object_list` are reported. If option `-leaf` is specified, the report would go through hierarchical cells and report all leaf

cells under current instance.

If the variable "power_report_leakage_breakdowns" is set to TRUE, an additional section of leakage breakdowns will also be printed out.

If option -derate is specified along with option -cell_power, effective power derate factors will be included in the power reports. The effective power derate factors are the ratio of derated power to underated power of a cell. Power components with zero power derate factors are excluded from calculating effective power derate factors. In such a case, an attribute 'z' will be added to the corresponding cell.

Net-based Power Report

When option -net_power is specified, net-based power is to be reported. The net-based power report displays Vdd, total net load, static probability, toggle rate, and net switching power.

By default, only nets in the current hierarchy are displayed. If option -leaf is specified, the report would traverse down all lower hierarchies and report all nets found in each hierarchy. If object_list is specified, only nets in the object_list are reported. If a net has more than one net segment in different hierarchies, only the net segment in the top most hierarchy is listed in the report.

Hierarchy-based Power Report

When option -hierarchy is specified, hierarchy-based power is to be reported. The report would be in a hierarchical format, with power information on a block-by-block basis. The hierarchy is shown through indentations. The -levels option specifies the number of levels of hierarchy to be displayed. Hierarchy levels deeper than the ones specified in this option will not be shown in the report. If option -leaf is specified, the leaf cell reference names are to be reported.

For each hierarchical block, the switching, internal, and leakage power are reported, as well as the total power and the percentage of total power compared with the overall power of the top hierarchy. The top hierarchy is determined by current_instance. Hierarchy-based report also has a second section to display peak power, peak time, glitching power, and X transition power, when necessary. Similarly, if option -nosplit is specified, the two section will be merged together. Peak power for the hierarchy is only available if the waveform is created for this hierarchy (See man page of **update_power** command).

Negative Power Reporting

When option -power_greater_than is specified with an appropriate negative threshold for -cell_power option, leaf cells with negative power are reported. The default value for the threshold is zero.

Sorting and Filtering

The cell/net list in the cell/net-based power report can be sorted by a certain sort mode specified by the option -sort_by. The available sorting modes for the -net_power or -cell_power options are listed below:

-net_power option	-cell_power option
-----	-----
name	name
net_static_probability	cell_internal_power

net_switching_power	cell_leakage_power
net_toggle_rate	dynamic_power
total_net_load	total_power
total_power	

If both the `-net_power` and `-cell_power` options are specified and a sorting mode is explicitly selected, the selected sorting mode is used for both the cell and nets reports. Therefore, you must select a sorting mode that applies to both the `-net_power` and `-cell_power` options.

When reporting with the `-hierarchical` option, the same sorting modes are available as the `-cell_power` option. The report is first sorted by hierarchy, and secondly by the requested sorting method, preserving the hierarchical structure of the report.

If the sorting mode is not explicitly set, a default is chosen based on the mode of the `report_power` command:

Mode	Implicit default
-----	-----
<code>-net_power</code>	net_switching_power
<code>-cell_power</code>	cell_internal_power
<code>-net_power -cell_power</code>	dynamic_power

The sorted cell/net list in the cell/net-based power report can be filtered to display only the top most number of cells/nets specified by option `-nworst`.

Clock Domain Power Reporting

When `clock_list` is specified by the option `-clocks`, power report is generated for the specified clock domains. Only cells/nets in the specified clock domains are considered for reporting.

The cells/nets in the clock tree network of clock CLK are taken as belonging to clock domain CLK. The cells/nets in a timing path launched by clock CLK are regarded as belonging to clock domain CLK. The cells/nets in a timing path with no launch clock but captured by clock CLK are regarded as belonging to clock domain CLK as well. If a cell/net belongs to multiple clock domains, it is forced to belong to the clock domain with the fastest clock. If a cell/net belongs to no clock domains, it is forced to belong to the fastest clock domain in the design.

The power for the design or a hierarchy is no longer the total effect of all leaf cells in the design or hierarchy, but instead, the total effect of all leaf cells that are in the design or hierarchy AND belong to the specified clock domains.

Clock domain power reporting supports all of the four types of power report.

Power Group Power Reporting

When `group_list` is specified by the option `-groups`, power report is generated for the specified power groups. Only cells/nets in the specified power groups are considered for reporting.

Power groups can be created by `create_power_group` command. The tool has also several

predefined power groups. See man page of `create_power_group` for details.

The power for the design or a hierarchy is no longer the total effect of all leaf cells in the design or hierarchy, but instead, the total effect of all leaf cells that are in the design or hierarchy AND belong to the specified power groups.

Power group power reporting supports all of the four types of power report. In the summary power report, if `-groups` is not specified, all the power groups are reported. For each power group, the cumulative switching, internal, and leakage power of the power group are reported, as well as the total power and the percentage of the total power compared with the overall power of the top hierarchy. The top hierarchy is determined by `current_instance`. This power group report also has a second section to display peak power, peak time, glitching power, and X transition power, when necessary. Similarly, if option `-nosplit` is specified, the two section will be merged together. Peak power for the power group is only available if the waveform is created for this power group (See man page of **`update_power`** command).

Clock Tree Power Reporting

Clock tree power and register (driven by the clock tree) power can be reported using the `report_power` command by exploiting the power group power reporting. There are two predefined power groups: `clock_network` and `register`. The `clock_network` power group contains all cells in the clock tree network. The `register` power group contains all registers driven by the clock tree network.

Two variables can affect clock tree power reporting:

`power_clock_network_include_clock_gating_network`
`power_clock_network_include_register_clock_pin_power`

Refer to their man pages for details.

The clock tree power reporting supports all of the four types of power report. By default, the summary power report displays power of the predefined `clock_network` and `register` power groups. An attribute label "i" is used to indicate whether the `clock_network` or `register` power group includes the register clock pin power. In the cell-based power report, an attribute label "c" is used to indicate that only clock pin internal power is displayed for the register cell; an attribute label "d" is used to indicate that the power displayed for the register cell does not include the register clock pin internal power.

The clock tree power for certain clocks (if not all clocks) can be reported by combining the options `-groups` and `-clocks`.

Estimated Clock Tree Power Reporting

Clock tree power and register (driven by the clock tree) power estimated using `estimate_clock_network_power` command can be reported using the `-include_estimated_clock_network` option. Estimated clock tree power can only be reported in the Summary Report, therefore `-cell_power`, `-net_power`, etc cannot be specified together with the `-include_estimated_clock_network` option.

The power of the existing clock tree network, if any, is subtracted from the total

power, and the power of the estimated clock tree power is added. If the estimated clock tree power include register power (by `estimate_clock_network_power -include_registers`), then the power of the existing registers is subtracted from the total power, and the power of the estimated register power is added. Accordingly, the power for the power groups "clock_network" and "register" should be affected. These power groups do not include estimated clock network power. However, the power for the existing clock network is subtracted.

A new section in the Summary Power Report shows the estimated clock network power for each clock that has been estimated. The command `estimate_clock_network_power` must be used before `report_power` command. Multiple `estimate_clock_network_power` commands can be used, but only the power from the latest command which estimate clock tree power for a certain clock is reported by `report_power`.

The option `-clocks` can be specified together with `-include_estimated_clock_network` option. In this case, only the clock tree power estimated for the specified clocks is included in the power report.

The option `-groups` can be specified together with `-include_estimated_clock_network` option. In this case, only when power groups "clock_network" and "register" are specified, the estimated clock network power is to be reported.

Low-Level Hierarchy Power Reporting

Power reports can be generated not only for the top design, but also for a lower-level hierarchy. The hierarchy can be determined by `current_instance`. The power reports for top design is a special case, when current instance is the top level hierarchy of the design. If current instance is not the top design, only the cells or nets under the current instance are targeted for power reporting.

By combination of `-clocks`, `-groups` options and `current_instance` command, more flexibilities are given to generate different power reports per clock domain, per power group, and per hierarchy.

Leakage Variation Reporting

The leakage variation feature is available for user that have both a PrimeTime-PX license and a PrimeTime-VX license.

When the leakage variation feature is used, only the leakage variation report can be generated. The leakage variation report describes the average leakage of the design, as well as the standard deviation and the 99th percentile (quantile) of the leakage. By default, only the leakage for the full design is reported. In order to report on additional hierarchical blocks, the blocks must be specified in advance of leakage variation analysis using the command **`set_power_analysis_options`** (or **`power_monitor_cells`** in the backward compatibility mode). For more information on the leakage variation feature, see the man page for the variable **`power_enable_leakage_variation_analysis`**.

BACKWARD COMPATIBILITY MODE

In the 2008.12 release, some options for **`report_power`** were deprecated and to

set_power_analysis_options. For this release, it is possible to use the deprecated options by setting the variable **power_ui_backward_compatibility** to **true**.

The following options are only available in this backward compatibility mode:

```
[-leakage_only]
[-no_propagation]
[-variation_quantile quantile]
```

float *quantile*

-leakage_only

Indicates to only report leakage power. If **update_power** has not been run, then leakage power analysis will be run. Dynamic power analysis will not be performed, potentially saving runtime.

-no_propagation

Indicates not to perform switching activity propagation. This option can only be used if the **-leakage_only** option is also used. Nets that do not have switching activity annotated will be assumed to have a static probability of 0.5.

-variation_quantile *quantile*

Indicates that the leakage variation report should report the specified leakage quantile in the quantile column of the report. This option can only be used if the variable **power_enable_leakage_variation_analysis** is set to TRUE.

The **-sort_by** method *total_power* is not available in backward compatibility mode.

EXAMPLES

The following example shows a verbose **report_power** summary report.

```
pt_shell> report_power -verbose
```

```
*****
Report : power
        -verbose
Design : testcase
Version: v3.2a
Date   : Sun Jun 19 15:45:24 1994
*****
```

Library(s) Used:

```
power_lib (File: /remote/libraries/power_lib.db)
```

Operating Conditions:

Wire Loading Model Mode: enclosed

Cell	Design	Wire Loading Model	Library
	testcase	0.5K_TLM	power_lib.db
sub	submodule	0.5K_TLM	power_lib.db

Power-specific unit information :

Voltage Units = 1 V
Capacitance Units = 1 pf
Time Units = 1 ns
Dynamic Power Units = 1 W
Leakage Power Units = 1 W

Attributes

i - Including register clock pin internal power
u - User defined power group

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attr
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	1.813e-04	4.199e-05	4.192e-10	2.233e-04	(70.03%)	i
register	8.442e-05	1.114e-05	9.208e-09	9.557e-05	(29.97%)	
combinational	0.0000	0.0000	0.0000	0.0000	(0.00%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	

Net Switching Power = 5.313e-05 (16.66%)
Cell Internal Power = 2.657e-04 (83.33%)
Cell Leakage Power = 9.627e-09 (0.00%)

Total Power = 3.188e-04 (100.00%)

1

The following example shows a net-based power report sorted by net_switching_power and filtered to display only the 5 nets with highest switching power.

pt_shell> **report_power -net_power -leaf -nworst 5**

Report : power
-net_power
-nworst 5
-leaf
-sort_by net_switching_power
-power_greater_than 0
Design : testcase

Date : Mon Mar 20 12:05:44 2006

Attributes

- a - Switching activity information annotated on net
- p - Propagated switching activity information on net
- d - Default switching activity used on net
- u - Net switching activity uninitialized
- m - Net is driven by multiple pins

Net	Vdd	Total Net Load	Static Prob.	Toggle Rate	Switching Power	Attrs

net36	1.80	0.026	0.248	0.1985	8.397e-06	p
net42	1.80	0.026	0.248	0.1985	8.397e-06	p
net48	1.80	0.026	0.248	0.1985	8.397e-06	p
net54	1.80	0.026	0.248	0.1985	8.397e-06	p
sub/net20	1.80	0.026	0.248	0.1985	8.397e-06	p

Total (5 nets)					4.199e-05 Watt	

1

The following example shows a cell-based power report for clock_network power group (clock tree power).

pt_shell> **report_power -cell_power -groups clock_network**

Report : power

-cell_power
-sort_by cell_internal_power
-power_greater_than 0
-groups clock_network

Design : testcase

Date : Mon Mar 20 12:09:46 2006

Attributes

- a - Annotated internal & leakage power
- b - Black-box (unresolved) cell
- c - Clock pin internal power only
- d - Does not include clock pin internal power
- h - Hierarchical cell

Cell	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs

report_power

736


```

sub          3.626e-05 8.397e-06 8.384e-11 4.466e-05 (20.00%) h
clk_out1_reg 1.228e-05 8.397e-06 8.384e-11 2.068e-05 ( 9.26%) h
clk_temp1_reg 1.228e-05 8.397e-06 8.384e-11 2.068e-05 ( 9.26%) h
temp1_reg_3_ 5.995e-06 0.0000 0.0000 5.995e-06 ( 2.68%) c
temp1_reg_0_ 5.995e-06 0.0000 0.0000 5.995e-06 ( 2.68%) c
-----
Totals      1.813e-04 4.199e-05 4.192e-10 2.233e-04 (100.0%)
1

```

The following example shows a hierarchy-based power report for the "sub" instance.

```
pt_shell> current_instance sub
```

```
pt_shell> report_power -hierarchy -levels 2
```

```
*****
```

```
Report : power
        -hierarchy
        -levels 2
```

```
Design : testcase/sub (submodule)
```

```
Date   : Mon Mar 20 12:14:32 2006
```

```
*****
```

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
sub (submodule)	9.98e-06	5.33e-05	1.93e-09	6.33e-05	100.0
clk_gate_out_reg (SNPS_CLOCK_GATE_HIGH_submodule)	9.14e-06	1.64e-05	2.41e-10	2.56e-05	40.4

```
1
```

The following example shows a report including estimated clock network power.

```
pt_shell> report_power -include_estimated_clock_network
```

```
*****
```

```
Report : power
```

```
Design : testcase
```

```
Version: v3.2a
```

```
Date   : Mon Mar 20 12:14:32 2006
```

```
*****
```

Attributes

```
-----
```

- i - Including register clock pin internal power
- u - User defined power group

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	

memory	0.0000	0.0000	0.0000	0.0000 (0.00%)
black_box	0.0000	0.0000	0.0000	0.0000 (0.00%)
clock_network	0.0000	0.0000	0.0000	0.0000 (0.00%)
register	8.442e-05	1.114e-05	9.208e-09	9.557e-05 (29.97%) i
combinational	0.0000	0.0000	0.0000	0.0000 (0.00%)
sequential	0.0000	0.0000	0.0000	0.0000 (0.00%)

Attributes

i - Including driven register power

Clock	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
clk	1.813e-04	4.199e-05	4.192e-10	2.233e-04		
Estimated Clock	1.813e-04	4.199e-05	4.192e-10	2.233e-04	(70.03%)	
Net Switching Power	= 5.313e-05	(16.66%)				
Cell Internal Power	= 2.657e-04	(83.33%)				
Cell Leakage Power	= 9.627e-09	(0.00%)				
Total Power	= 3.188e-04	(100.00%)				

1

The following example shows the leakage variation report.

pt_shell> **report_power**

```
*****
Report : Leakage Variation Report
Design : test_design
Version: B-2008.06-PRE_PROD-DEV
Date   : Sat Nov 17 17:58:46 2007
*****
```

Library(s) Used:

power_variations_lib (File: /remote/libraries/power_variations_lib.db)

Operating Conditions:
Wire Load Model Mode: unknown

<no wire load model is set>

Power-specific unit information :
Voltage Units = 1 V
Capacitance Units = 1 pF

report_power
738

Time Units = 1 ns
Dynamic Power Units = 1 W
Leakage Power Units = 1 W

Design	quantile	sensitiv	mean	stddev
test_design 1	1.21771e-06	0.416507	6.25407e-07	2.60486e-07

SEE ALSO

`update_power (2)`, `create_power_group (2)`,
`power_clock_network_include_clock_gating_network (3)`,
`power_clock_network_include_register_clock_pin_power (3)`,
`power_enable_leakage_variation_analysis (3)`.

report_power_analysis_options

Report the options for power analysis.

SYNTAX

```
int report_power_analysis_options
```

DESCRIPTION

The *report_power_analysis_options* command enables you to get the value set for the options used in power analysis. It reports both default and user settings.

The variable *collection_result_display_limit* will be used to control how many cells in *set_power_analysis_options -cell* will be displayed.

Command returns 1 if succeed, and returns 0 otherwise.

EXAMPLES

The following is an example from *report_power_analysis_options* command output:

```
pt_shell> report_power_analysis_options
```

```
*****
Report : Power Analysis Options
Design : mac
Version: B-2008.12
Date   : Mon Dec  1 14:59:26 2008
*****
```

```
Power analysis mode : time_based
```

Option Name	Value
include	top
waveform_format	fsdb
waveform_output	ptpx

1

SEE ALSO

```
power_analysis_mode(3),
collection_result_display_limit(3),
set_power_analysis_options(2),
update_power(2),
report_power(2).
```

report_power_calculation

Displays the actual calculation of internal power for a pin, leakage power for a cell or switching power for a net.

SYNTAX

```
int report_power_calculation object_list
[-state_condition state]
[-path_sources name_list]
[-rise | -fall]
[-verbose]
```

```
object_list object_list
string state
string name_list
```

ARGUMENTS

- object_list**
Specifies the objects for which the power calculation is reported. The type of power calculation depends on the object type. For a pin, internal power calculation is reported. For a cell, a leakage power calculation report is produced. Finally, a net results in a switching power calculation report.
- state_condition *state***
Specifies that the report should be restricted to tables or polynomials with matching state condition. This option can only be specified for objects of type pin or cell. Use **-state_condition default** to specify the default state condition. Use **-state_condition all** to specify that all states must be considered for reporting.
- path_sources *name_list***
Specifies the related input pin which causes the output pin transition. This option can only be specified for objects of type pin. The path information is used not only for picking the internal power table or polynomial, but also for choosing the input transition time for power calculation if the input transition time is one of the variables. Use **-path_sources default** to specify the default path source (without related pin). Use **-path_sources all** to specify all possible paths.
- rise**
Specifies that the internal power report should be limited to rise transition only. If not specified, both rise and fall transitions will be reported (equivalent to specifying both **-rise** and **-fall**).
- fall**
Specifies that the internal power report should be limited to fall transition only. If not specified, both rise and fall transitions will be reported (equivalent to specifying both **-rise** and **-fall**).
- verbose**
Specify this option to increase the amount of information that is reported

for cells or pins. For cells, the leakage power calculation report will be appended by an internal power calculation report for all pins and all possible states and path sources of the cells in the object list. For pins, all the possible states and paths will be printed (equivalent to **-state_condition all -path_sources all**).

DESCRIPTION

The **report_power_calculation** command provides detailed power calculation information for the specified pin, cell or net. You can use this information for debugging or verifying power data in a technology library. Before using this command to display details of internal or leakage power calculation, read the technology library file into the system with the library features attribute, **report_power_calculation** included in the library, which enables the **report_power_calculation** command for internal, leakage and net switching power. Otherwise, the built-in security mechanism terminates the command.

Command **report_power_calculation** is only enabled in averaged power analysis mode. For more information on analysis mode, please check out the man page for **power_analysis_mode**.

PrimeTime PX supports power net based power report feature for MV designs. You can use **set_current_power_net** command to set the power net of interest. Only the power dissipated on the specified power nets will be calculated and reported. As default (if not specified), all the power nets will be included in power analysis. Use **get_current_power_net** to show the current power net setting.

EXAMPLES

The following example shows the calculation of path dependent internal power from input pin B to output pin Y. The contributions for rise and fall internal power are shown separately.

```
pt_shell> report_power_calculation [get_pin U4/Y] -state default -path B
```

```
*****
Report : power_calculation
        U4/Y
        -state_condition default
        -path_sources B
Design  : rpc
Version: W-2004.12
Date    : Thu Sep  2 12:10:44 2004
*****
```

Library(s) Used:

example (File: /root/libraries/example.db)

Operating Conditions: typical Library: example
Wire Load Model Mode: segmented

Design	Wire Load Model	Library
rpc	a	example

Global Operating Voltage = 0.9

Library Power-specific unit information :

Voltage Unit : 1 V

Capacitance Unit : 1 pF

Time Unit : 1 ns

Dynamic Power Unit : 0.001 W (derived from V,C,T units)

Leakage Power Unit : 1e-09 W

Note: Unless specified, the numbers reported are in library units.

Pin Internal Power Calculation

cell: U4

pin: Y

path source: B

Path Dependent Rise Pin Internal Power = 2.35214e-05 W

pin internal power = internal energy * pin toggle rate

Rise internal energy per transition = 0.143423

library model: NLPM

table variables:

X = total_output_net_capacitance = 0.4

Y = input_net_transition = 0.3

relevant portion of lookup table:

	(X)	0.1050	(X)	0.3550	
(Y)	0.0500	(Z)	0.1310	(Z)	0.1410
(Y)	0.4510	(Z)	0.1320	(Z)	0.1420

$Z = A + B \cdot X + C \cdot Y + D \cdot X \cdot Y$

A = 0.1267 B = 0.0400

C = 0.0025 D = 0.0000

Z = 0.143423

(no scaling since the library has no internal power k-factors)

Path Dependent Rise Pin Toggle Rate = 0.164

Path Dependent Fall Pin Internal Power = 2.35214e-05 W

pin internal power = internal energy * pin toggle rate

Fall internal energy per transition = 0.143423

library model: NLPM

table variables:

X = total_output_net_capacitance = 0.4

Y = input_net_transition = 0.3

relevant portion of lookup table:

(X)	0.1050	(X)	0.3550

```
(Y)  0.0500      (Z)  0.1310      (Z)  0.1410
(Y)  0.4510      (Z)  0.1320      (Z)  0.1420
```

```
Z = A + B*X + C*Y + D*X*Y
A =  0.1267      B =  0.0400
C =  0.0025      D =  0.0000
```

```
Z = 0.143423
(no scaling since the library has no internal power k-factors)
```

```
Path Dependent Fall Pin Toggle Rate = 0.164
```

The following is an example of a state dependent leakage power calculation report.

```
pt_shell> report_power_calculation [get_cell U5] -state "A B"
```

```
*****
Report : power_calculation
      U5
      -state_condition A B
Design : rpc
Version: W-2004.12
Date   : Thu Sep  2 12:00:48 2004
*****
```

Library(s) Used:

```
example (File: /root/libraries/example.db)
```

```
Operating Conditions: typical  Library: example
Wire Load Model Mode: segmented
```

Design	Wire Load Model	Library
rpc	a	example

```
Global Operating Voltage = 0.9
Library Power-specific unit information :
  Voltage Unit : 1 V
  Capacitance Unit : 1 pF
  Time Unit : 1 ns
  Dynamic Power Unit : 0.001 W      (derived from V,C,T units)
  Leakage Power Unit : 1e-09 W
```

Note: Unless specified, the numbers reported are in library units.

```
=====
Cell Leakage Power Calculation
  cell: U5
  state condition: A B
```



```

State Dependent Leakage Power = 3.184e-12 W
cell leakage power = leakage power value * state probability

Leakage Power Value = 1.021e-07
library model: scalar
value = 1.021e-07
(no scaling since the library has no leakage power k-factors)

State Probability = 0.03120 (estimated)

```

The following shows how switching power calculation is performed.

```

pt_shell> report_power_calculation [get_net q]

```

```

*****
Report : power_calculation
      q
Design : rpc
Version: W-2004.12
Date   : Thu Sep  2 12:12:32 2004
*****

```

```

Library(s) Used:

example (File: /root/libraries/example.db)

```

```

Operating Conditions: typical  Library: example
Wire Load Model Mode: segmented

```

Design	Wire Load Model	Library
-----	-----	-----
rpc	a	example

```

Global Operating Voltage = 0.9
Library Power-specific unit information :
Voltage Unit : 1 V
Capacitance Unit : 1 pF
Time Unit : 1 ns
Dynamic Power Unit : 0.001 W (derived from V,C,T units)
Leakage Power Unit : 1e-09 W

```

Note: Unless specified, the numbers reported are in library units.

=====

```

Net Switching Power Calculation
net: q
driver: U4/Y

```

```

Switching power = 1.296e-04 W
net switching power = switching energy * net toggle rate

```

```
Switching Energy Per Transition = 0.162
  switching energy = 0.5 * capacitance * voltage ^ 2
  total net capacitance = 0.4
  voltage = 0.9
```

Net Toggle Rate = 0.8 (user annotated)

SEE ALSO

```
report_lib (2), update_power (2), report_power (2), read_saif (2),  
set_switching_activity (2), power_analysis_mode (3), set_current_power_net(2).
```

report_power_derate

Reports power derate factors annotated on the current design.

SYNTAX

```
int report_power_derate
    [-include_inherited]
    [-significant_digits digits]
    [-nosplit]
    object_list

    list    object_list
    int     digits
```

ARGUMENTS

-include_inherited

Indicates that the power derating factors reported on each object should also include those set by other objects in the design.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for values in the report. Allowed values are 0-12; the default is determined by the `report_default_significant_digits` variable. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The `-nosplit` option prevents line-splitting and facilitates writing software to extract information from the report output.

object_list

Specifies current design or a list of cells or library cells from which the power derating factors are reported.

DESCRIPTION

This command is used to report the derate factors either on the design, specific cells or library cells contained in the design. If this command is called without any option, a separate report will be invoked for the power derating factors stored on the current design (referred to as global power derating factors) and also for each cell or library cell that has power derating factors set on it.

If this command is called with the boolean option `-include_inherited`, each row containing an inherited derating factor will contain a column entry indicating the object from which it was inherited. If the command is called with the `object_list` specified then derate reports will only be issued for the instances specified in the object list.

For significant digits used in power derate report, the *-significant_digits* option can be used. If the option is not specified, the value of variable *report_default_significant_digits* will be applied. As default, the number of significant digits used to display values is 2.

EXAMPLES

In the following example power derating factors have only been set globally (on the design), therefore only global derate factors will be reported.

```
pt_shell> set_power_derate -switching 0.92
pt_shell> set_power_derate -internal 1.15
pt_shell> set_power_derate -gate_leakage 0.75
pt_shell> set_power_derate -subthreshold 1.26
pt_shell> report_power_derate
```

```
pt_shell> report_power_derate
*****
Report : power derate
Design : top
Version: D-2009.12
Date   : Mon Sep 25 11:08:59 2009
*****
```

Subthreshold Leakage	Gate Object Name Leakage	Object Type	Switching	Internal	Leakage
1.26	0.75	top design	0.92	1.15	--
1					

In the following example power derating factors have been set on the design named 'top'. More restrictive derates have been set on the hierarchical block name 'H1'. The derates that apply to the hierarchical cell named 'H1/U1' are reported.

```
pt_shell> set_power_derate 0.95
pt_shell> set_power_derate -switching 1.06
pt_shell> set_power_derate -internal -gate_leakage 0.82 [get_cell H1]
pt_shell> set_power_derate -subtrehsold_leakage 1.12 [get_cell H1/U1]
pt_shell> report_power_derate -include_inherited [get_cells H1/U1]
```

```
*****
Report : power derate
        -include_inherited
Design : top
Version: D-2009.12
Date   : Mon Sep 25 11:13:40 2009
*****
```

Subthreshold	Gate				
Object Name	Object Type	Switching	Internal	Leakage	

report_power_derate

Leakage	Leakage				

	H1/U1	cell (leaf)	1.06	0.82	--
	1.12	0.82			
	Inherited		top	H1	H1
--	H1				
1					

SEE ALSO

set_power_derate (2),
reset_power_derate (2),

report_power_domain

Report the specified power domain.

SYNTAX for UPF

```
int report_power_domain
```

```
[domain_list]
```

Data Types

```
domain_list          list
```

ARGUMENTS

```
domain_list
    Specify the list of power domains to be reported. The names in the list should
    be a simple (non-hierarchical) name.
    If there is no such a power domain with the same name in the current scope,
    the command fails.
    If this option is not specified, all power_domain in the specified scope will
    be reported.
```

DESCRIPTION

The *report_power_domain* command enables you to get the detail information of an existed power domain in current scope. The information of power domain includes: its full name, its scope, its element, its primary power/ground net.

Command returns 1 if succeed, and returns 0 otherwise.

EXAMPLES

```
pt_shell> report_power_domain
*****
Report : power domains
Design : top
Version: A-2007.12-Beta2-DEV
Date   : Thu Sep 27 18:59:04 2007
*****
```

```
Total of 2 power domains defined for design 'top'.
```

```
Power Domain : top_domain
Scope : <top level>
Elements : <top level>
```

```
Connections :      -- Power --      -- Ground --
Primary       <none>                <none>
```

```
Power Domain : PD0
Scope : <top level>
Elements : PD0_INST
```

```
Connections :      -- Power --      -- Ground --
Primary      <none>      <none>
```

1

```
pt_shell> report_power_domain [get_power_domain top_domain]
```

```
*****
```

```
Report : power domains
```

```
Design : top
```

```
Version: A-2007.12-Beta2-DEV
```

```
Date   : Thu Sep 27 18:59:04 2007
```

```
*****
```

```
Power Domain : top_domain
Scope : <top level>
Elements : <top level>
```

```
Connections :      -- Power --      -- Ground --
Primary      <none>      <none>
```

1

```
pt_shell> report_power_domain [get_power_domain -regexp {t.*}]
```

```
*****
```

```
Report : power domains
```

```
Design : top
```

```
Version: A-2007.12-Beta2-DEV
```

```
Date   : Thu Sep 27 18:59:04 2007
```

```
*****
```

```
Power Domain : top_domain
Scope : <top level>
Elements : <top level>
```

```
Connections :      -- Power --      -- Ground --
Primary      <none>      <none>
```

1

SEE ALSO

```
create_power_domain(2),
get_power_domains(2).
```

report_power_groups

Report the existing power groups.

SYNTAX

```
string report_power_groups  
group_names  
[-nosplit]
```

list *group_names*

ARGUMENTS

group_names
Specifies the power group names that are to be reported.

-nosplit
Indicates to prevent line splitting if column overflows.

DESCRIPTION

The **report_power_groups** command reports the power groups that has been created in the earlier stage, including the predefined power groups.

EXAMPLES

In the following example, all the existing power groups are reported.

```
pt_shell> create_power_group -name clock_tree [get_clock_network_objects -  
type cell]
```

```
1  
pt_shell> report_power_groups  
i*****  
Report : report_power_groups  
        -nosplit  
Design : testcase  
Version: Y-2006.06-Beta1-DEV  
Date   : Mon Mar 13 16:28:48 2006  
*****
```

Power Group	Size	Attribute
black_box	0	Default
clock_network	5	Default
combinational	0	Default
io_pad	0	Default
memory	0	Default
register	25	Default
sequential	0	Default
clock_tree	5	User defined

SEE ALSO

`create_power_group` (2), `remove_power_groups` (2), `get_power_group_objects` (2).

report_power_net_info

Reports the power net info for the current design.

SYNTAX

```
int report_power_net_info
```

ARGUMENT

The **report_power_net_info** command has no arguments.

DESCRIPTION

The **report_power_net_info** command reports the power net info for the current design.

EXAMPLE

The following example uses the command to report the power net info.

```
pt_shell> create_power_domain T
1
pt_shell> create_power_net_info T_VDD -power
1
pt_shell> create_power_net_info T_VSS -gnd
1
pt_shell> create_power_domain T
1
pt_shell> connect_power_domain T \
        -primary_power_net T_VDD \
        -primary_ground_net T_VSS
1
prompt> report_power_net_info

*****
Report : power_net
Design : top
Version: Z-2007.06-DEV
Date   : Wed May 16 15:18:42 2007
*****

Total of 2 power nets defined.

Power Net 'T_VDD' (power)
-----
Primary Power Hookups:          T
Internal Power Hookups:         T
-----

Power Net 'T_VSS' (ground)
-----
```

report_power_net_info

Primary Ground Hookups:	T
Internal Ground Hookups:	T

1

SEE ALSO

create_power_net_info (2)
connect_power_domain (2)

report_power_network

Reports connectivity in virtual power network formed by UPF supply nets, ports, and PG pins.

SYNTAX

```
string report_power_network
[-nets nets]
```

Data Types

```
nets      string
```

ARGUMENTS

```
-nets      Report connectivity of a subset of nets.
```

DESCRIPTION

Reports connectivity of supply nets, ports and explicitly connected power and ground pins.

EXAMPLES

```
prompt> report_power_network -nets VDDI

Supply Net: VDDI
Connections:
  Name :                Object Type                Domain
-----
VDDI                Supply Port                TOP
InstDecode/LS_u1/VDDL PG_power_pin                INST
```

The first example queries the current UPF version. The second example shows all UPF commands (i.e., commands that may be affected by a specific UPF standard).

SEE ALSO

```
create_supply_net(2),
report_supply_net(2).
```

report_power_pin_info

Reports the power pin info for technology library cells or leaf cells.

SYNTAX

```
int report_power_pin_info
    object_list

list object_list
```

ARGUMENT

object_list
Specify a list of technology library cells or a list of leaf cells or ports.

DESCRIPTION

The **report_power_pin_info** command reports the power pin information for technology library cells or leaf level cells in the current design.

When a list of library cells is specified, the name, the types and the voltage specification of the power pins are reported. All types of PG pins including power, ground and bias pins will be shown.

When a list of leaf cells are specified in the *object_list*, the power net info that are connected to the power pins are also report. Hierarchical cells will be ignored by this command.

EXAMPLES

The following example uses the command to report the power pin information.

```
pt_shell> report_power_pin_info [get_cells -hier]
```

```
*****
Report : power pin info
Design : top
Version: Z-2007.06-DEV
Date   : Wed May 16 15:18:42 2007
*****
```

Note: Power connections marked by (*) are exceptional

Cell	Power Pin Name	Type	Voltage	Power
Net Connected				
PD0_INST/I0	PWR	primary_power	1.0000	
PD0_INST/I0	GND	primary_ground	0.0000	
PD0_INST/I1	PWR	primary_power	1.0000	

PD0_INST/I1	GND	primary_ground	0.0000
I0	PWR	primary_power	1.0000
I0	GND	primary_ground	0.0000
I1	PWR	primary_power	1.0000
I1	GND	primary_ground	0.0000

pt_shell> **report_power_pin_info [get_lib_cells -of [get_cells -hier]]**

Report : power pin info

Design : top

Version: Z-2007.06-DEV

Date : Wed May 16 15:18:42 2007

Library Cell	Power Pin Name	Type	Voltage
INVS2	PWR	primary_power	1.0000
INVS2	GND	primary_ground	0.0000
BUFS2	PWR	primary_power	1.0000
BUFS2	GND	primary_ground	0.0000
AND2X1	PWR	primary_power	1.0000
AND2X1	GND	primary_ground	0.0000

1

pt_shell> **connect_power_net_info I0 -power_pin_name PWR -power_net_name exp_VDD**

1

pt_shell> **connect_power_net_info PD0_INST/I0 -power_pin_name PWR -
power_net_name exp_VDD**

1

pt_shell> **report_power_pin_info [get_cells -hier]**

Report : power pin info

Design : top

Version: Z-2007.06-DEV

Date : Wed May 16 15:18:42 2007

Note: Power connections marked by (*) are exceptional

Cell	Power Pin Name	Type	Voltage	Power
PD0_INST/ I0	PWR	primary_power	1.0000	exp_VDD (*)
PD0_INST/I0	GND	primary_ground	0.0000	A_VSS
PD0_INST/I1	PWR	primary_power	1.0000	A_VDD
PD0_INST/I1	GND	primary_ground	0.0000	A_VSS
I0	PWR	primary_power	1.0000	exp_
VDD (*)				
I0	GND	primary_ground	0.0000	T_VSS
I1	PWR	primary_power	1.0000	T_VDD
I1	GND	primary_ground	0.0000	T_VSS

report_power_pin_info

758

1

```
pt_shell> report_power_pin_info [get_cells -hier]
```

```
*****
Report : power pin info
Design : top
Version: Z-2007.06-DEV
Date   : Wed May 16 15:18:42 2007
*****
```

Note: Power connections marked by (*) are exceptional

Cell	Power	Pin Name	Type	Voltage	Powe
r Net Connected					

PD0_INST/					
I0	PWR	primary_power	1.0000	exp_VDD	(*)
PD0_INST/I0	GND	primary_ground	0.0000	A_VSS	
PD0_INST/I1	PWR	primary_power	1.0000	A_VDD	
PD0_INST/I1	GND	primary_ground	0.0000	A_VSS	
I0	PWR	primary_power	1.0000	exp_	
VDD (*)					
I0	GND	primary_ground	0.0000	T_VSS	
I1	PWR	primary_power	1.0000	T_VDD	
I1	GND	primary_ground	0.0000	T_VSS	
1					

SEE ALSO

```
connect_power_domain (2)
connect_power_net_info (2)
```

report_power_rail_mapping

Report the current power rail mapping.

SYNTAX

```
report_power_rail_mapping
[-cells cell_list]
[-verbose]
[-nosplit]
```

object_list top_module_list

ARGUMENTS

-cells cell_list

Specifies the instance names or collections of instances. This only takes effect when option **-verbose** is used. The instance can be hierarchical or leaf instances. For a instance block, only the top instance needs to be specified. All the leaf instances inside such block will be reported. Without this option, the mapping will be reported for the whole design.

-verbose

Report detailed rail mapping for each individual leaf instance. You can choose the leaf instances to be reported by using option **-instance**. When this option is used, PrimeTime PX also checks potential problems in rail mapping done for the instances. The list of checks are:

1. Different library rails are mapped to same design rail in a multi-rail leaf cell
2. No rail specific power table is defined for a library cell with multiple power rails

-nosplit

Most of the information is listed in fixed-width columns. If the information for a given field exceeds its column's width, the next field begins on a new line, starting in the correct column. This option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

The **report_power_rail_mapping** command displays all the power rails defined in the library and those existing in the design. The library power rails are the **voltage_map** attributes defined in the library (or **power_rail** attributes defined in **power_supply** group). The design power rails are the rails created by command **create_power_rail_mapping**. If a library doesn't have rail information, **<default_rail>** will be shown for its library power rail. If the mapping does not exist, the default design power rail will be shown and labeled with **(default)**. During power simulation, those missing cases will be actually mapped to the default design power rail. Internally, there is a default rail associated with each library and with the whole design. If **-verbose** is used, PrimeTime PX reports the detailed rail mapping for each individual leaf instances and also shows possible rail mapping

problems for design.

EXAMPLES

The following example shows an example of `report_power_rail_mapping`.

```
pt_shell> report_power_rail_mapping -verbose -cells block1
```

SEE ALSO

`create_power_rail_mapping` (2).

report_power_switch

Report all power switches defined in the design.

SYNTAX

```
int report_power_switch
```

DESCRIPTION

The **report_power_switch** command enables you to get the detail information of all power_switches defined in the design. The information of power switch includes: its full name, the power domain where it is created in, its output supply port, its input supply port, its control port and its on_state statement.

Command returns 1 if succeed, and returns 0 otherwise.

EXAMPLES

The following example reports the information of all power_switches defined:

```
pt_shell> report_power_switch

Power Switch : SW1
-----
Power Domain : PD1
Output Supply Port : vout VN2
Input Supply Port : vin VN1
Control Port : ctrl_small ON1
Control Port : ctrl_large ON2
On State : state1 vin { ON1 & ON2 }
-----
1
```

SEE ALSO

```
create_power_switch(2)
```

report_pulse_clock_max_transition

Displays maximum transition computation at the input of pulse generator and pulse clock network.

SYNTAX

```
int report_pulse_clock_max_transition
    [-rise][-fall]
    [-transitive_fanout]
    [-all_violators]
    [-significant_digits digits]
    [-nosplit]
    [port_pin_list]
```

list *port_pin_list*

ARGUMENTS

-rise

Report slack for rise transition.

-fall

Report slack for fall transition.

-transitive_fanout

Report transition slack at the transitive fanout of pulse generator cells. If this option is not set, then transition slack is reported at the input of pulse generator cells.

-all_violators

Specifies that only the pins violating maximum transition constraint will be reported.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

port_pin_list

Specifies a list of pins or ports to report. By default, the report contains all pins with pulse clock maximum transition constraint.

DESCRIPTION

The **report_pulse_clock_max_transition** command displays the maximum transition computation of pins and ports. By default only the input pins of the pulse generators are reported. If rise or fall is not specified, then both rise and fall transition slack is reported. To enable pulse clock constraint checking, set the variable `timing_enable_pulse_clock_constraints` to true. This variable is set to true by default.

EXAMPLES

The following example generates a report of all transition slack computation at the input of pulse generators in the the current design.

```
pt_shell> report_pulse_clock_max_transition
*****
Report : pulse clock max transition
-rise
-fall
Design : test
*****

pulse_clock_max_transition_rise

Pin                Required
Transition          Transition          Slack
-----
p1/i                0.40                0.12                0.28 (MET)

pulse_clock_max_transition_fall

Pin                Required
Transition          Transition          Slack
-----
p1/i                0.40                0.12                0.28 (MET)
```

The following example generates a report of all transition slack computation in the pulse clock network in the current design.

```
pt_shell> report_pulse_clock_max_transition -transitive_fanout
*****
Report : pulse clock max transition
-rise
-fall
Design : test
*****

pulse_clock_max_transition_rise_transitive_fanout
```

Pin	Required Transition	Actual Transition	Slack	
ff2/cp	0.20	0.08	0.12	(MET)
ff1/cp	0.20	0.02	0.18	(MET)
u3/zn	0.30	0.08	0.22	(MET)
p1/z	0.30	0.02	0.28	(MET)
u3/i	0.30	0.02	0.28	(MET)

pulse_clock_max_transition_fall_transitive_fanout

Pin	Required Transition	Actual Transition	Slack	
ff2/cp	0.20	0.08	0.12	(MET)
ff1/cp	0.20	0.02	0.18	(MET)
u3/zn	0.30	0.08	0.22	(MET)
p1/z	0.30	0.02	0.28	(MET)
u3/i	0.30	0.02	0.28	(MET)

SEE ALSO

**set_pulse_clock_max_transition (2), remove_pulse_clock_max_transition (2),
report_constraint (2).**

report_pulse_clock_max_width

Displays maximum pulse width computation for the pulse clock network.

SYNTAX

```
int report_pulse_clock_max_width
    [-all_violators]
    [-significant_digits digits]
    [-nosplit]
    [-path_type format]
    [-transitive_fanout]
    [port_pin_list]
```

list *port_pin_list*

ARGUMENTS

-all_violators

Specifies that only the pins in the pulse clock network violating the maximum pulse width constraint will be reported.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

-path_type *format*

Specifies the format of the path report and how the clock path is displayed. Allowed values are: *summary* (the default), which generates a report with a column format that shows one line for each path and shows only the required pulse width, actual pulse width and slack; and *short*, which displays only start and end points in the clock path.

-transitive_fanout

Report constraints at the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

port_pin_list

Specifies a list of pins or ports to report. By default, the report contains all pins with pulse clock maximum width constraint.

DESCRIPTION

The **report_pulse_clock_max_width** command displays the maximum pulse clock width computation of the specified pins and ports. By default all pins with pulse clock width constraint are reported. To enable pulse clock constraint checking, set the variable `timing_enable_pulse_clock_constraints` to true. This variable set to true by default.

EXAMPLES

The following example generates a report of all maximum pulse width violations in the current design.

```
pt_shell> report_pulse_clock_max_width
*****
Report : pulse clock max width
-path_type summary
Design : test
*****

sequential_pulse_clock_max_width

Pin                                Required    Actual
      pulse width                pulse width    Slack
-----
ff1/cp (high)                     1.00         0.54      0.46 (MET)
ff2/cp (low)                      1.00         0.56      0.44 (MET)

clock_tree_pulse_clock_max_width

Pin                                Required    Actual
      pulse width                pulse width    Slack
-----
u3/zn (low)                       1.00         0.56      0.44 (MET)
p1/z (high)                       1.00         0.54      0.46 (MET)
u3/i (high)                       1.00         0.54      0.46 (MET)
```

SEE ALSO

set_pulse_clock_max_width (2), **remove_pulse_clock_max_width** (2), **report_constraint** (2).

report_pulse_clock_min_transition

Displays minimum transition computation at the input of pulse generator.

SYNTAX

```
int report_pulse_clock_min_transition
    [-rise][-fall]
    [-all_violators]
    [-significant_digits digits]
    [-nosplit]
    [port_pin_list]
```

```
list port_pin_list
```

ARGUMENTS

-rise

Report slack for rise transition.

-fall

Report slack for fall transition.

-all_violators

Specifies that only the input pins of pulse generators violating minimum transition constraint will be reported.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

port_pin_list

Specifies a list of pins or ports to report. By default, the report contains all pins with pulse clock minimum transition constraint.

DESCRIPTION

The **report_pulse_clock_min_transition** command displays the minimum transition computation of pins and ports. By default all pins with pulse clock minimum transition constraint are reported. Similarly, if rise or fall is not specified, then both rise and fall transition slack is reported. To enable pulse clock constraint checking, set the variable `timing_enable_pulse_clock_constraints` to true. This variable is set to true by default.

report_pulse_clock_min_transition

EXAMPLES

The following example generates a report of all minimum transition violations in the current design.

```
pt_shell> report_pulse_clock_min_transition
report_pulse_clock_min_transition
*****
Report : pulse clock min transition
-rise
-fall
Design : test
*****
```

```
pulse_clock_min_transition_rise
```

Pin	Required Transition	Actual Transition	Slack	
p1/i	0.30	0.12	-0.18	(VIOLATED)

```
pulse_clock_min_transition_fall
```

Pin	Required Transition	Actual Transition	Slack	
p1/i	0.30	0.12	-0.18	(VIOLATED)

SEE ALSO

```
set_pulse_clock_min_transition (2), remove_pulse_clock_min_transition (2),
report_constraint (2).
```

report_pulse_clock_min_width

Displays minimum pulse width computation for the pulse clock network.

SYNTAX

```
int report_pulse_clock_min_width
    [-all_violators]
    [-significant_digits digits]
    [-nosplit]
    [-path_type format]
    [-transitive_fanout]
    [port_pin_list]
```

list *port_pin_list*

ARGUMENTS

-all_violators

Specifies that only the pins in the pulse clock network violating the minimum pulse width constraint will be reported.

-significant_digits *digits*

Specifies the number of reported digits to the right of the decimal point. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the width of the column, the next field begins on a new line, starting in the correct column. **-nosplit** prevents line splitting and facilitates writing software to extract information from the report output.

-path_type *format*

Specifies the format of the path report and how the clock path is displayed. Allowed values are: *summary* (the default), which generates a report with a column format that shows one line for each path and shows only the required pulse width, actual pulse width and slack; and *short*, which displays only start and end points in the clock path.

-transitive_fanout

Report constraints at the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

port_pin_list

Specifies a list of pins or ports to report. By default, the report contains all pins with pulse clock minimum width constraint.

DESCRIPTION

The **report_pulse_clock_min_width** command displays the minimum pulse clock width computation of the specified pins and ports. By default all pins with pulse clock width constraint are reported. To enable pulse clock constraint checking, set the variable `timing_enable_pulse_clock_constraints` to true. This variable is set to true by default.

EXAMPLES

The following example generates a report of all minimum pulse width violations in the current design.

```
pt_shell> report_pulse_clock_min_width
*****
Report : pulse clock min width
-path_type summary
Design : test
*****
```

sequential_pulse_clock_min_width

Pin	Required pulse width	Actual pulse width	Slack	
ff2/cp (low)	2.00	0.56	-1.44	(VIOLATED)
ff1/cp (high)	0.80	0.54	-0.26	(VIOLATED)

clock_tree_pulse_clock_min_width

Pin	Required pulse width	Actual pulse width	Slack	
u3/i (high)	0.80	0.54	-0.26	(VIOLATED)
u3/zn (low)	0.80	0.56	-0.24	(VIOLATED)
p1/z (high)	0.80	0.54	-0.26	(VIOLATED)

SEE ALSO

set_pulse_clock_min_width (2), remove_pulse_clock_min_width (2), report_constraint (2).

report_qtm_model

Reports Quick Timing Model (QTM) data.

SYNTAX

```
string report_qtm_model [-global_parameters] [-ports] [-arcs] [-nosplit]
```

ARGUMENTS

-global_parameters

Specifies that global parameters of the current QTM design be reported.

-ports

Specifies that ports of the current QTM design be reported.

-arcs

Specifies that timing arcs of the current QTM design be reported.

-nosplit

This option prevents line splitting and facilitates writing software to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line starting in the correct column.

DESCRIPTION

The **report_qtm_model** command prints a report of the active QTM model. The information details the global parameters, ports, and arcs of the QTM model. By default, the command prints out the report of global parameters, ports, and arcs of the model.

If you want to see the global parameters in the QTM model, use the **-global_parameters** option.

If you want to see only the ports in the QTM model, use the **-ports** option.

If you want to see the arcs in the QTM model, use the **-arcs** option.

For a basic description of QTM, see the **create_qtm_model** manual page. For a more detailed description of QTM, see the *Prime Time User Guide*.

EXAMPLES

The following command reports all the attributes of the currently active QTM model.

```
pt_shell> report_qtm_model

*****
Report : qtm_model
```

Design : test_lib

PATH TYPES

Type	Cell	Input pin	Output pin	Fanout	Delay per level
path1	AN210	A	Y	2	0.570929

DRIVE TYPES

Type	Cell	Input pin	Output pin
drive1	AN210	A	Y

LOAD TYPES

Type	Cell	Input pin	Capacitance
load1	AN210	A	1

CLOCK RELATED PARAMETERS

Parameter	Cell	Clock Pin	Related Pin	Delay
setup	DTN10	CLK	D	1.33
hold	DTN10	CLK	D	1.33
setup	DTN10	CLK	Q	1.80244

Wire load model not defined

Max Transition not defined

PORTS

Port	Direction	Width	Cap Value	Load Type	Drive Value	Drive Type
CLK	CLOCK	1	-	-	-	-
A	INPUT	1	2	load1	-	-
B	INPUT	1	2	load1	-	-
X	OUTPUT	1	-	-	-	drive1
Y	OUTPUT	1	-	-	-	drive1

ARCS

Name	From	To	Delay	Path Type	Path Fact
CLK_to_A	CLK	A	1.142	path1	2
CLK_to_B	CLK	B	3.997	path1	7
CLK_to_X	CLK	X	3.000	-	-
CLK_to_Y	CLK	Y	7.000	-	-

The following command reports the global parameters of the currently active QTM model.

```
pt_shell> report_qtm_model -global_parameters
```

```
*****
Report : qtm_model
Design : test_lib
*****
```

PATH TYPES

Type	Cell	Input pin	Output pin	Fanout	Delay per level

path1	AN210	A	Y	2	0.570929

DRIVE TYPES

Type	Cell	Input pin	Output pin

drive1	AN210	A	Y

LOAD TYPES

Type	Cell	Input pin	Capacitance

load1	AN210	A	1

CLOCK RELATED PARAMETERS

Parameter	Cell	Clock Pin	Related Pin	Delay

setup	DTN10	CLK	D	1.33
hold	DTN10	CLK	D	1.33
setup	DTN10	CLK	Q	1.80244

```
*****
Wire load model not defined
*****
*****
Max Transition not defined
*****
```

SEE ALSO

create_qtm_model (2), **save_qtm_model** (2).

report_reference

Reports the references in current instance or design.

SYNTAX

string **report_reference** [-nosplit]

ARGUMENTS

-nosplit

Does not split lines if the column overflows.

DESCRIPTION

Displays information about all references in the current instance or current design. If the current instance is set, the report is generated relative to that instance; otherwise, the report is generated for the **current_design**.

EXAMPLES

The following is an example of **report_reference**.

```
pt_shell> report_reference
```

```
*****
Report : reference
Design : middle
*****
```

Attributes:

```
  b - black box (unknown)
  h - hierarchical
  n - noncombinational
```

Reference	Library	Unit Area	Count	Total Area	Attributes
FD2	tech_lib	3.00	4	12.00	b
ND2	tech_lib	3.00	4	12.00	b
inter		6.00	2	12.00	h
low		2.00	2	4.00	h
Total 4 references				40.00	

SEE ALSO

`report_hierarchy` (2), `report_cell` (2).

report_scale_parasitics

Use to report the scaling that was done previously using the `scale_parasitics` command.

SYNTAX

```
int report_scale_parasitics  
[net_list]
```

Data Types

```
net_list      list
```

ARGUMENTS

```
net_list
```

Limits the report of scaling to this list of nets. If this option is used, it is assumed that net-specific scaling has been done for those nets. If this is not the case, an error gets generated.

DESCRIPTION

The `report_scale_parasitics` command can be used to report the scale factors for parasitic-scaling that was done previously through the `scale_parasitics` command.

EXAMPLES

This example reports scaling of parasitics done previously.

```
pt_shell> report_scale_parasitics
```

SEE ALSO

```
read_parasitics(2)  
scale_parasitics(2)  
reset_scale_parasitics(2)
```

report_scope_data

Reports relevant scope data stored in the specified file.

SYNTAX

```
int report_scope_data  
[-block_name blk_name]  
[-port_names port_names]  
[-clock_names clk_names]  
[-scope_scenarios names]  
[-check_types chk_types]  
[-instance inst_name]  
[-significant_digits digits]  
[-nosplit]  
[-verbose]  
file_name
```

ARGUMENTS

- block_name** *blk_name*
Specifies the name of the block for which the scope data is to be reported. By default, if not specified, scope data for all blocks stored in the file is reported.
- port_names** *port_names*
Specifies the names of ports for which the scope data is to be reported. Note these names are the block-level port names. If not specified, all input port scope data in the file is reported.
- clock_names** *clk_names*
Specifies the names of clocks for which the scope data is to be reported. Note these names are for the block-level clocks. If not specified, all clock scope data in the file is reported.
- instance** *inst_name*
Specifies the name of a cell instance in the design for which the scope data is to be reported. This requires that the top level design to be properly linked.
- scope_scenarios** *scenario_names*
Specifies the list of scenario names for which the scope data are to be reported. If not specified, all available scenarios in the scope data file are reported.
- check_types** *chk_types*
Specifies the types of scope data to be reported from the file. If not specified, the types defined by environment variable *hier_scope_check_defaults* are used. Note that check type *clock_skew_with_uncertainty* is a derived type so it is not supported by this command.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-nosplit

Specifies whether the long lines in the report should be split into multiple lines. By default, lines are split.

-verbose

Indicates that the report of scope data should be very detailed. If not specified, by default, only a very summarized report is given for the scope data stored in the file. However, when one of the options **-clock_names**, **-port_names**, **-check_types** is specified, the report will always be verbose with or without **-verbose**.

DESCRIPTION

This command is used to report on the information stored in the binary scope data file. It provides users with convenient access to the detailed but encrypted scope information. It is for data reporting purpose only, no modification is done to the existing data, nor any computation or checking is done.

Because it is just report the data from the file, this command can be used with or without any designs loaded and linked in memory. However, if option **-instance** is used, a linked top-level design is required.

EXAMPLES

This example generates a summarized reports of the available scope data stored in file `top/ilm.scope`.

```
pt_shell> report_scope_data top/ilm.scope
```

The following command generates a very detailed reports of the scope data stored in file `my_ETM_block.scope`

```
pt_shell> report_scope_data -verbose my_ETM_block.scope
```

SEE ALSO

create_ilm (2), **extract_model** (2), **create_scenario** (2), **check_block_scope** (2), **update_scope_data** (2), **hier_scope_check_defaults** (3).

report_si_aggressor_exclusion

Reports the exclusive groups set by command `set_si_aggressor_exclusion`.

SYNTAX

```
int report_si_aggressor_exclusion [-rise] [-fall] [-nosplit] [anets]  
list anets
```

ARGUMENTS

`-rise`

Reports all the exclusive groups that have been set to be exclusive in the *rise* direction. If neither the *-rise* nor the *-fall* options are specified, the exclusive groups of the aggressor nets *anets* in both *-rise* and *-fall* directions will be reported.

`-fall`

Reports all the exclusive groups that have been set to be exclusive in the *fall* direction. If neither the *-rise* nor the *-fall* options are specified, the exclusive groups of the aggressor nets *anets* in both *-rise* and *-fall* directions will be reported.

anets

Specifies the list of nets whose exclusive groups you want to be reported. You can use this option with either *-rise* or *-fall* option to specify exclusive groups only in that particular direction.

`-nosplit`

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

DESCRIPTION

The **report_si_aggressor_exclusion** reports all the exclusive groups that have been set by the command **set_si_aggressor_exclusion**.

EXAMPLES

The following example shows how the exclusion on nets *SCAN_LOGIC** for rise direction can be reported.

```
pt_shell> set_si_aggressor_exclusion [get_nets SCAN_LOGIC*] -rise  
1  
pt_shell> report_si_aggressor_exclusion  
*****  
Report : report_si_aggressor_exclusion  
        -rise  
Design : top
```

```
report_si_aggressor_exclusion  
780
```

```
Version: X-2005.12-DEV
Date   : Thu Aug 11 14:43:46 2005
*****
```

```
Attributes:
  R - Exclusive for Rise direction
  F - Exclusive for Fall direction
```

Exclusive group	Exclusive type	Number of Active Aggressors

SCAN_LOGIC1 SCAN_LOGIC2	R	1

The following example shows how the exclusive groups on specific nets can be reported for rise or fall direction.

```
pt_shell> set_si_aggressor_exclusion [get_nets {Agg1 Agg2 Agg3}]
-number_of_active_aggressors 2 -rise
1
pt_shell> set_si_aggressor_exclusion [get_nets {Agg2 Agg4}]
-number_of_active_aggressors 1 -fall
1
pt_shell> report_si_aggressor_exclusion [get_nets Agg2] -rise
*****
```

```
Report : report_si_aggressor_exclusion
  -rise
    -fall
Design : top
Version: X-2005.12-DEV
Date   : Thu Aug 11 14:43:46 2005
*****
```

```
Attributes:
  R - Exclusive for Rise direction
  F - Exclusive for Fall direction
```

Exclusive group	Exclusive type	Number of Active Aggressors

Agg1 Agg2 Agg3	R	2

```
pt_shell> report_si_aggressor_exclusion [get_nets Agg2] -fall
*****
Report : report_si_aggressor_exclusion
  -rise
    -fall
Design : top
Version: X-2005.12-DEV
Date   : Thu Aug 11 14:43:46 2005
*****
```

```
Attributes:
  R - Exclusive for Rise direction
  F - Exclusive for Fall direction
```

Exclusive group	Exclusive type	Number of Active Aggressors
-----	-----	-----
Agg2 Agg4	F	1

SEE ALSO

```
set_si_aggressor_exclusion (2), remove_si_aggressor_exclusion (2),
report_delay_calculation (2), report_noise_calculation (2),
si_analysis_logical_correlation_mode (3), set_si_delay_analysis (2),
set_si_noise_analysis (2),
```

report_si_bottleneck

Identify the crosstalk bottlenecks in the design. This is useful when the major sources of violations come from crosstalk effects.

SYNTAX

```
int report_si_bottleneck -cost_type type
[-slack_lesser_than slack_limit]
[-max_nets count]
[-significant_digits digits]
[-nosplit]
[-include_clock_nets]
[-minimum_active_aggressors active_aggressor_count]
[-min]
[-max]
[-pre_commands pre_command_string]
[-post_commands post_command_string]
cost_type
delta_delay, delta_delay_ratio, total_victim_delay_bump, delay_bump_per_aggressor
float slack_limit
int count, digits, active_aggressor_count
```

ARGUMENTS

-cost_type*type*
The nets are sorted based on the cost. The *delta_delay*, *delta_delay_ratio*, and *total_victim_delay_bump* are the costs of the victim net and the *delay_bump_per_aggressor* for aggressor nets of the selected victim nets.

-slack_lesser_than*slack_limit*
The cost function is applied only to the victim nets whose slack is less than the slack limit. The default is 0.0. To use a non-default *slack_limit*, you must set the **timing_save_pin_arrival_and_slack** variable to **true**.

-max_nets*count*
The maximum number of nets to be reported. The default is 20.

-significant_digits*digits*
The number of digits to display: Range: 0 to 13. The default is the as same as the global variable **report_default_significant_digits**.

-nosplit
Don't split lines if column overflows.

include_clock_nets
Include the clock nets for bottleneck. By default clock nets are excluded from the bottleneck analysis.

-minimum_active_aggressors*active_aggressor_count*
The minimum number of active aggressors for the net to be selected. The default is 1.

-min

For the minimum analysis. For example, the `slack_limit` applies to minimum slack of the net in the net selection.

-max

For the maximum analysis. For example, the `slack_limit` applies to maximum slack of the net in the net selection.

-pre_commands *pre_command_string*

This option is available only if you invoke PrimeTime with the `-multi_scenario` option. This option allows you to specify a string of commands to be executed in the slave context before the execution of the `merged_reporting` command. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

-post_commands *post_command_string*

This option is available only if you invoke PrimeTime with the `-multi_scenario` option. This option allows you to specify a string of commands to be executed in the slave context after the execution of the `merged_reporting` commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

DESCRIPTION

This command report the nets that are PrimeTime SI bottlenecks, either as a victim or aggressor. The cost functions are defined and reported on nets. The cost function is computed by considering only victims that have a slack lesser than `slack_limit`. If neither the `-min` nor `-max` options are specified, both minimum and maximum slacks are considered.

Four cost functions are available. The following three cost factors return a list of victim nets:

- * `delta_delay` - This cost factor is calculated by determining the worst delta delay on the victim net. This is useful for finding the largest delta delays in the design that contributes to failing paths.

- * `delta_delay_ratio` - This cost factor is calculated by determining the worst delta delay ratio (ratio of delta delay to total stage delay) on the victim net. This cost factor is useful for finding the delta delays that result in the largest percentage increase of a stage delay in a failing path.

- * `total_victim_delay_bump` - This cost factor is calculated by determining the height of the largest crosstalk bump on the victim net. This cost factor is useful for finding the strongest electrical couplings in the design that contributes to failing paths.

The following cost factor returns a list of aggressor nets:

- * `delay_bump_per_aggressor` - This cost factor is calculated for an aggressor net by finding all slack-qualifying victim nets that it aggresses, then summing up the bump voltages on those victims induced by the aggressor. This cost factor is useful for finding aggressors that result in the largest amount of electrical coupling to victim nets in failing paths.

If a victim meets the slack criteria only for one of its transition senses (rise or fall) but not for the other, only the PrimeTime SI information for the failing sense is considered.

By default, the clock nets are not included in the bottleneck victim analysis, as the slack of the clock nets is infinite. However, clock nets can be included as victims in the bottleneck search by specifying the `-include_clock_nets` option. Even without this option, clock aggressor nets can be returned by the `total_victim_delay_bump` cost factor.

Crosstalk problems can be caused by one or more of the following factors:

- * large coupling capacitance * weak victim net driver or slow victim transition * strong aggressor net driver

Depending on the relative victim and aggressor slacks, one of the following repairs could be considered:

- * downsize the aggressor net driver to a weaker driver

- * upsize the victim net driver to a stronger driver * model a physical decoupling using `set_coupling_separation` * insert a buffer on the victim net to break up long coupled routes

When sizing cells, it can be useful to use the `get_alternative_lib_cells` command to report equivalents, or the `report_alternative_lib_cells` command to also evaluate their potential slack improvements.

The aggressor-based cost factor can be useful for finding strong aggressors which attack many failing victim nets. If the aggressor has positive setup slack, it may be preferable to downsize the aggressor driver and weaken its effect over multiple victims, or to separate it from the victims using the `set_coupling_separation` command. When performing such an aggressor downsizing, ensure that the resulting weaker aggressor driver does not violate any library `max_capacitance` or `max_transition` DRC requirements.

When a large number of aggressors attacks a victim net, the user could potentially take advantage of the statistical nature of many aggressors where the chances of all of them switching at the same time are very low. To find the failing victim nets that are attacked by many aggressors, the `-minimum_active_aggressors` option can be applied. The number of active aggressors is defined as the number of effective aggressors which are active for the worst-case aggressor alignment. For example, a victim net can have ten effective aggressors. However, for any given min/max rise/fall scenario, perhaps no more than five of these ten effective aggressors are active in the worst-case alignment for that sense. Such a victim would not meet a `-minimum_active_aggressors` value higher than five.

If you want to exclude specific crosstalk victim/aggressor interactions from the analysis, the `set_si_delay_analysis -exclude` command can be used to achieve this without changing the parasitics or the design.

EXAMPLES

The following examples show a PrimeTime SI bottleneck report. arc with specified

start and end points.

```
pt_shell> report_si_bottleneck -cost_type delta_delay -significant_digits 6
```

```
*****
```

```
Report : si_bottleneck
        -cost_type delta_delay
        -slack_lesser_than 0
        -max_nets 1
        -significant_digits 6
        -number_of_active_aggressors 1
        -min
        -max
```

```
Design : TestBH
```

```
*****
```

```
Bottleneck Cost:  delta_delay
```

```
net                                                    cost
```

```
-----
InReg                                                    0.000007
```

SEE ALSO

report_default_significant_digits (3), **set_coupling_separation** (2), **size_cell** (2).

report_si_delay_analysis

Generates a report of user coupling information on nets for crosstalk delay analysis.

SYNTAX

```
int report_si_delay_analysis  
[-reselected]  
[-ignored_arrival]  
[-excluded]  
[-coupling_separated]  
[-disabled_statistical]  
[-nosplit]  
[nets]
```

list *nets*

ARGUMENTS

-reselected

Reports the list of nets you have reselected in each iteration, independent of reselection criteria. In some cases, the nets may not be reselected during crosstalk analysis because they have been filtered out. These nets will still be reported since you have specified them to be reselected and they may be potentially reselected after any parasitic changes.

-ignored_arrival

Reports the list of nets you have specified to be analyzed with infinite window.

-excluded

Reports the list of nets excluded by you from crosstalk analysis as victim nets or aggressor nets. Also specifies the analysis type for which the victim/aggressor nets are excluded. It indicates whether the information is set globally on a victim for all aggressors, or globally on a aggressor for all its victims, or between a particular victim and aggressor net. Also indicates the type of analysis - min path victim falling, min path victim rising, max path victim falling and max path victim rising analysis. The combination of analysis types for which the net is excluded is reported.

-coupling_separated

Reports the list of nets on which you have specified coupling separation constraint. It reports the aggressor nets which are separated from all their victims, the victim nets which are separated from all their aggressors, and separation applied between a particular pair of victim and aggressor net.

-disabled_statistical

Reports the nets which you have disabled for statistical analysis when the nets are considered as composite aggressor.

-no_split

Prevents line splitting and facilitates writing software to extract

information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

nets

Limits a list of nets in the current design for which the report needs to be generated.

DESCRIPTION

Produces a report showing coupling information specified by you on nets for crosstalk analysis.

The **report_si_delay_analysis** command allows you to view all the set of nets on which you have indicated the default crosstalk analysis behavior to be ignored and the behavior specified by you to take precedence. For every pair of nets, the type of behavior and analysis for which the default behavior is ignored, is reported. This command also specifies whether the information is specified for the net as a victim or aggressor.

Any combination of the options can be used to generate the corresponding report information. If no options are specified, the command reports all the information on all the nets.

This command reports the behavior set by the commands, **set_si_delay_analysis**, **remove_si_delay_analysis**, **set_coupling_separation**, **remove_coupling_separation**, **set_si_delay_disable_statistical** and **remove_si_delay_disable_statistical**.

EXAMPLES

The following example shows how the excluded option can be used to report all the pairs of victims or aggressors that have been excluded from crosstalk analysis.

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets CLK_NET*]
1
pt_shell> set_si_delay_analysis -exclude -agressors [get_nets AGG_NET*]
1
pt_shell> set_si_delay_analysis -exclude -victims [get_nets V_NET]
-aggressors [get_nets A_NET]
1
```

```
pt_shell> report_si_delay_analysis -excluded
```

```
*****
```

```
Report : report_si_delay_analysis
        -excluded
```

```
Design : TestBH
```

```
Version: X-2005.06
```

```
Date   : Fri Feb 11 11:28:09 2005
```

```
*****
```

Attributes:

E - Net is Excluded by user for

report_si_delay_analysis

788

mr - minimum rise analysis
 mf - minimum fall analysis
 Mr - Maximum rise analysis
 Mf - Maximum fall analysis
 I - Net has Ignored arrival
 P - Net has coupling constraint
 R - Net is Reselected for each crosstalk iteration
 T - Net is disabled for sTatistical analysis

Victim net	Aggressor net	Delay Analysis attributes
CLK_NET1	* E{ mr mf Mr Mf }	
CLK_NET2	* E{ mr mf Mr Mf }	
CLK_NET3	* E{ mr mf Mr Mf }	
*	AGG_NET1	E{ mr mf Mr Mf }
*	AGG_NET2	E{ mr mf Mr Mf }
V_NET	A_NET	E{ mr mf Mr Mf }

1

The '*' indicates all victim nets/ aggressor nets respectively.

The following example shows how to report all critical nets described by *SCAN_LOGIC** which have been reselected in all cross talk iterations.

```
pt_shell> set_si_delay_analysis -reselect [get_nets SCAN_LOGIC*]
1
```

```
pt_shell> report_si_delay_analysis -reselected
```

Report : report_si_delay_analysis
-reselected

Design : TestBH

Version: X-2005.06

Date : Fri Feb 11 11:28:09 2005

Attributes:

E - Net is Excluded by user for
 mr - minimum rise analysis
 mf - minimum fall analysis
 Mr - Maximum rise analysis
 Mf - Maximum fall analysis
 I - Net has Ignored arrival
 P - Net has coupling constraint
 R - Net is Reselected for each crosstalk iteration
 T - Net is disabled for sTatistical analysis

Victim net	Aggressor net	Delay Analysis attributes
SCAN_LOGIC1	* R	
SCAN_LOGIC2	* R	

1

The following example shows how to report all nets in a design which you have specified to be analyzed as infinite window.

```

pt_shell> set_si_delay_analysis -ignore_arrival [get_nets LOGIC0]
1
pt_shell> report_si_delay_analysis -ignored_arrival
*****
Report : report_si_delay_analysis
        -ignored_arrival
Design : TestBH
Version: X-2005.06
Date   : Fri Feb 11 11:28:09 2005
*****

```

Attributes:

- E - Net is Excluded by user for
 - mr - minimum rise analysis
 - mf - minimum fall analysis
 - Mr - Maximum rise analysis
 - Mf - Maximum fall analysis
- I - Net has Ignore arrival
- P - Net has coupling constraint
- R - Net is Reselected for each crosstalk iteration
- T - Net is disabled for statistical analysis

Victim net	Aggressor net	Delay Analysis attributes
LOGIC0	*	I
*	LOGIC0	I
1		

The following example shows how to report all nets in a design on which you have specified a coupling separation constraint.

```

pt_shell> set_coupling_separation [get_nets LOGIC1]
1
pt_shell> report_si_delay_analysis -coupling_separated
*****
Report : report_si_delay_analysis
        -coupling_separated
Design : TestBH
Version: X-2005.06
Date   : Fri Feb 11 11:28:09 2005
*****

```

Attributes:

- E - Net is Excluded by user for
 - mr - minimum rise analysis
 - mf - minimum fall analysis
 - Mr - Maximum rise analysis
 - Mf - Maximum fall analysis
- I - Net has Ignore arrival
- P - Net has coupling constraint
- R - Net is Reselected for each crosstalk iteration
- T - Net is disabled for statistical analysis

Victim net	Aggressor net	Delay Analysis attributes

```

LOGIC1          *          P
*                LOGIC1      P
1

```

The following example shows how to report all nets in a design which you have specified to be disabled from statistical analysis when considered as a composite aggressor.

```

pt_shell> set_si_delay_disable_statistical [get_nets LOGIC1]
1

```

```

pt_shell> report_si_delay_analysis -disabled_statistical

```

```

*****

```

```

Report : report_si_delay_analysis
        -disabled_statistical

```

```

Design : TestBH

```

```

Version: X-2005.06

```

```

Date   : Fri Feb 11 11:28:09 2005

```

```

*****

```

Attributes:

- E - Net is Excluded by user for
 - mr - minimum rise analysis
 - mf - minimum fall analysis
 - Mr - Maximum rise analysis
 - Mf - Maximum fall analysis
- I - Net has Ignored arrival
- P - Net has coupling constraint
- R - Net is Reselected for each crosstalk iteration
- T - Net is disabled for statistical analysis

Victim net	Aggressor net	Delay Analysis attributes
LOGIC1	*	T
*	LOGIC1	T
1		

The following example shows how to limit the report to a list of nets in a design on which you have specified any coupling information for crosstalk delay analysis.

```

pt_shell> report_si_delay_analysis [get_nets V1]

```

```

*****

```

```

Report : report_si_delay_analysis
        -coupling_separated

```

```

-disabled_statistical

```

```

-excluded

```

```

-ignored_arrival

```

```

-reselected

```

```

Design : TestBH

```

```

Version: X-2005.06

```

```

Date   : Fri Feb 11 11:28:09 2005

```

```

*****

```

Attributes:

- E - Net is Excluded by user for
 - mr - minimum rise analysis

- mf - minimum fall analysis
- Mr - Maximum rise analysis
- Mf - Maximum fall analysis
- I - Net has Ignore arrival
- P - Net has coupling constraint
- R - Net is Reselected for each crosstalk iteration
- T - Net is disabled for sTatistical analysis

Victim net	Aggressor net		Delay Analysis attributes

V1	*	I R	
*	V1	I T	
V1	V2	E{ mr mf Mr Mf }	
1			

SEE ALSO

set_si_delay_analysis (2), remove_si_delay_analysis (2); set_coupling_separation (2), remove_coupling_separation (2), set_si_delay_disable_statistical (2), remove_si_delay_disable_statistical (2).

report_si_double_switching

Reports the double switching violation detected in the design.

SYNTAX

```
int report_si_double_switching  
[-clock_network]  
[-rise]  
[-fall]  
[-nosplit]  
[nets]
```

ARGUMENTS

-clock_network
Report the double switch violations for the clock network only, independent of `si_xtalk_double_switching_mode` being "clock_network" or "full_design".

-rise
Report the double switch violations for the rising victim.

-fall
Report the double switch violations for the falling victim. If neither **-rise** or **-fall** is provided, the net is reported only once with smallest slack.

-nosplit
Don't split the line if it is longer than 80 characters.

nets
Report for the specific victim nets. If the *nets* is not provided it reports for all the nets of the design.

DESCRIPTION

The command **report_si_double_switching** reports the victim nets that has double switching violations. It prints the victim net name, actual bump height, the required bump height & the double switching slack.

Most users consider the clock network double switching as a critical violation. To report these higher priority double switching violations use **-clock_network** option.

The double switching detection needs to be enabled before **update_timing** by enabling the variable **si_xtalk_double_switching_mode**. Refer the man page of **si_xtalk_double_switching_mode** for more information.

EXAMPLES

Report all the double switching violations in the design for both rise and fall.

```
pt_shell> report_si_double_switching -nosplit -rise -fall
```

```

*****
Report : si_double_switching
        -no_split
        -rise
        -fall
Design : testcase
Version: X-2005.12
Date   : Thu Sep 15 13:30:53 2005
*****

```

Victim le Switching Net ~~~~~	Switching Direction ~~~~~	Actual Bump Height ~~~~~	Required Bump Height ~~~~~	Doub Slack ~~~~~
I2 0.26 (VIOLATING)	max_rise	0.69	0.42	-
I2 0.02 (VIOLATING)	max_fall	0.51	0.49	-

SEE ALSO

si_enable_analysis (2). **si_xtalk_double_switching_mode** (2).

report_si_noise_analysis

Generates a report of user coupling information on nets for crosstalk noise analysis.

SYNTAX

```
int report_si_noise_analysis  
[-ignored_arrival]  
[-excluded]  
[-coupling_separated]  
[-disabled_statistical]  
[-nosplit]  
[nets]
```

list *nets*

ARGUMENTS

-ignored_arrival

Reports the list of nets you have specified to be analyzed with infinite window for noise analysis.

-excluded

Reports the list of nets excluded by you from noise analysis as victim nets or aggressor nets. Also specifies the analysis type for which the victim/aggressor nets are excluded. The analysis type may be above low rail, below low rail, above high rail or below high rail. The combination of analysis types for which the net is excluded is reported. It indicates whether the information is set globally on a victim for all aggressors, or globally on a aggressor for all its victims, or between a particular victim and aggressor net.

-coupling_separated

Reports the list of nets on which you have specified coupling separation constraint. It reports the aggressor nets which are separated from all their victims, the victim nets which have been separated from all their aggressors, and coupling separation applied between a particular victim and aggressor net.

-disabled_statistical

Reports the nets which you have disabled for statistical analysis when the nets are considered as composite aggressor.

-no_split

Prevents line splitting and facilitates writing software to extract information from the report output. If you do not use this option, most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

nets

Limits a list of nets in the current design for which the report needs to be

generated.

DESCRIPTION

Produces a report showing coupling information specified by you on nets for noise analysis.

The **report_si_noise_analysis** command allows you to view all the set of nets on which you have indicated the default noise analysis behavior to be ignored and the behavior specified by you to take precedence. For every pair of nets, the type of behavior and analysis for which the default behavior is ignored, is reported. This command also specifies whether the information is specified for the net as a victim or aggressor.

Any combination of the options can be used to generate the corresponding report information. If no options are specified, the command reports all the information on all the nets.

This command reports the behavior set by the commands, **set_si_noise_analysis**, **remove_si_noise_analysis**, **set_coupling_separation**, **remove_coupling_separation**, **set_si_noise_disable_statistical** and **remove_si_noise_disable_statistical**.

EXAMPLES

The following example shows how the excluded option can be used to report all the pairs of victims or aggressors that have been excluded from noise analysis.

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets CLK_NET*]
1
pt_shell> set_si_noise_analysis -exclude -agressors [get_nets AGG_NET*]
1
pt_shell> set_si_noise_analysis -exclude -victims [get_nets V_NET]
-aggressors [get_nets A_NET]
1
```

```
pt_shell> report_si_noise_analysis -excluded
```

```
*****
```

```
Report : report_si_noise_analysis
        -excluded
```

```
Design : TestBH
```

```
Version: X-2005.06
```

```
Date   : Fri Feb 11 11:28:09 2005
```

```
*****
```

Attributes:

- E - Net is Excluded by user for
 - al - above low analysis
 - ah - above high analysis
 - bl - below low analysis
 - bh - below high analysis
- I - Net has Ignore arrival
- P - Net has coupling constraint
- R - Net is Reselected for each noise iteration

report_si_noise_analysis

T - Net is disabled for sTatistical analysis

Victim net	Aggressor net	Noise Analysis attributes
CLK_NET1	* E{ al ah bl bh }	
CLK_NET2	* E{ al ah bl bh }	
CLK_NET3	* E{ al ah bl bh }	
*	AGG_NET1	E{ al ah bl bh }
*	AGG_NET2	E{ al ah bl bh }
V_NET	A_NET	E{ al ah bl bh }

1

The '*' indicates all victim nets/ aggressor nets respectively.

The following example shows how to report all nets in a design which you have specified to be analyzed as infinite window.

```
pt_shell> set_si_noise_analysis -ignore_arrival [get_nets LOGIC0]
1
pt_shell> report_si_noise_analysis -ignored_arrival
*****
Report : report_si_noise_analysis
        -ignored_arrival
Design : TestBH
Version: X-2005.06
Date   : Fri Feb 11 11:28:09 2005
*****
```

Attributes:

- E - Net is Excluded by user for
 - al - above low analysis
 - ah - above high analysis
 - bl - below low analysis
 - bh - below high analysis
- I - Net has Ignore arrival
- P - Net has couPling constraint
- R - Net is Reselected for each noise iteration
- T - Net is disabled for sTatistical analysis

Victim net	Aggressor net	Noise Analysis attributes
LOGIC0	* I	
*	LOGIC0	I

1

The following example shows how to report all nets in a design on which you have specified a coupling separation constraint.

```
pt_shell> set_coupling_separation [get_nets LOGIC1]
1
pt_shell> report_si_noise_analysis -coupling_separated
*****
Report : report_si_noise_analysis
        -coupling_separated
Design : TestBH
```

```
Version: X-2005.06
Date   : Fri Feb 11 11:28:09 2005
*****
```

Attributes:

- E - Net is Excluded by user for
 - al - above low analysis
 - ah - above high analysis
 - bl - below low analysis
 - bh - below high analysis
- I - Net has Ignore arrival
- P - Net has coupling constraint
- R - Net is Reselected for each noise iteration
- T - Net is disabled for statistical analysis

Victim net		Aggressor net	Noise Analysis attributes
LOGIC1	*	P	
*	LOGIC1	P	
1			

The following example shows how to report all nets in a design which you have specified to be disabled from statistical analysis when considered as a composite aggressor.

```
pt_shell> set_si_noise_disable_statistical [get_nets LOGIC1]
1
```

```
pt_shell> report_si_noise_analysis -disabled_statistical
```

```
*****
```

```
Report : report_si_noise_analysis
        -disabled_statistical
```

```
Design : TestBH
```

```
Version: X-2005.06
```

```
Date   : Fri Feb 11 11:28:09 2005
```

```
*****
```

Attributes:

- E - Net is Excluded by user for
 - al - above low analysis
 - ah - above high analysis
 - bl - below low analysis
 - bh - below high analysis
- I - Net has Ignored arrival
- P - Net has coupling constraint
- T - Net is disabled for statistical analysis

Victim net		Aggressor net	Noise Analysis attributes
LOGIC1	*	T	
*	LOGIC1	T	
1			

The following example shows how to limit the report to a list of nets in a design on which you have specified any coupling information for noise noise analysis.

```
report_si_noise_analysis
```

```

pt_shell> report_si_noise_analysis [get_nets V1]
*****
Report : report_si_noise_analysis
        -coupling_separated
-disabled_statistical
-excluded
-ignored_arrival
-reselected
Design : TestBH
Version: X-2005.06
Date   : Fri Feb 11 11:28:09 2005
*****

```

Attributes:

```

E - Net is Excluded by user for
    al - above low analysis
    ah - above high analysis
    bl - below low analysis
    bh - below high analysis
I - Net has Ignore arrival
P - Net has coupling constraint
T - Net is disabled for statistical analysis

```

Victim net		Aggressor net	Noise Analysis attributes
V1	*	I	
*	V1		I T
V1	V2		E{ al ah bl bh }
1			

SEE ALSO

```

set_si_noise_analysis (2), remove_si_noise_analysis (2); set_coupling_separation
(2), remove_coupling_separation (2), set_si_noise_disable_statistical (2),
remove_si_noise_disable_statistical (2).

```

report_supply_net

Reports all supply nets associated with the specified power domain.

SYNTAX

```
int report_supply_net  
[-domain domain_name]  
[-scope scope_name]
```

Data Types

```
domain_name    string  
scope_name     string
```

ARGUMENTS

domain_name
Specify for which power_domain you want to define the supply nets that are reported. The domain must exist.

scope_name
Specify for which scope you want to define the supply nets that are reported. If this is not specified, report_supply_net reports all the power nets in the current scope.

DESCRIPTION

The *report_supply_net* command enables you to get the detail information of all supply nets associated with specified power domain or scope. The information of supply nets include: its full name, all power domain it is created in, all voltage information that is explicitly set on this net by the set_voltage command. Also, listed are all domains for which the net is the primary power or ground net.

Command returns 1 if succeed. Otherwise, 0 is returned.

EXAMPLES

The following example reports the information of all supply nets associated with the power domain named PD1 in the current scope.

```
prompt> report_supply_net -domain PD1  
Total of 1 supply nets defined.
```

```
-----  
Supply Net :      i2/net_i2  
Scope :          i2  
Power Domains :   i2/I1_LW2  
Supply Ports :    N/A
```

```
report_supply_net  
800
```


PG_power_pin : N/A
Max-delay Voltage : 1.08
Min-delay Voltage : 1.3
Resolve type : unresolved

1

SEE ALSO

create_supply_net(2)
create_power_domain(2)
connect_supply_net(2)
get_power_domains(2)
set_domain_supply_net(2)
set_scope(2)

report_switching_activity

Reports statistics on the switching activity and signal probability annotation on the current design or instance.

SYNTAX

```
int report_switching_activity
    [-cells cell_list]
    [-average_activity]
    [-base_clock clk]
    [-hierarchy]
    [-coverage]
    [-sort_by [hier | toggle]]
    [-toggle_limit limit]
    [-list_low_activity]
    [-list_by_source source]
    [-list_not_annotated]
    [-list_annotated]
    [-exclude exclusion_group]
    [-include_only inclusion_group]
    [-only_related_clock clock]
    [-show_pin]
```

```
int report_switching_activity -old
    [-rtl | -gate]
    [-list_not_annotated]
    [-cells cell_list]
```

```
list cell_list
string clk
int limit
string source
string exclusion_group
string inclusion_group
clock clock
```

ARGUMENTS

-rtl|-gate

These options are only supported when the **-old** option is used.

Indicates whether switching activity annotations are to be reported for objects annotated by an RTL backward SAIF file or a gate-level backward SAIF file. An RTL backward SAIF file is generated using RTL simulation, and contains the switching activity of synthesis invariant objects. These are objects that are not expected to change during synthesis, and include the design ports, and the outputs of sequential and tri-state cells. Calling the **report_switching_activity** command with the **-rtl** option reports the switching activity annotation on design ports, sequential cell outputs and tri-state outputs. Use the **-rtl** option after reading an RTL backward SAIF. A gate-level backward SAIF file is generated using gate-level simulation and contains the

switching activity of all design nets. Calling the **report_switching_activity** command with **-gate** option reports the switching activity annotation on the design nets and leaf cell internal power arcs and leakage states. Use the **-gate** option after reading a gate-level backward SAIF file. When neither **-rtl** or **-gate** options are used, the default **-gate** option is assumed. If both the options are provided, the default **-gate** is assumed.

-list_not_annotated

When the command is used with this flag, the report lists the design nets that have no user switching activity annotation. This report does not include constant value nets (logic one/zero nets). Note that such nets are annotated with default switching activity if they are not user annotated.

-list_annotated

When the command is used with this flag, the report lists the design nets that do have user switching activity annotation.
This option is not available with the *-old* option.

-cells *cells_list*

Indicates that switching activity annotation is to be reported only for the specified *cells_list*.

-average_activity

Produces a report that averages toggle rates over the nets in the design. When combined with the *-hierarchy* flag, average switching activity is computed for each subblock in the design.

It is possible using the filter options *-exclude* or *-include_only* to get a report of average toggle rates over a subset of the nets in the design. This can be useful, for instance, to get the average toggle rate for annotated nets, or even the average toggle rate for annotated nets driven by sequential cells.

This option is not available with the *-old* option.

-base_clock *clk*

When the *-average_activity* report is requested, average activities over nets are reported. The averaged toggle rates reported are by default with respect to 1ns. When the *-base_clock* option is used, the reported averaged toggle rates are with respect to the period of the clock *clk*. If the reported toggle rate is 0.5, and the period of the base clock is 3ns, then on average during simulation, there were 0.5 toggles every 3ns, or 1 toggle every 6 ns.

This option is not available with the *-old* option.

-hierarchy

When the command is used with this flag, the report generated will include information about sub blocks in the design. The flag cannot be used with the list generating options for the command.

This option is not available with the *-old* option.

-coverage

Causes the command to produce a summary report about the nets and the design that have switching activity information, but have few toggles. Coverage is defined as the percentage of nets with more than *limit* toggles.

See the *toggle_limit* option. You can use this report to verify that switching activities from simulation or propagation are properly exercising the design. Combined with the *-hierarchy* flag, you can use the report to make sure that

each block in the design is properly exercised.
This option is not available with the *-old* option.

-sort_by [hier | net_toggle_rate | name]

When used with the any of the hierarchical reports (that is, using the *-hierarchy* flag with the default report or with *-average_activity* or *-coverage* options), the hierarchical blocks in the report are sorted either by hierarchy (depth first), by averaged toggle rate, or by name.

When used with any of the list reports (that is, using *-list_low_activity* or *-list_by_source* or *-list_not_annotated* or *-list_annotated* options), the list is sorted by toggle rate or by name. For the list reports, the *-sort_by hier* option is not accepted.

This option is not available with the *-old* option.

-toggle_limit limit

Sets the lower limit for what is considered to be few toggles. This limit is used by the *-coverage* report and by the *-list_low_activity* list report. The default limit is 1 toggle.

The value of the argument to the **-toggle_limit** option represents the maximum number of toggles considered to be low activity. The number of toggles on a net is defined to be the toggle rate times the simulation duration. For vector free power analysis (where no SAIF file or VCD has been used) the default simulation duration is 10000 ns. If a SAIF has been used, the duration is taken from the SAIF file. If a VCD file has been used to annotate activity, the duration is the duration of the VCD simulation.

This option is not available with the *-old* option.

-list_low_activity

Reports a list of nets in the design with few toggles, where few is defined using the *-toggle_limit* option. The list report can be used to help debug which nets are not being exercised by the simulation testbench.

This option is not available with the *-old* option.

-list_by_source source

Reports a list of nets that have switching activity information from a given source.

Possible sources are any of the following:

[annotated, file, propagated, default, set_switching_activity, set_case_analysis, create_clock, implied, no_switching_activity]

Here *annotated* means any annotated (not propagated) source. The *file* source means any source with from an activity file (vcd or SAIF file).

This option is not available with the *-old* option.

-exclude exclusion_group

This filter option filters out nets from consideration before generating any of the reports the command can generate. Filtering occurs before toggle rate averaging, coverage percentage computations, and list generation. Possible exclusion groups are any of the following: [sequential, combinational, primary_inputs, black_boxes, three_state, memory, clock_gate, clock, low_activity, high_activity, rtl].

A net is defined as sequential if it is driven by a sequential cell. Likewise with combinational, primary_inputs, etc.

The rtl exclusion group refers to nets driven either by sequential cell outputs, black box outputs, primary inputs, or memories. The rtl group is mainly provided as a shorthand for writing out all of these.

The `clock_gate` exclusion group refers to ICGs only, not other types of clock gates.

Other possible exclusion groups are defined by the source of the activity information:

[*annotated*, *file*, *propagated*, *default*, *set_switching_activity*, *set_case_analysis*, *create_clock*, *implied*, *no_switching_activity*]

Here *annotated* means any annotated (not propagated) source. The *file* source means any source from an activity file (usually vcd or SAIF file).

To exclude multiple groups, the argument *exclusion_group* can be a list with multiple groups in the list.

This option is not available with the *-old* option.

Note: In this release, the "sequential" group should not include nets driven by clock gates. In the 2006.12 release, the group included such nets.

`-include_only inclusion_group`

This filter option filters out nets from consideration before generating any of the reports the command can generate. Filtering occurs before toggle rate averaging, coverage percentage computations, and list generation. Only nets that are members of the inclusion group are considered in the report generation.

Possible inclusion groups are any of the following: [*sequential*, *combinational*, *primary_inputs*, *black_boxes*, *three_state*, *memory*, *clock_gate*, *clock*, *low_activity*, *high_activity*, *rtl*].

A net is defined as sequential if it is driven by a sequential cell. Likewise, with combinational, primary_inputs, etc.

The `clock_gate` inclusion group refers to ICGs only, not other types of clock gates.

Other possible inclusion groups are defined by the source of the activity information:

[*annotated*, *file*, *propagated*, *default*, *set_switching_activity*, *set_case_analysis*, *create_clock*, *implied*, *no_switching_activity*]

Here *annotated* means any annotated (not propagated) source. The *file* source means any source from an activity file (usually VCD or SAIF file).

To include multiple groups, the argument *inclusion_group* can be a list with multiple groups in the list. In this case, only nets that are in one of the listed groups are included in the report.

To include only nets that are in several groups, you can use the `&` operator. This operator has low precedence; lists are or'ed first, then and'ed. For instance, to include only nets that are driven by rtl points or three_state buffers, and have no switching activity, use the following:

```
pt_shell> report_switching_activity -  
include_only {rtl,three_state & no_switching_activity}
```

To invert the selection, the `!` operator can be used. For example, one way to include only nets that are driven by rtl points, and have switching activity, would be the following:

```
pt_shell> report_switching_activity -  
include_only {rtl & !no_switching_activity}
```

There may be other ways (possibly using the *-exclude* option) to get this same set of nets included in the report.

This option is not available with the *-old* option.

`-only_related_clock clock`

If specified, nets that have a base_clock other than *clock* will be filtered out before the results of the report are included. This option can affect the overview, coverage, or the average toggle report. Also, this option can

affect the output lists. Such reports consider only the nets with the given `base_clock`.

`-show_pin`

Specifies that driver pins as well as nets are shown in certain net lists. The lists generated by `-list_not_annotated`, `-list_annotated`, `-list_low_activity`, and `-list_by_source` lists are affected. Pins are not shown for multi-driver nets.

DESCRIPTION

The **`report_switching_activity`** command underwent significant revision in the 2006.12 release. The old functional capabilities are available using the **`-old`** option. This old functionality will be obsoleted in the 2009.06 release.

Reports the switching activities on the nets in the current design or instance.

The default report generated by **`report_switching_activity`** gives an overview of switching activity information in the current instance; the report lists the percentage of nets with switching activity and signal probability that is user-annotated, default-annotated, and propagated (if any).

Switching activity and signal probability can be annotated by the user by the **`set_switching_activity`** and `read_saif` commands, or by using event-simulation data using the **`read_vcd`** command. The commands **`create_clock`** and **`set_case_analysis`** also affect switching activities. Note that the **`read_vcd`** command, when used with a named pipe or the **`-pipe`** option, does not immediately annotate activities on the design. The annotations happen later during **`update_power`**. Because of this, the **`report_switching_activity`** command does not report activity change in this situation until the **`update_power`** command has been run.

During power calculations, PrimeTime PX estimates the switching activity of objects that are not user-annotated by propagating the user or default annotated switching activity values. These objects will be reported as propagated after **`update_power`** has been run, but as not propagated prior to running **`update_power`**. In general, propagated switching activity is less accurate than switching activity on objects derived by simulation, so the percentage values in the user-annotated column of the report generated by **`report_switching_activity`** are an indication of the accuracy of power reports.

If the **`power_analysis_mode`** has been set to **`averaged`** or **`leakage_variation`**, and if a vcd file has been read with the **`read_vcdP`** command (without the **`-pipe`** option), then the tool will annotate the activity from the vcd prior to reporting the switching activity. In addition, unannotated objects where the activity can be derived accurately without random vector simulation (e.g. clock nets, Qbar pins where the Q pin is annotated, constant nets) will be reported by **`report_switching_activity`** as having *impliedP* activity. *Activity propagation for other unannotated nodes will not happen until **`update_power`** is run.*

If the **`power_analysis_mode`** has been set to **`time_based`**, then **`get_switching_activity`** will not have access to toggle rate information prior to running **`update_power`**. The command will report whether a net is uninitialized or whether it will be annotated from the VCD file, but since the VCD file has not been read yet, the toggle rate (and other values) will be reported as -1.

In some instances, you may have incomplete annotation from simulation. In this case, one or more nets may be unannotated after reading the activity file. Part of the purpose of the **report_switching_activity** command is to help you locate these annotation problems. If possible, you should get complete simulation data. If this is not possible, it may be useful to use the **-average_activity** report to get average toggle rates in the design for the purpose of patching these annotation problems using **set_switching_activity**.

EXAMPLES

The following examples report switching accuracy for the current default in **pt_shell**.

In the following example, the **report_switching_activity** command is used after the **read_saif** command.

```
pt_shell> report_switching_activity
```

```
*****
```

```
Report : Switching Activity
```

```
Design : mac
```

```
Version: B-2008.12-Beta2-DEV
```

```
Date   : Thu Oct  2 09:29:40 2008
```

```
*****
```

```
Switching Activity Overview Statistics for "mac"
```

Object Type (%)	From Activity		From Not	From	From	Default
	File (%)		SSA (%)	SCA (%)	Clock (%)	
	Propagated(%)	Implied(%)	Annotated(%)	Total		

Nets	1613 (99.02%)		0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (0.98%)	1629		

Nets Driven by						

Primary Input	67 (100.00%)		0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	67		
Tri- State	0 (0%)		0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		
Black Box	0 (0%)		0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		
Sequential	209 (92.89%)		0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (7.11%)	225		
Combinational	1337 (100.00%)		0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	1337		
Memory	0 (0%)		0 (0%)	0 (0%)	0 (0%)	0 (0%)

	0 (0%)	0 (0%)	0 (0%)	0		
Clock Gate	0 (0%)		0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		

Static Probability Overview Statistics for "mac"

Object Type	From Activity	From	From	From	
(%)	File (%)	Not	SSA (%)	SCA (%)	Clock (%)
Propagated(%)	Implied(%)	Annotated(%)	Total		Default
Nets	1613 (99.02%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (0.98%)	1629	

Nets Driven by

Primary Input	67 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	67	
Tri-State	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0	
Black Box	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0	
Sequential	209 (92.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (7.11%)	225	
Combinational	1337 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	1337	
Memory	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0	
Clock Gate	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0	

1

In the following example, the **report_switching_activity** is used with **-list_not_annotated** to find the nets in the design that do not have switching activity information. Note that the overview report is also printed, and lists several nets as having no activity. These nets are then listed by name below the overview report.

```
pt_shell> report_switching_activity -list_not_annotated
```

```
*****
Report : Switching Activity
        -list_not_annotated
Design : mac
Version: B-2008.12-Beta2-DEV
Date   : Thu Oct  2 09:27:39 2008
```

Switching Activity Overview Statistics for "mac"

Object Type (%)	From Activity		From Not		From Clock (%)	Default
	File (%)	Implied(%)	SSA (%)	SCA (%)		
	Propagated(%)		Annotated(%)	Total		
Nets	1613 (99.02%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (0.98%)	1629		

Nets Driven by

Primary Input	67 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	67		
Tri-State	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		
Black Box	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		
Sequential	209 (92.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (7.11%)	225		
Combinational	1337 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	1337		
Memory	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		
Clock Gate	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0		

Static Probability Overview Statistics for "mac"

Object Type (%)	From Activity		From Not		From Clock (%)	Default
	File (%)	Implied(%)	SSA (%)	SCA (%)		
	Propagated(%)		Annotated(%)	Total		
Nets	1613 (99.02%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	16 (0.98%)	1629		

Nets Driven by

Primary Input	67 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	0 (0.00%)	0 (0.00%)	0 (0.00%)	67		
Tri-State	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

0 (0%)	0 (0%)	0 (0%)	0		
Black Box	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Sequential	209 (92.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	16 (7.11%)	225		
Combinational	1337 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	0 (0.00%)	1337		
Memory	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Clock Gate	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		

List of nonannotated nets :

```

a_r[1]
a_r[0]
a_r[14]
a_r[8]
a_r[13]
a_r[4]
a_r[3]
a_r[10]
a_r[12]
a_r[2]
a_r[5]
a_r[11]
a_r[15]
a_r[7]
a_r[9]
a_r[6]

```

16 nets with no annotation

1

In the following example, the **report_switching_activity** command is used after the **update_power** command. The unannotated nets are now listed as having been propagated.

Report : Switching Activity

Design : mac

Version: B-2008.12-Beta2-DEV

Date : Thu Oct 2 09:29:16 2008

Switching Activity Overview Statistics for "mac"

Object Type	File (%)	From Activity	From Not	From	From	Default
(%)	Propagated(%)	Implied(%)	SSA (%)	SCA (%)	Clock (%)	
			Annotated(%)	Total		

report_switching_activity

810

Nets	1613 (99.02%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
16 (0.98%)	0 (0.00%)	0 (0.00%)	1629		

Nets Driven by

Primary Input	67 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	0 (0.00%)	67		
Tri-State	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Black Box	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Sequential	209 (92.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
16 (7.11%)	0 (0.00%)	0 (0.00%)	225		
Combinational	1337 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	0 (0.00%)	1337		
Memory	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Clock Gate	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		

Static Probability Overview Statistics for "mac"

Object Type (%)	From Activity File (%) Propagated(%)	From Activity Implied(%)	From Not SSA (%) Annotated(%)	From SCA (%) Total	From Clock (%)	Default
Nets	1613 (99.02%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
16 (0.98%)	0 (0.00%)	0 (0.00%)	1629			

Nets Driven by

Primary Input	67 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	0 (0.00%)	67		
Tri-State	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Black Box	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		
Sequential	209 (92.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
16 (7.11%)	0 (0.00%)	0 (0.00%)	225		
Combinational	1337 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
0 (0.00%)	0 (0.00%)	0 (0.00%)	1337		
Memory	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0 (0%)	0 (0%)	0 (0%)	0		

Clock Gate	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	0 (0%)	0 (0%)	0 (0%)	0	

1

In the following example the **report_switching_activity** command is used to get the average toggle rates on the design and subblocks of the design (using the **-hierarchy** flag)

```
pt_shell> report_switching_activity -average -hier
```

```
*****
Report : Switching Activity
        -average_activity -hierarchy
Design : counter
Version: V-2004.06-Alpha1
Date   : Mon Mar 29 11:09:34 2004
*****
Switching Activity Average for "counter"
```

```
-----
Switching Activities per clock is with respect to clock 'CLK'
Simulation time 10us (1000 clock periods of clock 'CLK')
-----
```

Object	Toggle Count Per Net	Toggle Rate Per Clock Per Net	Glitch Count Per Net	Glitch Rate Per Clock Per Net
counter	200	0.2	1	0.001
counter_low_bits	150	0.15	1	0.001
counter_high_bits	50	0.05	0	0
reset_block	1	0.001	0	0
overflow_detect	1	0.001	0	0

In the following example, the **report_switching_activity** command is used with the **-coverage** flag to determine whether or not the simulation testbench from which the SAIF file was derived did a good job in exercising each part of the design. In this example, the simulation did a poor job in exercising the counter_high_bits subblock.

```
pt_shell> report_switching_activity -coverage -hierarchical
```

```
*****
Report : Switching Activity
        -coverage -hierarchical
Design : counter
Version: V-2004.06-Alpha1
Date   : Mon Mar 29 11:09:34 2004
*****
```

report_switching_activity

Switching Activity Coverage for design "counter"

Coverage is defined as the percent of nets with at least 1 toggle

Block	% Nets Covered	# Nets Covered	Total Nets Counted
counter	76	76	100
counter_low_bits	100	44	44
counter_high_bits	25	11	44
reset_block	50	3	6
overflow_detect	50	3	6

SEE ALSO

`read_saif(2), reset_switching_activity(2), set_switching_activity(2),
get_switching_activity(2), set_rtl_to_gate_name(2), update_power(2), report_power
(2);`

report_timing

Reports timing paths.

SYNTAX

```

string report_timing
[-from from_list
    | -rise_from rise_from_list
    | -fall_from fall_from_list]
[-to to_list
    | -rise_to rise_to_list
    | -fall_to fall_to_list]
[-exclude exclude_list
    | -rise_exclude rise_exclude_list
    | -fall_exclude fall_exclude_list]
[-through through_list]
[-rise_through rise_through_list]
[-fall_through fall_through_list]
[-delay_type delay_type]
[-nworst paths_per_endpoint]
[-max_paths count]
[-path_type format]
[-true]
[-true_threshold path_delay]
[-justify]
[-false]
[-input_pins]
[-unique_pins]
[-start_end_pair]
[-nets]
[-slack_greater_than greater_slack_limit]
[-slack_lesser_than lesser_slack_limit]
[-ignore_register_feedback feedback_slack_cutoff]
[-report_ignored_register_feedback]
[-group group_name]
[-significant_digits digits]
[-nosplit]
[-transition_time]
[-capacitance]
[-crosstalk_delta]
[-trace_latch_borrow]
[-derate]
[-dont_merge_duplicates]
[-pre_commands pre_command_string]
[-post_commands post_command_string]
[-exceptions exception_type]
[-pba_mode effort]
[-voltage]
[path_collection]

```

Data Types

```
from_list      list
```

```
report_timing
814
```

<i>rise_from_list</i>	list
<i>fall_from_list</i>	list
<i>to_list</i>	list
<i>rise_to_list</i>	list
<i>fall_to_list</i>	list
<i>exclude_list</i>	list
<i>rise_exclude_list</i>	list
<i>fall_exclude_list</i>	list
<i>through_list</i>	list
<i>rise_through_list</i>	list
<i>fall_through_list</i>	list
<i>delay_type</i>	string
<i>paths_per_endpoint</i>	int
<i>count</i>	int
<i>format</i>	string
<i>path_delay</i>	float
<i>greater_slack_limit</i>	float
<i>lesser_slack_limit</i>	float
<i>feedback_slack_cutoff</i>	float
<i>group_name</i>	list
<i>digits</i>	int
<i>pre_command_string</i>	string
<i>post_command_string</i>	string
<i>exception_type</i>	string
<i>effort</i>	string
<i>path_collection</i>	collection

ARGUMENTS

-from from_list

Specifies that only paths from the named pins, ports, nets, cell instances, or startpoints clocked by named clocks are to be reported. If *from_list* is not specified, the default behavior reports the longest path to an output port if the design has no timing constraints. Otherwise, the default behavior is to report the path with the worst slack within each path group if the design has timing constraints.

-rise_from rise_from_list

Same as the *-from* option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from fall_from_list

Same as the *-from* option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-to to_list

Specifies that only paths to the named pins, ports, nets, cell instances or endpoints clocked by named clocks are to be reported. If *to_list* is not

specified, the default behavior reports the longest path to an output port if the design has no timing constraints. Otherwise, the default behavior is to report the path with the worst slack within each path group if the design has timing constraints. If a clock object is to be specified with the `-to` option, it must be specified as a collection (that is using the `[get_clocks 'clock name']` command).

`-rise_to` *rise_to_list*

Same as the `-to` option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path.

`-fall_to` *fall_to_list*

Same as the `-to` option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

`-exclude` *exclude_list*

Specifies that only paths not including the named pins, ports, nets, and cell instances in the data paths are to be reported. Reporting excludes all data paths from/through/to the named pins, ports, nets, and cell instances. If a cell instance is specified, all pins of the cell are excluded. The following applies to the `-exclude` option:

- It has higher precedence than the `-from`, `-through`, or `-to` option.
- It does not work with `-true` option.
- It is exclusive with the `-rise_exclude` or `-fall_exclude` option.
- It does not apply to borrowing path from the `-trace_latch_borrow` option or the clock path from the `-path full_clock` or `-path full_clock_expanded` option. This option is not applied to clock pins.

`-rise_exclude` *rise_exclude_list*

Same as the `-exclude` option, but applies only to paths rising at the named pins, ports, nets, and cell instances.

`-fall_exclude` *fall_exclude_list*

Same as the `-exclude` option, but applies only to paths falling at the named pins, ports, nets, and cell instances.

`-through` *through_list*

Specifies that only paths through the named pins, ports, cell instances, or nets are to be reported. If the *through_list* option is not specified, the default behavior reports the longest path to an output port if the design has no timing constraints. Otherwise, the default behavior reports the path with the worst slack within each path group if the design has timing constraints. You can use multiple `-through`, `-rise_through`, and `fall_through` options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

-from A1 -through B1 -through C1 -to D1

If more than one object is specified within one *-through*, *-rise_through*, or *-fall_through* option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

-from A1 -through {B1 B2} -through {C1 C2} -to D1

-rise_through *through_list*

This option is similar to the *-through* option, but applies only to paths with a rising transition at the specified objects.

-fall_through *through_list*

This option is similar to the *-through* option, but applies only to paths with a falling transition at the specified objects.

-delay_type *delay_type*

Specifies the type of path delay to be reported. Valid values are *max* (the default), *min*, *min_max*, *max_rise*, *max_fall*, *min_rise*, or *min_fall*. The "rise" or "fall" in the *delay_type* refers to a rising or falling transition at the path endpoint.

-nworst *paths_per_endpoint*

Specifies the number of paths to be reported per endpoint per path group. Allowed values are 1 to 2000000; the default is 1.

-max_paths *count*

Specifies the number of paths to be reported per path group. Allowed values are 1 to 2000000; the default value is equal to the *-nworst* setting.

-path_type *format*

Specifies the format of the path report and how the timing path is displayed. The allowed values are

- *short*, which displays only start and end points "in the timing path"
- *full* (the default), which displays the full path
- *full_clock*, which displays full clock paths in addition to the full timing path
- *full_clock_expanded*, which displays full clock paths between a primary clock and a related generated clock in addition to the *full_clock* timing path.
- *end*, which generates a report with a column format that shows one line for each path and shows only the endpoint path total, required-time, slack and CRP (clock reconvergence pessimism value) when the **timing_remove_clock_reconvergence_pessimism** variable is set to *true*
- *summary*, which displays only the path without the accompanying required-time and slack calculation

-true

Indicates that the longest (least-slack) true paths in the design are to be reported. This option can require long runtimes for certain designs that have many false paths. The **true_delay_prove_true_backtrack_limit** and **true_delay_prove_false_backtrack_limit** variables are used to limit the amount of backtracking during the operation of the **report_timing -true** command. The **set_case_analysis** command is used to specify a partial input vector to be considered for -true analysis. The -true option cannot be combined with the -max_paths(>1), -nworst(>1), -delay_type (path type other than max) , -unique, -rise/fall_through and -rise/fall_from options.

-true_threshold *path_delay*

Used with the -true option. Specifies a threshold path delay value, in library time units, used by the -true option to speed up the searching. If this option is specified, the **report_timing -true** command returns the first path it finds greater than or equal to *path_delay* rather than continuing to search for a longer one.

-justify

Indicates to find and report an input vector that sensitizes the reported paths, or to report the path as false if no input vector is found. The **set_case_analysis** command is used to specify a partial input vector to be considered for -justify analysis.

-false

Indicates that only false paths are to be reported. These are the paths where no sensitizing input vector is found. The **set_case_analysis** command is used to specify a partial input vector to be considered for -false analysis.

-input_pins

Indicates that input pins are to be shown in the path report. The default is to show only output pins.

-unique_pins

Indicates that only paths through a unique set of pins are to be reported. This option can require longer runtimes when used in combination with the -nworst option with a large number of paths targeted for reporting.

-start_end_pair

Indicates that paths are reported for each pair of startpoint and endpoint based on connectivity. This option can lead to long runtime and can lead to generating a huge number of paths depending on the design. By default this option will only search for paths which are violating. This default value can be changed by having an explicit -slack_lesser_than option. The options that do not work with this option are -nworst, -max_paths, -unique_pins, -true, -false, -justify, -slack_greater_than, -ignore_register_feedback, -report_ignored_register_feedback. Unlike other options of the **report_timing** command, this option causes the paths reported to no longer be sorted based on slack. Instead, paths are arranged based on the endpoint with those sharing the same endpoint appearing next to one another. The maximum number of paths reported is limited to 2000000. To avoid the potential of returning duplicate paths, this option works as though the **timing_report_always_use_valid_start_end_points** variable was set to true.

-nets

Indicates that nets are to be shown in the path report. The default is not to show nets.

-slack_greater_than *greater_slack_limit*

Indicates that only those paths with a slack greater (more positive) than *greater_slack_limit* are to be shown. This option is applied as a filter to the paths after they are generated. Therefore, the number of paths generated might be less than the number specified with the *-nworst* and *-max_paths* options. This option can be combined with the *-slack_lesser_than* option to show only those paths inside a given slack range.

-slack_lesser_than *lesser_slack_limit*

Indicates that only those paths with a slack less (more negative) than *lesser_slack_limit* are to be shown. This option can be combined with *-slack_greater_than* to select only those paths inside a given slack range.

-ignore_register_feedback *feedback_slack_cutoff*

Indicates that non-inverting timing loops should be ignored if they start and end at the same register pin that holds a value. To be ignored, the data-to-output arc and the output-to-data path must either both be inverting or both be non-inverting. This option applies to minimum delay as well as maximum delay reports. Paths are ignored only if they have a slack less than the specified *feedback_slack_cutoff* value. This option is applied as a filter to the paths after they are generated. Therefore, the number of paths generated might be less than the number specified with the *-nworst* and *-max_paths* options.

-report_ignored_register_feedback

Indicates that paths are to be reported if they are ignored when the *-ignore_register_feedback* option is specified.

-group *group_name*

Specifies the path groups from which timing paths are selected for reporting based on other specified options for reports.

-transition_time

Indicates that transition time (slew) is to be shown in the path report. The default is not to show transition time. For each driver pin or load pin the transition time is displayed in a column preceding incremental path delay.

-capacitance

Indicates that total (lump) capacitance is to be shown in the path report. The default is not to show capacitance. For each driver pin the total capacitance driven by the driver is displayed in a column preceding both incremental path delay and transition time (with *-transition_time*). When the *-nets* option is specified, the capacitance is printed on the lines with nets instead of the lines with driver pins.

-crosstalk_delta

Indicates that annotated delta delay and delta transition time is reported. The deltas are computed during crosstalk signal integrity analysis, or they can be annotated manually using the **set_annotated_delay -delta_only** and **set_annotated_transition -delta_only** commands. Note that the *-crosstalk_delta* option only reports the calculated or annotated deltas; it

does not initiate crosstalk analysis. Only deltas on input pins are shown. Delta transition time is shown only with the *-transition_time* option. The *-crosstalk_delta* option automatically sets the *-input_pins* option.

-derate

Indicates that derate factors are to be shown in the timing report. The default is to show no derate factors. Specifying this option automatically sets both the *-input_pins* and *-path_type full_clock_expanded* options. For each output pin of a cell in the report that cell's derate factor used is displayed in a column preceding the incremental path delay. For each input pin of a cell in the report its preceding nets' derate factors are displayed in a column preceding the incremental path delay. In addition a summary report will follow the timing report indicating what portion of the slack is a result of the application of derate factors. There is a minimum significant digits limit for derate factors of 2.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The *-nosplit* option prevents line-splitting and facilitates writing software to extract information from the report output.

-trace_latch_borrow

This option controls the type of report generated for a path that starts at a transparent latch. If the path startpoint borrows from the previous stage, using this option causes the report to show the entire set of borrowing paths that lead up to the borrowing latch, starting with a nonborrowing path or a noninverting sequential loop. By default, the report shows only the last path in the sequence of borrowing stages. Each stage is reported separately, showing the time borrowed and lent and the endpoints of the stage. The cumulative amount of borrowed time along a sequence of stages is not included in the report. The *-input_pins*, *-nets*, *-transition_times*, *-capacitance*, and *-significant_digits* options apply to every stage in the sequence of borrowing paths, but the remaining options (for example, the *-from* and *-true* options) apply only to the last stage reported.

-dont_merge_duplicates

This option is available only if you invoke PrimeTime with the *-multi_scenario* option. It turns OFF a main capability in merged reporting that is ON by default. The option affects the manner in which paths from multiple scenarios are merged. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. By using this option, PrimeTime will not merge duplicate instances of the same path into a single instance, but instead shows all critical instances

of the path from all scenarios. Since the number of paths reported is limited by the *-nworst*, *-max_paths*, and other options of this command, the resulting merged report, when this option is used, may not be evenly spread out across the design, but instead can be focused on the portion of the design that is critical in each scenario.

-pre_commands pre_command_string

This option is available only if you invoke PrimeTime with the *-multi_scenario* option. This option allows you to specify a string of commands to be executed in the slave context before the execution of the merged reporting commands. Commands must be grouped using the ";" character. The maximum size of a command is 1000 chars.

-post_commands post_command_string

This option is available only if you invoke PrimeTime with the *-multi_scenario* option. This option allows you to specify a string of commands to be executed in the slave context after the execution of the merged reporting commands. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

-exceptions

Prints user-entered timing exceptions, namely false paths, multicycle paths, and minimum and maximum delays, that are satisfied per timing path being reported. The reason of an unconstrained timing path is also printed. The *-exceptions* options requires one and only one of the following three values: *dominant*, *overridden*, or *all*.

Note: The additional analysis required per path with the *-exceptions* option is non-trivial. Therefore, using a **report_timing** command with the *-exceptions* option is expected to execute slower than the exact same command without this option. The *-exceptions* option does not work with *-path_type short/end/summary* option and forces the *-input_pins* option.

-pba_mode effort

This option controls path-based analysis. The *effort* value can be one of {*none*, *path*, *exhaustive*}.

- When *effort* is set to *none* (the default), path-based analysis is not applied.
- When *effort* is set to *path*, path-based analysis is applied to paths after they have been gathered.
- When *effort* is set to *exhaustive*, an exhaustive path-based analysis path search algorithm is applied to determine the worst path-based analysis path set in the design.

This option cannot be specified with the *-true* option when *effort* is set to *path*.

This option cannot be specified with the *-justify*, *-start_end_pair*, or *path_collection* options when *effort* is set to *exhaustive*.

-voltage

Reports the operating voltage for each path element. This option allows to debug voltages in multi-voltage designs. The displayed voltage is the voltage used in delay calculation - typically the voltage in operating condition, voltage of the related supply net or power pin, or pin signal level.

path_collection

Specifies the collection of timing paths to report. When this option is specified with the *-pba_mode path* option, path-based analysis is performed on the paths in the *path_collection* before they are printed. This option is mutually exclusive with options that control the selection of paths to report and is only compatible with options which control the formatting of the report.

DESCRIPTION

Provides a report of timing information for the current design.

Back-annotations on path elements in the timing path are indicated using a character symbol. If the *-input_pins* option is used, each pin-to-pin delay will span either a net or cell. The symbol shown refers to the dominant annotation on this path element. (Certain annotations dominate others; for example, SDF takes precedence over back-annotated RC parasitics.) Without the *-input_pins* option, the delay shown might span both a net and a cell. If they have the same dominant annotation, the appropriate symbol will be shown; otherwise, the symbol "H" appears to show that a hybrid of annotation types exists on the corresponding path segment.

Symbol	Annotation
-----	-----
H	Hybrid annotation
^	Ideal network latency annotation
*	SDF back-annotation
&	RC network back-annotation
\$	RC pi back-annotation
+	Lumped RC
<none>	Wire-load model or none

The symbol is indicative only of the dominant annotation, and you should note that this might not have been used to calculate the observed delay. For example, suppose that a back-annotated RC network exists on a net, but that the network calculation fails to converge for the driver cell delay. In this case, you are alerted, and a lumped RC model is used instead; but, during report timing, the "&" symbol appears anyway. To see how the pin-to-pin delay is actually calculated, use the **report_delay_calculation** command.

The command can also report on a collection of timing paths which were retrieved using the **get_timing_paths** command.

When the *-exceptions* option is used, the possible reasons for unconstrained paths and their corresponding timing path attributes are listed below (the timing path attributes are not contained in the report):

Attribute	Possible Reason

```

-----
dominant_exception          false_path / min_max_delay /
                             multicycle_path
startpoint_unconstrained_reason no_launch_clock / dangling_start_point /
                             fanout_of_disabled
endpoint_unconstrained_reason no_capture_clock / dangling_end_point /
                             fanin_of_disabled

```

EXAMPLES

The following example shows a timing report using only the default values.

```
pt_shell> report_timing
```

```
*****
```

```
Report : timing
```

```
    -path full
```

```
    -delay max
```

```
    -max_paths 1
```

```
Design : led
```

```
Version: 1997.01-development
```

```
Date   : Tue Aug  6 12:28:38 1996
```

```
*****
```

```
Startpoint: c (input port)
```

```
Endpoint: z2 (output port)
```

```
Path Group: default
```

```
Path Type: max
```

Point	Incr	Path

input external delay	0.00	0.00 r
c (in)	0.00	0.00 r
u1/Z (IVA)	0.54	0.54 f
u0/Z (NR2)	1.20	1.74 r
u8/Z (IVA)	0.43	2.17 f
u7/Z (OR3)	1.24	3.41 f
z2 (out)	0.00	3.41 f
data arrival time		3.41
max_delay	0.00	0.00
output external delay	0.00	0.00
data required time		0.00

data required time		0.00
data arrival time		-3.41

slack (VIOLATED)		-3.41

The following example reports the endpoint path delay, required time, and slack for each path.

```
pt_shell> report_timing -path end
```

```
*****
Report : timing
        -path end
        -delay max
Design : led
Version: 1997.01-development
Date   : Tue Aug  6 12:31:18 1996
*****
```

Endpoint	Path Delay	Path Required	Slack
z2	3.41 f	0.00	-3.41
z3	3.03 f	0.00	-3.03
z4	2.77 f	0.00	-2.77
z6	2.69 r	0.00	-2.69
z0	2.59 f	0.00	-2.59
z1	2.37 f	0.00	-2.37
z5	2.26 f	0.00	-2.26

The following example reports the start and end points of the path from a to z2.

```
pt_shell> report_timing -from a -to z2 -path short
```

```
*****
Report : timing
        -path short
        -delay max
        -max_paths 1
Design : led
Version: 1997.01-development
Date   : Tue Aug  6 12:34:15 1996
*****
```

```
Startpoint: a (input port)
Endpoint: z2 (output port)
Path Group: default
Path Type: max
```

Point	Incr	Path
input external delay	0.00	0.00 f
a (in)	0.00	0.00 f
...		
z2 (out)	1.24	1.24 f
data arrival time		1.24
max_delay	0.00	0.00

output external delay	0.00	0.00
data required time		0.00

data required time		0.00
data arrival time		-1.24

slack (VIOLATED)		-1.24

The following example shows input pins in the report, in addition to the default values.

```
pt_shell> report_timing -input_pins
```

```
*****
Report : timing
        -path full
        -delay max
        -input_pins
        -max_paths 1
Design : led
Version: 1997.01-development
Date   : Tue Aug  6 12:35:24 1996
*****
```

```
Startpoint: c (input port)
Endpoint:   z2 (output port)
Path Group: default
Path Type:  max
```

Point	Incr	Path

input external delay	0.00	0.00 r
c (in)	0.00	0.00 r
u1/A (IVA)	0.00	0.00 r
u1/Z (IVA)	0.54	0.54 f
u0/A (NR2)	0.00	0.54 f
u0/Z (NR2)	1.20	1.74 r
u8/A (IVA)	0.00	1.74 r
u8/Z (IVA)	0.43	2.17 f
u7/B (OR3)	0.00	2.17 f
u7/Z (OR3)	1.24	3.41 f
z2 (out)	0.00	3.41 f
data arrival time		3.41
max_delay	0.00	0.00
output external delay	0.00	0.00
data required time		0.00

data required time		0.00
data arrival time		-3.41

slack (VIOLATED)		-3.41

The following example shows input pins and nets in the report and does not show required time and slack calculation.

```
pt_shell> report_timing -input_pins -nets -path only
```

```
*****
Report : timing
        -path only
        -delay max
        -input_pins
        -nets
        -max_paths 1
Design : led
Version: 1997.01-development
Date   : Tue Aug  6 12:37:12 1996
*****
```

```
Startpoint: c (input port)
Endpoint: z2 (output port)
Path Group: default
Path Type: max
```

Point	Incr	Path
input external delay	0.00	0.00 r
c (in)	0.00	0.00 r
c (net)	0.00	0.00 r
u1/A (IVA)	0.00	0.00 r
u1/Z (IVA)	0.54	0.54 f
cell124/n22 (net)	0.00	0.54 f
u0/A (NR2)	0.00	0.54 f
u0/Z (NR2)	1.20	1.74 r
cell124/n21 (net)	0.00	1.74 r
u8/A (IVA)	0.00	1.74 r
u8/Z (IVA)	0.43	2.17 f
cell124/n19 (net)	0.00	2.17 f
u7/B (OR3)	0.00	2.17 f
u7/Z (OR3)	1.24	3.41 f
z2 (net)	0.00	3.41 f
z2 (out)	0.00	3.41 f
data arrival time		3.41

The following example reports the longest true paths in the design.

```
pt_shell> report_timing -true
```

```
*****
Report : timing
```

```

    -path full
    -delay max
    -true
    -max_paths 1
Design : led
Version: 1997.01-development
Date   : Tue Aug  6 12:38:35 1996
*****

```

```

Startpoint: c (input port)
Endpoint: z2 (output port)
Path Group: default
Path Type: max

```

Point	Incr	Path

input external delay	0.00	0.00 r
c (in)	0.00	0.00 r
u1/Z (IVA)	0.54	0.54 f
u0/Z (NR2)	1.20	1.74 r
u8/Z (IVA)	0.43	2.17 f
u7/Z (OR3)	1.24	3.41 f
z2 (out)	0.00	3.41 f
data arrival time		3.41
max_delay	0.00	0.00
output external delay	0.00	0.00
data required time		0.00

data required time		0.00
data arrival time		-3.41

slack (VIOLATED)		-3.41
True-delay Input Vector		

d (in)	0	
a (in)	0	
b (in)	0	
c (in)	r	

The following example reports transition time and capacitance.

```

pt_shell> report_timing -transition_time -capacitance
*****
Report : timing
    -path full
    -delay max
    -max_paths 1

```

```

        -transition_time
        -capacitance
Design : counter
Version: 1997.08-PV
Date   : Mon Aug  4 09:51:19 1997
*****

```

```

Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

```

Point	Cap	Trans	Incr	Path

clock CLK (rise edge)			0.00	0.00
clock network delay (ideal)			0.00	0.00
ffa/CLK (DTC10)		0.00	0.00	0.00 r
ffa/Q (DTC10)	3.85	0.57	1.70	1.70 f
U7/Y (IV110)	6.59	1.32	0.84	2.55 r
U12/Y (NA310)	8.87	2.47	2.04	4.58 f
U17/Y (NA211)	4.87	1.01	1.35	5.94 f
U23/Y (IV120)	2.59	0.51	0.37	6.30 r
U15/Y (BF003)	2.61	0.88	0.82	7.12 f
U16/Y (BF003)	2.61	1.46	0.99	8.11 r
U10/Y (AN220)	2.63	0.46	1.04	9.15 r
ffd/D (DTN10)		0.46	0.00	9.15 r
data arrival time				9.15
clock CLK (rise edge)			10.00	10.00
clock network delay (ideal)			0.00	10.00
ffd/CLK (DTN10)				10.00 r
library setup time			-1.33	8.67
data required time				8.67

data required time				8.67
data arrival time				-9.15

slack (VIOLATED)				-0.48

The following example includes nets and input pins and reports transition time and capacitance.

```

pt_shell> report_timing -transition_time -capacitance -nets -input_pins
*****
Report : timing
        -path full
        -delay max
        -input_pins
        -nets
        -max_paths 1

```

```

        -transition_time
        -capacitance
Design : counter
Version: 1997.08-PV
Date   : Mon Aug  4 09:57:10 1997
*****

```

```

Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

```

Point	Fanout	Cap	Trans	Incr	Path

clock CLK (rise edge)				0.00	0.00
clock network delay (ideal)				0.00	0.00
ffa/CLK (DTC10)			0.00	0.00	0.00 r
ffa/Q (DTC10)			0.57	1.70	1.70 f
b (net)	2	3.85			
U7/A (IV110)			0.57	0.00	1.70 f
U7/Y (IV110)			1.32	0.84	2.55 r
QA (net)	5	6.59			
U12/A (NA310)			1.32	0.00	2.55 r
U12/Y (NA310)			2.47	2.04	4.58 f
n9 (net)	3	8.87			
U17/A (NA211)			2.47	0.00	4.58 f
U17/Y (NA211)			1.01	1.35	5.94 f
n14 (net)	2	4.87			
U23/A (IV120)			1.01	0.00	5.94 f
U23/Y (IV120)			0.51	0.37	6.30 r
n13 (net)	1	2.59			
U15/A2 (BF003)			0.51	0.00	6.31 r
U15/Y (BF003)			0.88	0.81	7.12 f
n12 (net)	1	2.61			
U16/B2 (BF003)			0.88	0.00	7.12 f
U16/Y (BF003)			1.46	0.99	8.11 r
n7 (net)	1	2.61			
U10/A (AN220)			1.46	0.00	8.11 r
U10/Y (AN220)			0.46	1.04	9.15 r
n15 (net)	1	2.63			
ffd/D (DTN10)			0.46	0.00	9.15 r
data arrival time					9.15
clock CLK (rise edge)				10.00	10.00
clock network delay (ideal)				0.00	10.00
ffd/CLK (DTN10)					10.00 r
library setup time				-1.33	8.67
data required time					8.67

data required time					8.67
data arrival time					-9.15

slack (VIOLATED)					-0.48

The following example includes crosstalk delta delays and transition times.

```
pt_shell> report_timing -transition_time -capacitance -nets \
          -crosstalk_delta -significant_digits 4
```

```
Report : timing
        -path full
        -delay max
        -input_pins
        -nets
        -max_paths 1
        -transition_time
        -capacitance
        -crosstalk_delta
```

```
Design : TestBH
```

```
Startpoint: FirstReg (rising edge-triggered flip-flop clocked by myclock2)
Endpoint: SecondReg (rising edge-triggered flip-flop clocked by myclock3)
Path Group: myclock3
Path Type: max
```

Point	Fanout	Cap	DTrans	Trans	Delta	Incr
Path						

clock myclock2 (rise edge)						2.0000
2.0000						
clock network delay (propagated)						0.0000
2.0000						
FirstReg/						
CP (FD1Q)			0.1634		0.0000	2.0000 r
FirstReg/						
Q (FD1Q)			0.4030		0.4286 &	2.4286 f
OutFirstReg (net)	1	0.0996				
InstSimple/						
InC (SimpleComb)			0.0000		0.0000	2.4286 f
InstSimple/InC (net)						
InstSimple/InterNand/						
B (ND2)	0.1435	0.5467	0.1030	0.1092 &	2.5377 f	
InstSimple/InterNand/						
Z (ND2)		0.7001		0.3598 &	2.8976 r	
InstSimple/OutS (net)	1	0.1001				
InstSimple/						
OutS (SimpleComb)			0.0000		0.0000	2.8976 r
Reg (net)						
SecondReg/						
D (FD1Q)		0.1540	0.8542	0.2562	0.2615 &	3.1590 r
data arrival time						
3.1590						
clock myclock3 (rise edge)						3.0000

```

3.0000
  clock network delay (propagated) 0.0000
3.0000
  SecondReg/
CP (FD1Q) 3.0000 r
  library setup time -
0.2251 2.7749
  data required time
2.7749
-----
  data required time
2.7749
  data arrival time -
3.1590
-----
  slack (VIOLATED) -
0.3841

```

The following example shows an unconstrained path. It should be noted that for unconstrained paths, PrimeTime removes any clock network delay and external delay from the path report. However, the paths are sorted by the actual arrival time including clock network delay and external delay. In this example, the second path reported has greater delay than the first since the clock network delay used to find the first path was greater.

```

pt_shell> report_timing -to an2/B -delay_type max_rise -path_type full_clock \
          -nworst 2

```

```

*****

```

```

Report : timing
        -path full_clock
        -delay max_rise
        -nworst 2
        -max_paths 2

```

```

Design : simpl
Version: 2000.11-SI1
Date   : Tue Sep  5 11:13:30 2000

```

```

*****

```

```

Startpoint: ff1 (rising edge-triggered flip-flop clocked by SYSCLK2)
Endpoint: an2/B (internal pin)
Path Group: (none)
Path Type: max

```

Point	Incr	Path
ff1/CP (FD2)	0.00	0.00 r
ff1/Q (FD2)	1.34	1.34 r
an1/Z (AN2)	0.48	1.82 r

```

an2/B (sub) <-          0.00      1.82 r
data arrival time      1.82
-----
(Path is unconstrained)

```

```

Startpoint: ff0a (rising edge-triggered flip-flop clocked by SYSCLK1)
Endpoint: an2/B (internal pin)
Path Group: (none)
Path Type: max

```

Point	Incr	Path

ff0a/CP (FD2)	0.00	0.00 r
ff0a/Q (FD2)	1.34	1.34 r
an0/Z (AN2)	0.91	2.25 r
an1/Z (AN2)	0.48	2.73 r
an2/B (sub) <-	0.00	2.73 r
data arrival time		2.73

(Path is unconstrained)		

The following example shows the *-full_clock_expanded* option. In this example the 'DC2' clock is a generated clock from the 'DC1' clock, which is a generated clock from CLK. When the *-full_clock_expanded* option is used, the path between the primary 'CLK' clock and the generated 'DC2' clock is shown going through register levels and through other generated clocks. This helps in debugging clock source latency values.

```

pt_shell> report_timing -path full_clock_expanded -group DC2
*****
Report : timing
        -path full_clock_expanded
        -delay max
        -max_paths 1
        -path_group DC2
Design : crpr07
Version: 2002.03-Beta3
Date   : Tue Jan 15 20:00:07 2002
*****

```

```

Startpoint: ff3 (rising edge-triggered flip-flop clocked by DC2)
Endpoint: ff3 (rising edge-triggered flip-flop clocked by DC2)
Path Group: DC2
Path Type: max

```

Point	Incr	Path

clock DC2 (rise edge)	0.00	0.00
clock CLK (source latency)	163.00	163.00
clk (in)	0.00	163.00 r

u1/Y (inv1a4) (gclock source)	83.32	246.32	f
u1/Y (inv1a4)	0.00	246.32	f
u2/Y (buf1a4)	340.72	587.04	f
u3/Y (inv1a2)	95.79	682.83	r
u4/A (buf1a2) (gclock source)	0.00	682.83	r
u4/Y (buf1a2)	324.15	1006.98	r
ff3/CLK (fd1a1)	0.00	1006.98	r
ff3/CLK (fd1a1)	0.00	1006.98	r
ff3/Q (fd1a1)	939.28	1946.26	f
u10/Y (or4b1)	990.35	2936.62	f
ff3/D (fd1a1)	0.00	2936.62	f
data arrival time		2936.62	

(The required path not shown)

The following example shows a timing report with the `-derate` option specified

pt_shell> **report_timing -derate -from ff1/CP -to ff2/D -delay_type min**

```
*****
Report : timing
        -path full_clock_expanded
        -delay min
        -input_pins
        -max_paths 1
Design : simple_2ff
Version: V-2004.06-Alpha1
Date   : Thu Mar 25 06:45:20 2004
*****
```

```
Startpoint: ff1 (rising edge-triggered flip-flop clocked by clk)
Endpoint: ff2 (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min
```

Point	Derate	Incr	Path

clock clk (rise edge)		0.0000	0.0000
clock source latency		0.0000	0.0000
clk (in)		0.0000	0.0000 r
u1/A (IBUF1)	0.8000	0.0000	0.0000 r
u1/Z (IBUF1)	0.8000	0.6814	0.6814 r
ff1/CP (FD1)	0.8000	0.0000	0.6814 r
ff1/CP (FD1)	0.8000	0.0000	0.6814 r
ff1/Q (FD1)	1.0000	1.2927	1.9740 r
u3/A (IV)	1.0000	0.0000	1.9740 r
u3/Z (IV)	1.0000	0.2319	2.2059 f
ff2/D (FD1)	1.0000	0.0000	2.2059 f
data arrival time			2.2059
clock clk (rise edge)		0.0000	0.0000

clock source latency	0.0000	0.0000
clk (in)	0.0000	0.0000 r
u1/A (IBUF1)	1.2000	0.0000 0.0000 r
u1/Z (IBUF1)	1.2000	1.0221 1.0221 r
u2/A (IBUF1)	1.2000	0.0000 1.0221 r
u2/Z (IBUF1)	1.2000	0.8167 1.8388 r
ff2/CP (FD1)	1.2000	0.0000 1.8388 r
clock reconvergence pessimism	-0.3407	1.4981
library hold time	1.0000	0.4000 1.8981
data required time		1.8981

data required time		1.8981
data arrival time		-2.2059

slack (MET)		0.3078

Derate Summary Report

total derate : required time	-0.3065
total derate : arrival time	0.1703

total derate : slack	0.4768
slack (with derating applied) (MET)	0.3078
clock reconvergence pessimism (due to derating)	-0.3407

slack (with no derating) (MET)	0.4440

1

The following example shows a timing report where dynamic clock latency has been specified using the **set_clock_latency** command.

```
*****
Report : timing
        -path_type full_clock_expanded
        -delay_type max
        -max_paths 1
Design : latency
Version: Z-2007.06-DEV
Date   : Tue Mar 20 08:14:26 2007
*****
```

```
Startpoint: ff1 (rising edge-triggered flip-flop clocked by clk)
Endpoint: ff2 (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max
```

Point	Incr	Path

clock clk (rise edge)	0.00	0.00
clock source latency	3.00	3.00
clock source latency (dynamic)	0.60	3.60

clk (in)	0.00	3.60 r
u1/Z (IBUF1)	0.85	4.45 r
ff1/CP (FD1)	0.00	4.45 r
ff1/Q (FD1)	1.44	5.89 f
u3/Z (IV)	0.58	6.47 r
ff2/D (FD1)	0.00	6.47 r
data arrival time		6.47
clock clk (rise edge)	10.00	10.00
clock source latency	1.00	11.00
clock source latency (dynamic)	-0.50	10.50
clk (in)	0.00	10.50 r
u1/Z (IBUF1)	0.85	11.35 r
u2/Z (IBUF1)	0.68	12.03 r
ff2/CP (FD1)	0.00	12.03 r
clock reconvergence pessimism	2.00	14.03
library setup time	-0.80	13.23
data required time		13.23

data required time		13.23
data arrival time		-6.47

slack (MET)		6.76

The following example shows a timing report with the *-exceptions all* option specified

```

pt_shell> report_timing -from ff1/CP -to ff3/D -exceptions all
*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : test
Version: Y-2006.06-Beta2-DEV
Date   : Tue Apr 11 15:30:06 2006
*****

Startpoint: ff1 (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ff3 (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point                               Incr      Path
-----
ff1/CP (FD1)                         0.00      0.00 r
ff1/Q (FD1)                         1.43      1.43 f
inv1/Z (IV)                         0.54      1.97 r
and1/Z (AN2)                        0.80      2.78 r

```

inv3/Z (IV)	0.25	3.02 f
ff3/D (FD1)	0.00	3.02 f
data arrival time		3.02
max_delay	3.00	3.00
library setup time	-0.80	2.20
data required time		2.20

data required time		2.20
data arrival time		-3.02

slack (VIOLATED)		-0.83

The dominant exceptions are:

From	Through	To	Setup	Hold

*	inv3/Z	*	max=3	

The overridden exceptions are:

From	Through	To	Setup	Hold

*	inv1/Z	*	max=5	

1

If the *-pba_mode exhaustive* option is specified, path-based analysis is used during the path search, and the worst recalculated paths meeting the user’s criteria are returned. This option requires that a path search be performed to ensure that the paths being returned truly are the worst paths. The additional runtime required for this option depends on the amount of timing improvement resulting from path-based analysis. As the timing improvement from path-based analysis increases, the runtime for the path search will increase. Large *-nworst* values can significantly increase the time needed for the search.

To see the worst path in the design with path-based analysis applied:

```
pt_shell> report_timing -pba_mode exhaustive
```

To report all endpoints in the design which fail after considering path-based analysis:

```
pt_shell> report_timing -pba_mode exhaustive -slack_lesser_than 0 -max_paths 1000
```

SEE ALSO

```
report_constraint(2)
report_delay_calculation(2)
```

```
set_case_analysis(2)
get_timing_paths(2)
report_default_significant_digits(3)
si_enable_analysis(3)
true_delay_prove_false_backtrack_limit(3)
true_delay_prove_true_backtrack_limit(3)
timing_report_always_use_valid_start_end_points(3)
```

report_timing_derate

Reports derate factors annotated on the current design.

SYNTAX

```
int report_timing_derate
    [-include_inherited]
    [-variation | -aocvm_guardband]
    [-significant_digits digits]
    [-nosplit]
    list object_list
```

```
int      digits
```

ARGUMENTS

-include_inherited

Indicates that the derate factors reported on each object should also include those set by other objects in the design.

-variation

Indicates that only variation derate factors should be reported. If neither the **-variation** nor the **-aocvm_guardband** options are specified, then only deterministic derate factors are reported. The **-variation** and **-aocvm_guardband** options are mutually exclusive.

-aocvm_guardband

Indicates that only guardband derate factors should be reported. If neither the **-variation** nor the **-aocvm_guardband** options are specified, then only deterministic derate factors are reported. The **-variation** and **-aocvm_guardband** options are mutually exclusive.

-significant_digits *digits*

Specifies the number of digits after the decimal point to be displayed for values in the report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable. Use this option if you want to override the default.

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

object_list

Specifies a list of cells, lib_cells, nets in the current design.

DESCRIPTION

Reports the derate factors either on the design, specific instances (cells or

library cells) contained in the design or on specific nets contained in the design. If this command is successful a boolean TRUE (or 1) will be returned, otherwise it will return FALSE (0).

If this command is called with no arguments then a separate report will be invoked for the derate factors stored on the top design (hereafter referred to as global derate factors) and also for each instance and net that has derate factors set on it.

If this command is called with the boolean argument *-include_inherited* then the derate report for each instance will also contain derate factors set from other objects in the design that were inherited by the instance being reported. Examples of this would be a leaf cell inheriting cell derate factors set on the hierarchical cell to which it belongs, or a leaf cell inheriting a derate factor set on the library cell of which it is an instantiation. In addition to the above each row containing an inherited derate factor will also contain a column entry indicating the object from which it was inherited. The argument *-include_inherited* will not have any effect on derate reports for nets.

If the command is called with the *object_list* specified then derate reports will only be issued for the instances or nets specified in the object list. If no derate factors exist on any of the objects specified then a warning will be issued for that object. If the boolean argument *-include_inherited* is also specified then inherited derate factors will be reported as outlined in the above paragraph.

EXAMPLES

In the following example derate factors have only been set globally (on the design), therefore only global derate factors will be reported.

```

pt_shell> set_timing_derate -data -early 0.92
pt_shell> set_timing_derate -data -late 1.14
pt_shell> set_timing_derate -clock -early 0.75
pt_shell> set_timing_derate -clock -late 1.26
pt_shell> report_timing_derate
*****
Report : timing derate
Design : simple_path
*****

Data -----
                                ----- Clock -----
                                Rise                Fall
                                Early   Late         Early   Late
                                -----
                                Early   Late         Early   Late
-----
design: simple_path
  Net delay static      0.75    1.26    0.75    1.26    0.92    1.14
0.92    1.14
  Net delay dynamic     0.75    1.26    0.75    1.26    0.92    1.14
0.92    1.14
  Cell delay            0.75    1.26    0.75    1.26    0.92    1.14

```

```

0.92      1.14
  Cell check          --          --          --          --          --          --
  --          --

1

```

In the following example derate factors have been set on the design named 'top'. More restrictive derates have been set on the hierarchical block name 'H1'. The derates that apply to the hierarchical cell named 'H1/u5' are reported.

```

pt_shell> set_timing_derate -early 0.95
pt_shell> set_timing_derate -late 1.06
pt_shell> set_timing_derate -early 0.82 [get_cell H1]
pt_shell> set_timing_derate -late 1.12 [get_cell H1]
pt_shell> report_timing_derate -include_inherited [get_cells H1/u5]
*****
Report : timing derate
        -include_inherited
Design : top
*****

```

Data -----		----- Clock -----					
		Rise		Fall		Rise	
		Early	Late	Early	Late	Early	Late
Fall							
Early	Late						

cell (hier): H1/u5							
Net delay static		0.95	1.06	0.95	1.06	0.95	1.06
0.95	1.06						
Inherited		top	top	top	top	top	top
top	top						
Net delay dynamic		0.95	1.06	0.95	1.06	0.95	1.06
0.95	1.06						
Inherited		top	top	top	top	top	top
top	top						
Cell delay		0.82	1.12	0.82	1.12	0.82	1.12
0.82	1.12						
Inherited		H1	H1	H1	H1	H1	H1
H1	H1						
Cell check		--	--	--	--	--	--
--	--						
Inherited		--	--	--	--	--	--
--	--						

1

SEE ALSO

set_timing_derate (2),


```
reset_timing_derate (2),  
timing_derate_report_compatibility (3).
```

report_transitive_fanin

Reports logic in fanin of specified sink objects.

SYNTAX

```
string report_transitive_fanin [-nosplit] -to sink_list [-trace_arcs arc_types]  
list sink_list
```

ARGUMENTS

-nosplit

Does not split lines if column overflows.

-to *sink_list*

Specifies a list of sink pins, ports, or nets in the design, whose transitive fan-in is reported. If a net is specified, the effect is the same as listing all driver pins on the net.

-trace_arcs arc_types

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (the default), which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); **enabled**, which permits the tracing of all enabled arcs and disregards case analysis values; and **all**, which permits the tracing of all combinational arcs regardless of either case analysis or arc disabling. (Note that the **enabled** option corresponds to the default behavior of this command in releases of PrimeTime prior to 2006.12.)

DESCRIPTION

The command produces a report showing the transitive fanin of specified pins, ports, or nets in the design. A pin is considered to be in the transitive fanin of a sink if there is a timing path through combinational logic from the pin to that sink (please also see the **-trace_arcs** option). The fanin report stops at the clock pins of registers (sequential cells).

If the **current_instance** is set, the report will be focussed on the fanin within the **current_instance**. The report stops at the boundaries of the **current_instance**, and no paths outside the **current_instance** are reported. The report shows the hierarchical boundary pins on the current instance.

EXAMPLES

The following example shows the transitive fanin of a pin in the design.

```
pt_shell>report_transitive_fanin -to FF1/D
```

```
*****
```

```
Report : transitive_fanin
```

```
Design : top
```

```
report_transitive_fanin
```

Version: 1997.01-development
Date : Tue Aug 6 14:53:47 1996

Fanin network of sink 'FF1/D':

Driver Pin	Load Pin	Type	Sense
-----	-----	-----	-----
gate1/Y	FF1/D	(net arc)	same
Load Pin	Driver Pin	Type	Sense
-----	-----	-----	-----
gate1/A	gate1/Y	AN2	same
gate1/B	gate1/Y	AN2	same
Driver Pin	Load Pin	Type	Sense
-----	-----	-----	-----
IN1	gate1/A	(net arc)	same
IN2	gate1/B	(net arc)	same

SEE ALSO

current_design (2), **current_instance** (2), **report_clock** (2), **report_timing** (2),
report_transitive_fanout (2).

report_transitive_fanout

Reports logic in fanout of specified sources.

SYNTAX

```
string report_transitive_fanout [-nosplit] -clock_tree -from source_list [-  
trace_arcs arc_types]  
list source_list
```

ARGUMENTS

-nosplit

Does not split lines if column overflows.

-clock_tree

Reports transitive fanout of all clock sources in the design.

-from *source_list*

Specifies a list of source pins, ports, and nets in the design whose transitive fanout is reported. If a net is specified, the effect is same as listing all the load pins on the net.

-trace_arcs *arc_types*

Specifies the type of combinational arcs to trace during the traversal. Allowed values are **timing** (the default), which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); **enabled**, which permits the tracing of all enabled arcs and disregards case analysis values; and **all**, which permits the tracing of all combinational arcs regardless of either case analysis or arc disabling. (Note that the **enabled** option corresponds to the default behavior of this command in releases of PrimeTime prior to 2006.12.)

DESCRIPTION

Produces a report showing the transitive fanout of specified pins or ports in the design. A pin is considered to be in the transitive fanout of a source if there is a timing path through combinational logic from the source to that pin (please also see the **-trace_arcs** option). The fanout report stops at the inputs to registers (sequential cells).

If the **current_instance** is set, the report focuses on the fanout within the **current_instance**. The report stops at the boundaries of the **current_instance**, and no paths outside the **current_instance** are reported. The report also shows the hierarchical boundary pins on the **current_instance**.

EXAMPLES

The following example shows the transitive fanout of a pin in the design.

```
pt_shell> report_transitive_fanout -from FF1/Q
```

```

*****
Report : transitive_fanout
Design : top
Version: 1997.01-development
Date   : Tue Aug  6 15:19:19 1996
*****

```

Fanout network of source 'FF1/Q':

Driver Pin	Load Pin	Type	Sense
FF1/Q	gate5/A	(net arc)	same

Load Pin	Driver Pin	Type	Sense
gate5/A	gate5/Y	AN2	same

Driver Pin	Load Pin	Type	Sense
gate5/Y	OUT2	(net arc)	same

The following command shows the transitive fanout of all the clock sources in the design.

```
pt_shell> report_transitive_fanout -clock_tree
```

```

*****
Report : transitive_fanout
        -clock_tree
Design : top
Version: 1997.01-development
Date   : Tue Aug  6 15:24:00 1996
*****

```

Fanout network of source 'CLK':

Driver Pin	Load Pin	Type	Sense
CLK	FF3/CLK	(net arc)	same
CLK	FF2/CLK	(net arc)	same
CLK	FF1/CLK	(net arc)	same

SEE ALSO

```

create_clock (2), current_design (2), current_instance (2), report_clock (2),
report_timing (2), report_transitive_fanin (2).

```

report_units

Reports the unit information.

SYNTAX

```
string report_units  
[-nosplit]
```

ARGUMENTS

-nosplit

Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

DESCRIPTION

The **report_units** command displays the units used by the current_design. To generate the report the design should be loaded and linked to a library. The units are always reported in reference to the MKS units like Farad, Amp, Ohm, Second and Volt. The resistance unit is inferred from the time and capacitance unit. Both time and capacitance unit are required for PrimeTime. If either of them is missing the units is undefined and it is reported as "Not specified". The most common cause of this problem is the library is very old format it needs to be recompiled with the latest **Library Compiler**.

If power analysis is on, the power units will also be reported.

EXAMPLES

```
pt_shell> report_units  
  
*****  
Report : units  
Design : simple  
Version: X-2005.06-Beta2  
Date   : Mon Apr 28 11:55:27 2005  
*****  
  
Units  
-----  
Capacitive_load_unit      : 1e-12 Farad
```

Current_unit	: 0.001 Amp
Resistance_unit	: 1000 Ohm
Time_unit	: 1e-09 Second
Voltage_unit	: 1 Volt

SEE ALSO

`read_db (2)`, `set_min_library (2)`.

report_user_sensitization

Report user sensitization of an instance or library arc for write_spice_deck output.

SYNTAX

```
int report_user_sensitization
[-analysis_type [rise | fall | high | low]]
[-arc arcs_list]
[-library library_name]
[-design design_name]
```

```
list arcs_list
string library_name
string design_name
```

ARGUMENTS

`[-analysis_type [rise | fall | high | low]]`
Specifies the final state of the output pin of timing or library arc.

`[-arc arcs_list]`
Specifies a list timing or library arcs with their annotated user sensitization to be reported. You can get this argument with the **get_timing_arcs** or **get_timing_lib_arc** comments.

`[-library library_name]`
Specifies a list of input pins of the gate that contains the arc. It must includes the input pin of the arc in the **arcs_list**. The sensitization vectors/sequences are applied to these pins to achieve the desirable state indicated in the **-analysis_type** options.

`[-design design_name]`
Specifies the sensitization vectors/sequences to sensitize the arc. It is expressed as a list of **1**, **0**, **r**, and **f**. They stand for the states of logic one, logic zero, rising, and falling. The number of stats must be a multiple of the number of pin in the **-event_pins** options.

DESCRIPTION

It reports user sensitization of a library, a design, an instance arc collection or a library arc collection for write_spice_deck output.

EXAMPLES

Sensitization of a MUX with Falling S to rising X pin.

```
pt_shell> set arcs_1 [ get_timing_arcs -from inp_victim/S -to inp_victim/X ]
pt_shell> set_user_sensitization -analysis_type rise -event_pin { D0 D1 S } \
?      -event_states { r f 1 1 0 f } $arcs_1
*****
Report : set_user_sensitization
```



```

    -analysis_type rise
    -event_step 0.05
    -event_pins D0 D1 S
    -event_states R F 1 1 0 F
    timing_arc S -> X
    when = D0 * !(D1)
    sense = negative_unate
Design : SDN_MUX2_1_X_S
*****

```

```

pt_shell> report_user_sensitization -analysis_type rise -arc $arcs_1
User sensitization of timing arc inp_victim/S -> inp_victim/X
    Sense = negative_unate, when = (D0 * !(D1)), analysis_type = rise.
    Minimum transition separation time = 0.05 (LU).
    Event pin : inp_victim/D0 inp_victim/D1 inp_victim/S
    States : R F 1
             1 0 F
Information: 1 user sensitization reported. (SPICE-118)

pt_shell> remove_user_sensitization -analysis_type rise -arc $arcs_1
Information: 1 user sensitization removed. (SPICE-118)

```

SEE ALSO

```
write_spice_deck (2); set_user_sensitization (2); report_user_sensitization (2);
```

report_variation

Reports variation for timing paths, cells, nets, library cells, and current design.

SYNTAX

```
string report_variation
[-verbose]
[-delay_type delay_type]
[-clock_network]
[-slack_lesser_than slack_limit]
[-nworst num_worst_cells]
[-input_pins]
[-transition]
[-all_cells]
[-all_nets]
[-parametric_sensitivity parameter_list]
[-significant_digits digits]
[-nosplit]
[collection1]
```

```
string      delay_type
float       slack_limit
int         num_worst_cells
list        parameter_list
int         digits
collection  collection1
```

ARGUMENTS

- verbose
Enables verbose reporting format for timing_path, cells, or net collections.
- delay_type *delay_type*
Specifies the type of delay or slew to be reported in the cell/net variation report. Valid values are **max** (the default), **min**, **max_rise**, **max_fall**, **min_rise**, or **min_fall**.
- clock_network
Filters cells/nets in clock networks to be reported in the cell/net variation report. If this option is not specified, the command will only report cells/nets in data networks.
- slack_lesser_than *slack_limit*
Indicates that only those arcs or pins with a slack less than *slack_limit* are to be shown in the cell/net variation report.
- nworst *num_worst_cells*
Specifies the number of most sensitive cells/cells to be reported in the cell/net variation report. The default value is 20.
- input_pins
Indicates that input pins are to be shown in the verbose timing_path report.

-transition
Reports pin slew sensitivity in the cell/net variation report.

-all_cells
Reports all cells in the cell variation report, bypassing the requirement to pass a cell collection as the argument of the command.

-all_nets
Reports all nets in the net variation report, bypassing the requirement to pass a net collection as the argument of the command.

-parametric_sensitivity *parameter_list*
Reports parameter-specific sensitivities for nets.

-significant_digits *digits*
Specifies the number of digits after the decimal point to be displayed for time values in the generated report. Allowed values are 0-13; the default is determined by the **report_default_significant_digits** variable, whose default value is 2. Use this option if you want to override the default. This option controls only the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

-nosplit
Most of the design information is listed in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line-splitting and facilitates writing software to extract information from the report output.

collection1
Specifies the collection to report. The collection type can be *timing_path*, *cell*, *net*, *lib_cell* or *design*.

DESCRIPTION

Provides a report of variation-aware information in paths, cells, nets, library cells, or current design.

EXAMPLES

The following example shows a variation report of a recalculated timing path. The columns correspond to statistical slack attributes quantile, mean and standard deviation of the timing path. The "sensitivity" field displays the sensitivity value of the statistical slack, which equals to $\text{stddev} / \text{mean} * 100$.

```
pt_shell> report_variation $path -significant_digits 6
```

```
*****
Report : variation
Design : test
Version: Y-2006.06-DEV
```

Date : Mon Mar 27 15:29:33 2006

	Path	startpoint	endpoint	quantile	mean	stddev	sensitiv

----	0	ff1	ff2	9.868649	9.886533	0.007124	0.0
72059							

The following example shows a variation report of a timing path with verbose output showing statistical data of each timing point. The detailed report shows additional attribute of the timing_path, along with statistical arrival of each timing_point in launch clock path plus delay path, and the capture clock path (separated by an empty line) of the timing_path. The "sensitivity" fields display the sensitivity values of the corresponding statistical attributes. The "incr" field shows the incremental standard deviation compared to the previous stage. The asterisk "*" adjacent to a pin name indicates a critical stage that contributes to the most significant incremental standard deviation in the path.

pt_shell> report_variation \$path -verbose -significant_digits 6

 Report : variation
 Design : test
 Version: Y-2006.06-DEV
 Date : Mon Mar 27 15:29:33 2006

	Path	startpoint	endpoint	quantile	sensitivity	mean	stddev

----	0	ff1	ff2	9.868649	0.072059	9.886533	0.0
07124							

	Path attribute	quantile	sensitivity	mean	stddev

	arrival	0.219282	3.440833	0.202071	0.006953
	slack	9.868649	0.072059	9.886533	0.007124
	required	10.084497	0.018024	10.088605	0.001818
	startpoint_clock_latency	0.065879	2.346163	0.062349	0.001463
	endpoint_clock_latency	0.089497	1.942685	0.093604	0.001818

	Point arrival	quantile	sensitivity	mean	stddev

-----	clk (in)	0.000000	0.000000	0.000000	0.000000
r					
	b01/				
ZN (INVD1)	0.009146	2.494643	0.008588	0.000214	0.000214 f
	b02/				

ZN (INVD1) *	0.033515	3.722505	0.030891	0.001150	0.000936	r
	b11/					
ZN (INVD1)	0.048738	2.773270	0.046092	0.001278	0.000128	f
	b12/					
ZN (INVD1)	0.065879	2.346163	0.062349	0.001463	0.000185	r
	ff1/					
CP (DFD1)	0.065879	2.346163	0.062349	0.001463	0.000000	r
	ff1/					
Q (DFD1) *	0.200052	3.751771	0.183375	0.006880	0.005417	r
	b21/					
ZN (INVD1)	0.219282	3.440833	0.202071	0.006953	0.000073	f
	ff2/					
D (DFD1)	0.219282	3.440833	0.202071	0.006953	0.000000	f
	clk (in)		0.000000	0.000000	0.000000	0.000000
r						
	b01/					
ZN (INVD1)	0.008154	2.494643	0.008588	0.000214	0.000214	f
	b02/					
ZN (INVD1) *	0.028676	3.722505	0.030891	0.001150	0.000936	r
	b31/					
ZN (INVD1)	0.043222	3.016317	0.046120	0.001391	0.000241	f
	b32/					
ZN (INVD1)	0.061246	2.358874	0.064573	0.001523	0.000132	r
	b33/					
ZN (INVD1)	0.074507	2.147924	0.078099	0.001678	0.000154	f
	b34/					
ZN (INVD1)	0.089497	1.942685	0.093604	0.001818	0.000141	r
	ff2/					
CP (DFD1)	0.089497	1.942685	0.093604	0.001818	0.000000	r

The following example reports the library sensitivity of a collection of library cells. The report shows min, average and max sensitivity values of each variation source for all timing arcs of a library cell. "r" and "f" denotes rise and fall of the timing sense respectively.

```
pt_shell> report_variation $libcell -significant_digits 6
```

```
*****
Report : variation
Design : test
Version: Y-2006.06-DEV
Date   : Mon Mar 27 15:34:20 2006
*****
```

Library sensitivity for cell: INVD1

delay sensitivity for cell INVD1 arc I->ZN		
timing sense	r negative_unate	f negative_unate
len: min/avg/max	0.000127/0.001236/0.003896	0.000066/0.000678/0.002429
vth: min/avg/max	0.040482/0.409654/1.425395	0.021974/0.286770/1.051886

```

slew sensitivity for cell INVD1 arc I->ZN
timing sense          r negative_unate          f negative_unate
-----
len: min/avg/max    -0.000214/0.000842/0.003220  -0.000149/0.000278/0.001265
vth: min/avg/max     0.012500/0.268289/0.972281  -0.062368/0.102758/0.465702

```

The following example shows the variation libraries and all random variables that are present in the current design.

```
pt_shell> report_variation [current_design]
```

```

*****
Report : variation
Design : inv
Version: Y-2006.06-DEV
Date   : Mon Mar 27 15:52:56 2006
*****

```

Variation libraries:

library	len	vth
tt.db	100.00	0.23
tt_minlen.db	95.00	0.23
tt_maxlen.db	105.00	0.23
tt_minvth.db	100.00	0.22
tt_maxvth.db	100.00	0.24

Cell variations:

parameter	name	type	mean	s
tddev				

len	Ldie	normal	0.00	
1.00	Lfab	normal	0.00	
1.80	Lref	normal	0.00	
0.50	len	normal	0.00	
3.33	vth	normal	0.00	
0.02				

Cell variation for arc: inv3/I->inv3/ZN

parameter	name	type	mean	s
tddev				

0.10	len	Ldie_inv3	normal	0.00
0.00	vth	vth_inv3	constant	0.00

Parasitic variations:

	parameter	name	type	mean	s
tddev	-----				
----	METAL1_WIDTH	METAL1_WIDTH	normal	0.00	
0.15	METAL2_THICKNESS	METAL2_THICKNESS	normal	0.00	
0.15					

Correlations specified for design:

Auto-correlations:

variation	correlation

Ldie	1.000
Lfab	1.000
Lref	1.000
len	1.000
vth	0.000
Ldie_inv3	1.000
vth_inv3	1.000
METAL1_WIDTH	1.000
METAL2_THICKNESS	1.000

Cross-correlations name: inv3_arc_cross

variation	variation	correlation

Ldie_inv3	vth_inv3	0.414

Cross-correlations name: para_width_thickness_cross

variation	variation	correlation

METAL1_WIDTH	METAL2_THICKNESS	1.000

The following example shows the variation report of a collection of cells. The columns correspond to the worst slack of all arcs of each cell, scalar delay, mean and standard deviation of the statistical delay of one arc, which has the largest delay standard deviation value among all arcs of the cell. The cells are ordered by standard deviation values. In order to show slack values in cell/net variation report, the variable "timing_save_pin_arrival_and_slack" must be set to true prior to update_timing.

```
pt_shell> set timing_save_pin_arrival_and_slack true
pt_shell> update_timing
pt_shell> report_variation [get_cells -hierarchical]
```

```
*****
Report : variation
        -delay_type max
Design : test
Version: Z-2006.12-DEV
Date   : Thu Oct 19 14:28:37 2006
*****
```

cell	lib_cell	slack	quantile	mean	stddev
u13	AN2D1	0.5449	0.1156	0.0965	0.0073
u24	OR2D1	0.4962	0.0906	0.0758	0.0056
u02	BUFFD1	0.4962	0.0873	0.0752	0.0047

The following example shows the verbose output of the variation report of a cell collection. The report shows arc sense, slack, and variation data on all cell arcs. "r" or "f" denotes the cell arc rises or falls at the output pin. The asterisk "*" adjacent to the last column indicates an arc with the worst delay standard deviation among all arcs within a cell.

```
pt_shell> report_variation [get_cells -hierarchical] -verbose
```

```
*****
Report : variation
        -delay_type max
        -verbose
Design : test
Version: Z-2006.12-DEV
Date   : Thu Oct 19 14:28:37 2006
*****
```

```
Cell delay variation: u13 (AN2D1)
from      to      sense      slack  quantile  mean  stdd
-----
-
  u13/A1    u13/
Z  r positive_unate  0.5557  0.1098  0.0924  0.0066
  u13/A1    u13/
Z  f positive_unate  0.5592  0.0958  0.0823  0.0054
  u13/A2    u13/
Z  r positive_unate  0.5449  0.1156  0.0965  0.0073 *
  u13/A2    u13/
Z  f positive_unate  0.5491  0.1043  0.0899  0.0059

Cell delay variation: u24 (OR2D1)
```


	from	to	sense	slack	quantile	mean	stdd
ev	-----						
-							
	u24/A1	u24/					
Z	r positive_unate		0.5443	0.0906	0.0758	0.0056	*
	u24/A1	u24/					
Z	f positive_unate		0.5737	0.0869	0.0755	0.0048	
	u24/A2	u24/					
Z	r positive_unate		0.5353	0.0840	0.0718	0.0048	
	u24/A2	u24/					
Z	f positive_unate		0.4962	0.0963	0.0835	0.0055	
Cell delay variation: u02 (BUFFD1)							
	from	to	sense	slack	quantile	mean	stdd
ev	-----						
-							
	u02/I	u02/					
Z	r positive_unate		0.5353	0.0783	0.0692	0.0038	
	u02/I	u02/					
Z	f positive_unate		0.4962	0.0873	0.0752	0.0047	*

The following example shows the variation report of all nets whose worst slack is less than 0. Option "-all_nets" bypasses the requirement to pass a net collection to the command. The reporting format is similar to the cell variation report.

```
pt_shell> report_variation -all_nets -slack_lesser_than 0
```

```
*****
```

```
Report : variation
```

```
-delay_type max
```

```
-slack_lesser_than 0.000000
```

```
-all_nets
```

```
Design : test
```

```
Version: Z-2006.12-DEV
```

```
Date : Wed Nov 15 11:39:33 2006
```

```
*****
```

net	slack	quantile	mean	stddev
udat/in3	-4.0884	0.0100	0.0071	0.0012
udat/w5	-3.7312	0.0088	0.0071	0.0007
udat/w8	-3.6901	0.0067	0.0053	0.0006
udat/in1	-3.5159	0.0064	0.0052	0.0005

```
pt_shell> report_variation -all_nets -slack_lesser_than 0 -verbose
```

```
*****
```

```
Report : variation
```

```
-delay_type max
```

```

        -slack_lesser_than 0.000000
        -all_nets
        -verbose
Design : test
Version: Z-2006.12-DEV
Date   : Wed Nov 15 11:44:19 2006
*****

```

Net delay variation: udat/in3

from	to		slack	quantile	mean	stddev
ffi3/Q	udat/u4/A1	r	-2.9094	0.0100	0.0071	0.0012 *
ffi3/Q	udat/u4/A1	f	-2.1000	0.0092	0.0069	0.0009
ffi3/Q	udat/u2/A1	r	-3.9189	0.0094	0.0071	0.0010
ffi3/Q	udat/u2/A1	f	-4.0884	0.0088	0.0070	0.0008

Net delay variation: udat/w5

from	to		slack	quantile	mean	stddev
udat/u1/ZN	udat/u6/A2	r	-2.7033	0.0083	0.0071	0.0005
udat/u1/ZN	udat/u6/A2	f	-3.7312	0.0082	0.0071	0.0005
udat/u1/ZN	udat/u4/A2	r	-3.6901	0.0088	0.0071	0.0007 *
udat/u1/ZN	udat/u4/A2	f	-3.5759	0.0086	0.0071	0.0007

Net delay variation: udat/w8

from	to		slack	quantile	mean	stddev
udat/u4/Z	udat/u5/A2	r	-3.6901	0.0067	0.0053	0.0006 *
udat/u4/Z	udat/u5/A2	f	-3.5759	0.0065	0.0052	0.0006

Net delay variation: udat/in1

from	to		slack	quantile	mean	stddev
ffi1/Q	udat/u1/A1	r	-3.5142	0.0063	0.0051	0.0005
ffi1/Q	udat/u1/A1	f	-3.5159	0.0064	0.0052	0.0005 *

The following example shows parameter-specific sensitivities of a net. The interconnect variation parameters are sorted by parametric sensitivity values, which represent each individual parameter's contribution to the standard deviation of the net delay distribution. The parametric sensitivity values are non-negative. If a collection of nets is given, the report will order nets by the most dominant parametric sensitivity value of each net.

```
pt_shell> report_variation -parametric_sensitivity * [get_nets n1]
```

```

*****
Report : variation
        -delay_type max
        -parametric_sensitivity { * }
Design : test
Version: Z-2007.06-DEV

```

Date : Fri Mar 2 17:51:28 2007

net	parameter	sensitivity
n1	metal2_T	0.035652
	metal2_W	0.026661
	metal3_T	0.025091
	metal3_W	0.017360
	metal1_T	0.015018
	metal1_W	0.011877

SEE ALSO

variation_enable_analysis (3), **variation_analysis_mode** (3),
timing_save_pin_arrival_and_slack (3), **set_variation_library** (2), **set_variation** (2),
get_cells (2), **get_nets** (2), **get_lib_cells** (2), **current_design** (2).

report_vcd_hierarchy

Reports the hierarchy in the VCD file.

SYNTAX

```
string report_vcd_hierarchy [-pipe_exec command] [-full_path] [-find patten]  
[file_name]  
string command  
string file_name  
string patten
```

ARGUMENTS

file_names

Specifies a file in VCD format from which the switching activity information is to be read. If *file_name* ends with .gz, .Z, .vpd and .fsdb, it is read as a gzipped file, a compressed file, a VCD+ file, and a FSDB file, respectively, otherwise it is just read as a normal ASCII VCD file. If *file_name* is omitted, the name will be token from the read_vcd command if it exists, otherwise an error message will be issued.

-pipe_exec command

Specifies a shell command which is used to generate the VCD file *file_names*. This option will invoke the command and directly pipe the output VCD file to PrimeTime-PX. In another word, the simulation and power analysis are in parallel run. No VCD disk file is generated at all.

-find patten

Report only those scopes whose last hierarchy name is matched the specified *patten*. They are always reported in full path format. Nothing will print out if not found.

-full_path

Displays each scopes in full path format.

DESCRIPTION

Displays the indented hierarchy in the specified VCD file.

In order to read in a gzipped VCD file, a compressed VCD file, a VCD+ file, and a FSDB file, user's PATH should include gunzip, uncompress, vpd2vcd, and fsdb2vcd. vpd2vcd and fsdb2vcd come with Synopsys VCS and Novas Debussy, respectively.

The pipe option allows PrimePower reading in VCD file directly from simulator, which avoids generating huge VCD file. The VCD file name in this scenario is just served as a pipe name. After running the VCD file does not exist. You don't need to change your existing test bench. Use it just like normal VCD dump (inserting \$dumpfile, \$dumpvars, etc.) You don't need to invoke the simulator, either. PrimePower will automatically run the simulation and directly read in the VCD output from the simulator.

SEE ALSO

`read_vcd` (2) ,

report_wire_load

Reports wire load information.

SYNTAX

```
string report_wire_load [-nosplit] [cell_names]  
list cell_names
```

ARGUMENTS

-nosplit
Does not split lines if the column overflows.

cell_names
Provides a list of hierarchical cells.

DESCRIPTION

This command displays the characteristics of wire load models set on the current design and on cells of the current design. By default (if cell list is not specified) the command displays the wire load models and wire load mode used in the current design. If a cell list is specified, the command displays the wire load models used in the specified cells.

EXAMPLES

This command displays the wire load model in the current design.

```
pt_shell> report_wire_load
```

```
*****  
Report : wire_load  
Design : top  
Version: 1997.01-alpha3.3  
Date   : Tue Aug  6 09:27:23 1996  
*****
```

Wire Loading Model Mode: segmented

Cell	Design	Wire_model	Library	Selection_type

--				
	inv_top	WLM1	my_lib	user-specified
cell1	tst	WLM2	my_lib	user-specified
1				

SEE ALSO

`set_wire_load_model (2)`, `set_wire_load_mode (2)`, `report_design (2)`,
`set_wire_load_selection_group (2)`, `set_wire_load_min_block_size (2)`,
`remove_wire_load_model (2)`, `remove_wire_load_selection_group (2)`.

reset_design

Removes user specified information from design.

SYNTAX

```
string reset_design [-timing]
```

ARGUMENTS

-timing

Resets the timing information on the current design.

DESCRIPTION

Removes all attributes from the design. Whether these attributes were created as user attributes, were created as a result of commands, or were read from design db does not make any difference. Attributes include but not limited to exceptions, input/output delays, clocks, etc.

EXAMPLES

The following command resets the attributes on the current design.

```
pt_shell> reset_design
```

SEE ALSO

current_design (2), **remove_user_attribute** (2),

reset_mode

Resets cell mode groups or design mode groups to the default state.

SYNTAX

```
Boolean reset_mode [-type cell | design]
[-group group_list]
[instance_list]
list    group_list
list    instance_list
```

ARGUMENTS

-type cell | design

Indicates the type of mode group to be set to default. This option has the following mutually-exclusive valid values: **design** and **cell**. The **cell** value specifies that cell mode groups are to be set to default. Cell mode groups are defined on library cells in the library. The **design** value specifies that design mode groups are to be set to default. Design mode groups are user specified using **define_design_mode_group**. If the **-type** option is ommitted from the command then this is equivalent to specifying **-type cell**

-group *group_list*

Specifies a list of mode groups, each of which is to be reset to its default setting. If **-type** has a cell value then the *group_list* must contain only cell mode groups. If **-type** has a design value then the *group_list* must contain only design mode groups.

instance_list

Specifies a list of instances for which the specified cell mode groups are to be set to default. If no instance list is specified then all moded cells are considered. This list must only be included with **-type cell**. No error occurs if you reset the modes on a cell that has no modes.

DESCRIPTION

Resets mode groups to the default state of all modes enabled. Cell mode groups are defined in the library. Each library cell can have a set of cell mode groups. Each of these cell mode groups will have two or more cell modes. Each of these cell modes can be mapped to a set of timing arcs of the library cell. When a cell mode group is set to default for a given instance of the library cell all of its modes are enabled for that cell. All timing arcs of the enabled modes also get enabled with respect to modes for that cell.

Cell mode groups can have different states due to **set_mode -type cell**, **set_mode -type design** and conditional evaluation. When conflict arises the following precedence rule is employed
 set_mode -type cell > **set_mode -type design** > evaluation of mode conditions.

To reset the state of the cell mode group due to **set_mode -type cell** use **reset_mode -type cell**. To reset the state of the cell mode group due to **set_mode -type design**

use either `reset_mode -type design` or `remove_design_mode`. To reset the state of the cell mode group due to conditional evaluation, use **`remove_case_analysis`** or edit the verilog netlist to remove logical constants.

Design modes and design mode groups are defined by the user with the **`define_design_group`** and **`map_design_mode`** commands. Design modes can be mapped to a set of cell modes and/or a set of paths. When a design mode group is reset to default all design modes of that group are enabled. When a design mode is enabled all paths mapped to the design mode are reset and all previous setting of cell modes mapped to the design modes are removed.

EXAMPLES

In the following example, we have two cell mode groups RW and SH defined on cell Uram1/D0. The first command sets READ as the active mode for RW and EARLY as the active mode for SH.

```
pt_shell> set_mode {EARLY READ} Uram1/D0
```

The following command can be employed to reset the cell mode groups to default.

```
pt_shell> reset_mode -group {RW SH} Uram1/D0
```

To reset all cell mode group in Uram1/D0 use the following command `pt_shell> reset_mode Uram1/D0`

To reset all cell mode groups in the design use `pt_shell> reset_mode`

In the following example, the first command defines two design modes, DM1 and DM2. (Since no mode group name was specified, the mode group is named "default".) The second command specifies that for design mode DM1, the READ cell mode is active for the RAM instance Uram1. The third command specifies that for design mode DM2, the WRITE cell mode is active for the RAM instance Uram1. The fourth command maps all paths ending at Uram1/D0 to the design mode DM2. The fifth command sets the design_mode DM1. The sixth command sets READ as the active cell mode and disables the WRITE cell mode.

```
pt_shell> define_design_mode_group {DM1 DM2}<br>
pt_shell> map_design_mode DM1 READ Uram1 <br>
pt_shell> map_design_mode DM2 WRITE Uram1<br>
pt_shell> map_design_mode DM2 -to Uram1/D0 <br>
pt_shell> set_mode -type design DM1<br>
pt_shell> set_mode -type cell READ Uram1<br>
```

The following example resets the design mode group "default".

```
pt_shell> reset_mode -type design -group default
```

This command enables the design_mode DM2 resulting in the resetting of all paths mapped to Uram1/D0. However the cell mode WRITE on Uram1/D0 does not get enabled since it was also disabled by command 5 above. To reset the cell mode group to default use

```
pt_shell> reset_mode -type cell Uram1
```

The following command is equivalent to the previous command

```
pt_shell> reset_mode Uram1
```

SEE ALSO

```
define_design_mode_group(2), remove_design_mode(2), report_mode(2),  
map_design_mode(2). set_mode(2). set_case_analysis(2).
```

reset_noise_parameters

Resets the noise analysis parameters for the current design.

SYNTAX

```
int reset_noise_parameters
```

DESCRIPTION

This command resets the parameters that are considered during the noise analysis. If this command is performed, the default settings are used for the noise analysis: beyond the rail analysis is ignored, arrival times of aggressors are considered, analysis effort level is high, and propagation of noise is disabled.

EXAMPLES

The following example resets the default noise parameters:

```
pt_shell> reset_noise_parameters
```

SEE ALSO

```
update_noise (2); report_noise (2); set_noise_parameters (2);  
report_noise_parameters (2); si_noise_effort_threshold_within_rails (3).  
si_noise_effort_threshold_within_rails (3).
```

reset_path

Resets specified paths to single-cycle behavior.

SYNTAX

Boolean **reset_path**

```
[-setup] [-hold]
[-rise] [-fall]
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
```

```
list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
```

ARGUMENTS

-setup

Indicates that only setup (maximum delay) evaluation is to be reset to its default, single-cycle behavior. If neither **-setup** nor **-hold** is specified, both setup and hold checking are reset to single-cycle.

-hold

Indicates that only hold (minimum delay) evaluation is to be reset to its default, single-cycle behavior. If neither **-setup** nor **-hold** is specified, both setup and hold checking are reset to single-cycle.

-rise

Indicates that only rising path delays are to be reset to single-cycle behavior. If neither **-rise** nor **-fall** is specified, both rising and falling delays are reset to single-cycle.

-fall

Indicates that only falling path delays are to be reset to single-cycle behavior. If neither **-rise** nor **-fall** is specified, both rising and falling delays are reset to single-cycle.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint

is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through *through_list*

Specifies a list of pins, ports, and nets through which the paths must pass that are to be reset. Nets are interpreted to imply the leaf-level driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

-rise_through *rise_through_list*

This option is similar to the **-through** option, but applies only to paths with a rising transition at the through points.

-fall_through *fall_through_list*

This option is similar to the **-through** option, but applies only to paths with a falling transition at the through points.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is

a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

DESCRIPTION

The command restores designated timing paths in the current design the default single-cycle behavior. Use **reset_path** to undo the effect of **set_multicycle_path**, **set_false_path**, **set_max_delay**, and **set_min_delay**. Attributes set by these commands are removed by **reset_path**.

Note that a general **reset_path** command removes the effect of matching general or specific point-to-point exception commands, as long as there is a matching object in the path specification of the original exception-setting commands. For example, the following command

reset_path -from {CLK}

resets more specific matching exceptions for the object CLK, such as

```
set_false_path -from {CLK} -to {d/Z}  
set_false_path -from {CLK} -to {g/Z}
```

If you do not specify either **-setup** or **-hold**, the default behavior is restored to both. The default is a setup relation of one cycle and a hold relation of zero cycles, so that hold is checked one active edge before the setup data at the destination register.

Setup and hold information is different for rising and falling transitions. In most cases, these are reset together. Use the **-rise** or **-fall** options to reset only one or the other. To disable setup or hold calculations for paths, use **set_false_path**.

The general and specific constraint options (for example, **-through** and **-rise_through** or **-fall_through**) are treated as different qualifiers. That is, if you issue **reset_path -fall_through**, only those constraints are removed that were set with the **-fall_through** option; any that were set with the **-through** option are not removed. The same applies to the general and specific **-to** and **-from** options. For an example, see the EXAMPLES section.

EXAMPLES

The following command resets the timing relation between ff1/CP and ff2/D to the default single-cycle behavior.

```
pt_shell> reset_path -from { ff1/CP } -to { ff2/D }
```

The following command resets only the rising delay to single cycle for all paths ending at latch2d.

```
pt_shell> reset_path -rise -to latch2d
```

The following example first sets a specific and a general point-to-point exception, then removes the exceptions.

```
pt_shell> set_multicycle_path 2 -from A -to Z
pt_shell> set_max_delay 15 -to Z
pt_shell> reset_path -to Z
```

In following example, the **reset_path** command removes only the max_delay constraints, because **-to** and **-rise_to** are treated as different qualifiers and do not match.

```
pt_shell> set_multicycle_path 2 -from A -to Z
pt_shell> set_max_delay 15 -rise_to Z
pt_shell> reset_path -rise_to Z
```

SEE ALSO

current_design (2), **report_constraint** (2), **reset_design** (2), **set_false_path** (2), **set_max_delay** (2), **set_min_delay** (2), **set_multicycle_path** (2).

reset_power_derate

Resets user specified power derating factors set either on a design or on a specified list of instances (cells or library cells).

SYNTAX

```
int reset_power_derate  
[-groups group_name_list]  
object_list
```

Data Type

```
list    group_name_list  
list    object_list
```

ARGUMENTS

-groups
Specifies the power group(s) in which the derate factors of all the cells will be reset.

object_list
Specifies current design or a list of cells or library cells on which the power derating factors will be reset to 1.0.

DESCRIPTION

Call this command with no arguments to reset all derate factors set on the design (to the default value, 1.0). This will include both derating factors set globally on the design and those set on specific instances in the design. If the command is called with a list of objects, only the power derating factors on the specified objects are reset. Once a `reset_power_derate` command is applied to an object, the power derating factors of all power components, including switching, internal and leakage power are reset. If *object_list* contains the current design, then only global power derating factors are reset and instance specific power derating factors remain untouched.

If the *object_list* contains any hierarchical cells then all cells within that hierarchical cell and also all cells of lower hierarchies will have their inherited power derating factors updated accordingly. Note that power derating factors set specifically on an object will not be overwritten by a set/reset command on its parent hierarchical cell. To reset the power derating factors set specifically on the child cells, include them in the object list of the command.

The **-groups** options allows to reset the derate factors of cells only in the specified groups.

Heterogeneous collections of objects may be combined and passed to this command.

If *object_list* contains any hierarchical cells then all cells within that

hierarchical cell and also all cells of all lower hierarchies will have their inherited derate factors updated accordingly. Note that derate factors set specifically on an object will not be overwritten by a set/reset command on its parent hierarchical cell.

EXAMPLES

The following command resets both power derating factors set globally on the design and those set on specific instances in the design.

```
pt_shell> reset_power_derate
```

The following command resets the power derating factors set globally on the design MY_DESIGN only.

```
pt_shell> reset_power_derate [get_design MY_DESIGN]
```

The following example resets the power derating factors set on cell U1.

```
pt_shell> reset_power_derate [get_cell U1]
```

The following command resets the power derating factors set on all cells matching "*".

```
pt_shell> reset_power_derate [add_to_collection [get_cells *]]
```

The following example resets the power derating factors set on all instances of library cell IV in the library MY_LIB.

```
pt_shell> reset_power_derate [get_lib_cells MY_LIB/IV]
```

Assuming that the hierarchical cell H1 contains a cell U1, then the following command resets the derate factors set on U1 and as a result U1 inherits the derate factors that are set on H1.

```
pt_shell> reset_power_derate [get_cells H1/U1]
```

Assuming that the hierarchical cell H1 also contains a hierarchical cell H2, then the following command resets the power derate factors set on H2 and as a result H2 and all of its cells inherit the power derating factors that are set on H1.

```
pt_shell> reset_power_derate [get_cells H1/H2]
```

SEE ALSO

set_power_derate (2), **report_power** (2), **report_power_derate** (2).

reset_scale_parasitics

Resets the scaling that was done previously using *scale_parasitics* command.

SYNTAX

```
int reset_scale_parasitics [net_list] list net_list
```

ARGUMENTS

net_list

Limits the reset of scaling to this list of nets. If this option is used, it is assumed that net-specific scaling has been done for those nets. If not, an error gets generated.

DESCRIPTION

Use the *reset_scale_parasitics* command to reset the scaling of parasitics that was previously done through the *scale_parasitics* command.

EXAMPLES

This example resets scaling of parasitics previously done.

```
pt_shell> reset_scale_parasitics
```

SEE ALSO

```
read_parasitics (2). scale_parasitics (2). report_scale_parasitics (2).
```

reset_switching_activity

Resets switching activity annotation on nets, pins, ports, and cells of the current design.

SYNTAX

```
int reset_switching_activity
    [-static_probability]
    [-toggle_rate]
    [-state_condition state]
    [-path_sources name_list]
    [-no_hierarchy]
    [object_list]
    [-quiet]
```

```
string state
string name_list
list object_list
```

ARGUMENTS

-static_probability

Specifies the static probability value to be reset for the given design object. The static probability value is internally reset to an invalid number.

-toggle_rate

Specifies the toggle rate (rise and fall if applicable) and glitch rate (rise and fall if applicable) to be reset for the given design object. The toggle and glitch rate values are internally reset to invalid numbers.

-state_condition *state*

Specifies the state condition when resetting state-dependent toggle rate and glitch rates on pins or state-dependent static probabilities on cells. State dependent toggle rate and glitch rate can be annotated when the internal power of the library cell pin is characterized with state-dependent power tables. State-dependent static probabilities can be annotated when the cell leakage power is characterized with state-dependent power tables. The state condition specified with this argument must be logically equivalent to a state condition in the internal / leakage power characterization. The state condition should be enclosed between " ". Moreover, if you want to reset switching activity information for the default state condition, the argument of the **-state** option should be "default". Note that **-state_condition** applies only to the leaf level cells.

-path_sources *name_list*

Specifies the path sources when resetting path-dependent toggle rate and glitch rate on pins. This is used when the library cell pin has path-dependent internal power. The path sources specified with this argument must be the same as those in the internal power characterization. When listing more than one pin in path sources the pin names should be separated by a space and enclosed between quotes " ". For example, if the pins in path sources are A

and B, the argument for the `-path` option looks like "A B". The **`-path_sources`** argument applies only to the leaf level cells. Note that if the *name_list* option is given using **`-path_sources`** argument, the *state* option should also be provided using the **`-state_condition`** condition.

`-no_hierarchy`

This option specifies that only the objects in the hierarchy of the list of instances, specified using **`object_list`** option, is reset. If no instance list is specified then the switching activity on the objects in the same hierarchy of the current instance is reset. If the **`-no_hierarchy`** option is not specified and *object_list* is also not given, the switching activity on all the objects in all the hierarchy of the current instance are reset. Note that if leaf level cells are given using the *object_list* option, the command internally assumes this option is true.

`object_list`

Specifies a list of nets, pins, ports, or instances in the current design on which the *static_probability*, *toggle_rate*, and *glitch_rate* switching activity values are to be reset.

`-quiet`

Specifies quiet mode and suppresses warnings on design objects for which switching activity could not be reset.

DESCRIPTION

You can use this command to reset design nets, ports, pins, and cells with the different kinds of switching activity. These include simple toggle rate, glitch rate, and static probability on nets, ports and pins; state and path dependent toggle and glitch rate on cell pins, and state dependent static probabilities on cells. Toggle/glitch rates, and static probabilities are reset using the **`-toggle_rate`** and **`-static_probability`** options, respectively. You can make the toggle rate, glitch rate, and static probability state dependent by specifying the *state* with the **`-state_condition`** option. You can make the toggle rate and glitch rate path dependent by specifying the path sources with the **`-path_sources`** option. The design objects that are reset with switching activity can be specified as a list of objects. You can reset only the objects in the same hierarchy by using **`-no_hierarchy`** option.

Most of the above functionality, and all the options, apply when **`power_analysis_mode`** has been set to **`averaged`** or **`leakage_variation`**. When **`power_analysis_mode`** has been set to **`time_based`**, **`reset_switching_activity`** has a different meaning. In this mode, **`reset_switching_activity`** removes any VCD or other activity indicated previously with **`read_vcd`**. Because the **`time_based`** mode is not based on toggle rates, attempting to reset switching activity on individual objects will have no effect.

EXAMPLES

The following example shows resetting *toggle_rate*, *glitch_rate*, and *static_probability* values only on the current instance of the design TOP_MULT.

```
pt_shell > current_design TOP_MULT
```

```

pt_shell > reset_switching_activity -no_hierarchy

pt_shell > reset_switching_activity -no_hierarchy -toggle_rate -static_probability

pt_shell > reset_switching_activity -toggle_rate -no_hierarchy

pt_shell > reset_switching_activity -static_probability -no_hierarchy

```

The following example shows resetting toggle_rate, glitch_rate and static_probability values on the whole design TOP_MULT.

```

pt_shell > current_design TOP_MULT

pt_shell > reset_switching_activity

pt_shell > reset_switching_activity -toggle_rate -static_probability

pt_shell > reset_switching_activity -toggle_rate

pt_shell > reset_switching_activity -static_probability

```

The following example shows resetting toggle_rate, glitch_rate, and static_probability values on the list of instances.

```

pt_shell > current_design TOP_MULT
pt_shell > reset_switching_activity {U1 U2}
pt_shell > reset_switching_activity -no_hierarchy {U1 U2}
pt_shell > reset_switching_activity -toggle_rate {U1 U2}
pt_shell > reset_switching_activity -static_probability {U1 U2}
pt_shell > reset_switching_activity -no_hierarchy -toggle_rate {U1 U2}
pt_shell > reset_switching_activity -no_hierarchy -static_probability {U1
U2}

```

The following **reset_switching_activity** command resets simple toggle rate value, glitch rate value, and static probability value on all design input ports.

```

pt_shell> reset_switching_activity -toggle_rate -static_probability
[get_inputs]
pt_shell> reset_switching_activity [get_inputs]

```

The following example resets different switching activity values on design output ports.

```

pt_shell> reset_switching_activity -toggle_rate [get_outputs]
pt_shell> reset_switching_activity -static_probability [get_outputs]
pt_shell> reset_switching_activity -static_probability -toggle_rate
[get_outputs]

```

The following example resets state dependent static probabilities on the cell or1:

```

pt_shell> reset_switching_activity -static_probability -state_condition "A & B"
[get_cell or1]

```

```

pt_shell> reset_switching_activity -state_condition "A & B" [get_cell or1]

pt_shell> reset_switching_activity -static_probability -state_condition "A & ! B"
[get_cell or1]

pt_shell> reset_switching_activity -static_probability -state_condition "! A & B"
[get_cell or1]

pt_shell> reset_switching_activity -static_probability -state_condition "! A & ! B"
[get_cell or1]

pt_shell> reset_switching_activity -static_probability -state_condition "default"
[get_cell or1]

```

The following example resets simple and path_sources dependent toggle rates and glitch rates on the output pin Y of the cell xor1:

```

pt_shell> reset_switching_activity -toggle_rate [get_pin xor1/Y]
pt_shell> reset_switching_activity -toggle_rate -path_sources "A" -
state_condition "B" [get_pin xor1/Y]
pt_shell> reset_switching_activity -path_sources "A" -state_condition "B"
[get_pin xor1/Y]
pt_shell> reset_switching_activity -toggle_rate -path_sources "B" -
state_condition "A" [get_pin xor1/Y]
pt_shell> reset_switching_activity -toggle_rate -path_sources "A B" -
state_condition "default" [get_pin xor1/Y]

```

SEE ALSO

```

report_switching_activity (2), read_saif(2), get_switching_activity(2),
set_switching_activity(2), report_power (2);

```

reset_timing_derate

Resets user specified derate factors set either on a design or on a specified list of instances (cells, nets or library cells).

SYNTAX

```
int reset_timing_derate
[-hierarchical_net_delay]
[-scalar] [-variation] [-aocvm_guardband]
object_list

list   object_list
```

ARGUMENTS

-hierarchical_net_delay
Indicates that net derate factors within specified hierarchical cells should also be reset.

-scalar
Indicates that only derate factors for deterministic delays should be reset.

-variation
Indicates that only derate factors for statistical delays should be reset.

-aocvm_guardband
Indicates that only guardband derate factors should be reset.

object_list
Specifies a list containing the design, cells, library cells or nets which will be reset.

DESCRIPTION

Call this command with no arguments to reset all derate factors set on the design (to the default value, 1.0). This will include both derate factors set globally on the design and those set on specific instances in the design. Call this command with a list of objects to reset a specific set of objects.

The *object_list* option may be used to reset derate factors on specific objects (cells, library cells or nets) in the design. All derate factors are reset on each instance. If *object_list* contains a design, then only global derate factors are reset and instance specific derate factors are not reset.

Heterogeneous collections of objects may be combined and passed to this command.

If neither *-scalar* nor *-variation* options are specified, then both deterministic and variation derates are reset.

If *object_list* contains any hierarchical cells then all cells within that hierarchical cell and also all cells of all lower hierarchies will have their

inherited derate factors updated accordingly. In addition, if the `-hierarchical_net_delay` option is specified, then each net and/or every net in each hierarchical cell in the `object_list` will have their inherited derate factors updated. Note that derate factors set specifically on an object will not be overwritten by a set/reset command on its parent hierarchical cell.

If a portion of a hierarchical net is reset then all portions of that hierarchical net will also be reset and a warning will be issued indicating this fact.

EXAMPLES

The following command resets both derate factors set globally on the design and those set on specific instances in the design.

```
pt_shell> reset_timing_derate
```

The following command resets the derate factors set globally on the design MY_DESIGN only.

```
pt_shell> reset_timing_derate [get_design MY_DESIGN]
```

The following example resets the derate factors set on cell U1.

```
pt_shell> reset_timing_derate [get_cell U1]
```

The following example resets the derate factors set on net w1.

```
pt_shell> reset_timing_derate [get_net w1]
```

The following command resets the derate factors set on all cells matching "*" and on all nets matching "w*".

```
pt_shell> reset_timing_derate [add_to_collection [get_cells *]  
[get_nets w*] ]
```

The following example resets the derate factors set on all instances of library cell IV in the library MY_LIB.

```
pt_shell> reset_timing_derate [get_lib_cells MY_LIB/IV]
```

Assuming that the hierarchical cell H1 contains a cell U1, then the following command resets the derate factors set on U1 and as a result U1 inherits the derate factors that are set on H1.

```
pt_shell> reset_timing_derate [get_cells H1/U1]
```

Assuming that the hierarchical cell H1 also contains a hierarchical cell H2, then the following command resets the derate factors set on H2 and as a result H2 and all of its cells inherit the derate factors that are set on H1.

```
pt_shell> reset_timing_derate [get_cells H1/H2]
```

Assuming that the hierarchical cell H1 also contains a hierarchical cell H2, then the following command resets the derate factors set on H2 and as a result H2 and all

of its cells AND NETS inherit the derate factors that are set on H1.

```
pt_shell> reset_timing_derate -hierarchical_net_delay [get_cells H1/H2]
```

SEE ALSO

```
set_timing_derate (2), report_design (2), report_timing_derate (2).
```

reset_variation

Resets the association of one or more variations with one or more timing objects.

SYNTAX

```
int reset_variation
[-all]
[variation_list]
object_list
string variation_list
collection object_list
```

ARGUMENTS

-all

Reset the association of all variations with the given timing objects.
Exactly one of *-all* and *variation_list* must be specified.

variation_list

The list of variations that are to be unassociated with the timing objects.
Exactly one of *-all* and *variation_list* must be specified.

object_list

Resets the association of the variations with each of these timing objects.
The objects can be either designs, cells, libcells, cell arcs, or libcell arcs.

DESCRIPTION

The association of each variation with each timing object is removed. The variations will no longer affect the timing on the list of timing objects, but the variation still exists.

This is the inverse operation of **set_variation**.

EXAMPLES

The following example creates a variation, sets the variation on the current design, and then resets the variation.

```
pt_shell> create_variation -name Lfab1 -parameter_name len -type normal -
values {0 1}]
_sel22
pt_shell> set_variation [get_variations Lfab1] [current_design]
_sel23
pt_shell> reset_variation [get_variations Lfab1] [current_design]
1
```

SEE ALSO

`create_variation(2)`, `set_variation(2)`, `remove_variation(2)`.

restore_session

Restore a PrimeTime session from a directory saved by the `save_session` command.

SYNTAX

```
int restore_session directory_name
```

ARGUMENTS

directory_name

Specifies the name of a directory to read the session information from.

DESCRIPTION

This command reads a directory that was written by a **save_session** command and restores PrimeTime to the same state as the session where the **save_session** command was issued. No current designs or libraries can be loaded to execute this command. Either issue this command in a clean shell or use the **remove_designs -all** and **remove_library -all** commands to clear the current session before restoring a new one.

The command first tries to grab all licenses that were present at the time of **save_session**. If this fails, then restoring is aborted.

The design data is restored from data in the session directory. The **restore_session** command reads the library files that are needed to restore the session. The restore directory contains an ASCII file name "lib_map". This file contains the path and leaf name for each technology library needed to restore the session. If necessary, you can edit the path names to give **restore_session** updated locations of the needed technology library files. The leaf names of those files cannot be changed. The technology libraries are assumed to be the same as those used when the **save_session** was issued.

EXAMPLES

This example reads a PrimeTime savable image in a directory named `state1`.

```
pt_shell> restore_session state1
```

SEE ALSO

save_session (2).

save_qtm_model

Saves the current Quick Timing Model (QTM) description.

SYNTAX

```
string save_qtm_model [-format format_list] [-output file_name] [-library_cell]
string format_list
string file_name
```

ARGUMENTS

-format *format_list*
Specifies the output format to be used. Valid output formats are lib and db (the default).

-output *file_name*
Specifies the name of the output file overriding the default. By default, QTM writes the db file to *design_name_lib.db*. If this option is used, the db file is *output_file_lib.db*.

-library_cell
Specifies the model should be written as a library cell rather than a wrapper design and core library cell. Without this option a wrapper design is written to the file *output_file.db* and the model in the file *output_file_lib.db* is named *design_name_core*. With this option no wrapper design is written and the model in the file *output_file_lib.db* is named *design_name*.

DESCRIPTION

This command saves the QTM model and indicates to PrimeTime that the model being defined is completed. All QTM commands must be placed between a **create_qtm_model** and a **save_qtm_model** command.

To display information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, see the **create_qtm_model** manual page. For a more detailed description about QTM, see the *PrimeTime User Guide*.

EXAMPLES

The following example saves a wrapper plus core QTM model in db format.

```
pt_shell> save_qtm_model -format {db}
```

The following example saves a library cell QTM model in lib format and uses the string "current" for the base file name.

```
pt_shell> save_qtm_model -format lib -output current -library_cell
```

SEE ALSO

```
create_qtm_model (2), report_qtm_model (2).
```

save_session

Saves data of a PrimeTime session in the named directory so that it can be restored later with **restore_session** .

SYNTAX

```
int save_session  
[-replace] [-include <include_list> ]  
[-only_used_libraries]  
dir_name
```

ARGUMENTS

dir_name

Specifies the name of a directory to save the session in. If the named directory does not exist, `save_session` will try to create it. If it already exists, `save_session` will issue error message and stop, unless the `-replace` option is given.

`-replace`

If specified, it indicates that if any file or directory with the same name as the specified already exists in the file system, it will be deleted and then overwritten with the saved data for the current session. Therefore, please use `-replace` with great caution, because it may remove all the existing data, including all files and sub-directories, already present in the target directory.

`-only_used_libraries`

By default, `save_session` will save references to all the technology libraries loaded at the saving of the session. And, all these libraries will be required to restore the session later. This option, if specified, indicates that `save_session` should determine which libraries are in use and only save references to those libraries. Only the used libraries will be needed to restore the session later.

`-include`

If specified, it indicates that additional data is to be saved for the items listed. The only valid option is currently "noise".

DESCRIPTION

This command creates a repository of data to save the current PT session to. The name of the directory to save the session is a required argument. If the directory does not exist, a new one will be created and the data for the session will be written into the directory. If the directory exists, it is an error unless the `-replace` option is given. The `-replace` option will cause an existing file or directory to be removed and replaced by the data of the current session to be saved.

As of the 2008.06 release, a pre-updated session can be saved. The `save_session` command does not perform an implicit `update_timing` so sessions can be saved prior to the `update_timing` command. Only a linked design can be saved so `save_session` may

cause an implicit link to be performed. If there are incremental timing updates pending, PrimeTime performs an implicit `update_timing`, as needed. If noise is to be included, an implicit `"update_noise"` will be also issued as needed.

Please note there are a few things that are not saved - collections that involve timing objects, tcl procedures and message (warning/information) suppressions. Also, if there are other designs in PT memory other than the linked design, those are not saved. These have to be read into the restored session separately.

EXAMPLES

This example saves a PrimeTime session in a directory named `state1`.

```
pt_shell> save_session state1
```

SEE ALSO

```
restore_session (2).
```

scale_parasitics

Used to scale the parasitics in memory.

SYNTAX

```
int scale_parasitics
[-resistance_factor r_factor]
[-ground_capacitance_factor c_factor]
[-coupling_capacitance_factor cc_factor]
[net_list]
float r_factor
float c_factor
float cc_factor
list net_list
```

ARGUMENTS

-resistance_factor *r_factor*
A positive floating point number that specifies the factor to apply for scaling resistances.

-ground_capacitance_factor *c_factor*
A positive floating point number that specifies the factor to apply for scaling ground capacitances.

-coupling_capacitance_factor *cc_factor*
A positive floating point number that specifies the factor to apply for scaling coupling capacitances.

net_list
Limits the scaling to this list of nets.

DESCRIPTION

This command enables scaling of resistance, grounded capacitance and/or coupling capacitance values. The primary purpose is to allow you to perform a single parasitic extraction run, then to scale parasitics for effects, such as resistance values for temperature dependence.

The command works for detailed parasitics as well as Pi models. The command works irrespective of the source of the parasitics information (whether SPEF, SBPF, DSPF or RSPF). Any subsequent calculations use the scaled parasitics instead of the original parasitics. Also, if the parasitics are written out using the **write_parasitics** command, scaled parasitics are written out.

Also, the parasitics can be scaled either globally or to a set of specific nets. Multiple scale parasitics are with respect to the original parasitics, and are not cumulative.

If conflicting coupling capacitance scale factors are given for two coupled nets, the last command takes precedence. When parasitics are read in again for previously

scaled nets, the global scaling is still honored, but the net-specific scaling is ignored.

EXAMPLES

This example scales the ground capacitances by a factor of 1.3, resistance values by a factor 1.1 and the coupling capacitance values by a factor of 0.95.

```
pt_shell> scale_parasitics -resis 1.1 -ground 1.3 -coupling 0.95
```

This example scales the ground capacitances by a factor of 2.1, and the coupling capacitance values by a factor of 0.95 for one net n1.

```
pt_shell> scale_parasitics -ground 2.1 -coupling 0.95 [get_net n1]
```

SEE ALSO

```
read_parasitics (2). reset_scale_parasitics (2). report_scale_parasitics (2).
```

set_active_clocks

Sets a group of clocks to be active in the current analysis scope.

SYNTAX

Boolean **set_active_clocks**

active_clock_list | [all_clocks]

list *active_clock_list*

ARGUMENTS

active_clock_list

Specifies a list of clocks matching the clock names. The *active_clock_list* option and the **all_clocks** option are mutually exclusive.

all_clocks

Specifies to analyze all clocks simultaneously. The *active_clock_list* option and the **all_clocks** option are mutually exclusive.

DESCRIPTION

Sets a group of clocks to be active in the current analysis scope. Provides capability to perform timing analysis for a particular set of clock domains, not the entire chip. If few of many clocks are set as active, the tool performs timing analysis only for these clock domains.

If this command is not specified, the tool analyzes all clocks. The last **set_active_clocks** command always overrides previous ones. If you want to analyze all clocks simultaneously again, use the **set_active_clocks** command with *active_clock_list* set to a value of * or use it with the **all_clocks** option.

When a clock or a generated clock is created, it is active by default. If a generated clock is active itself, but its master clock is inactive, it is not analyzed. You can use the *is_active* attribute of the clock object to create a collection of active clock objects. Thus, you can freely add or remove individual clocks from this collection.

You can use the **set_active_clocks** and **set_clock_groups** commands together to simultaneously analyze multiple clocks per register without manually specifying false paths. Please see the **set_clock_groups** command man page for examples.

When **set_active_clocks** command applied on crosstalk delay analysis, the victim and aggressors driven by active clocks only will be considered. So the nets driven by the non active aggressor will be considered quiet. However all nets are sensitive to static noise (as victim) even when there is no active clocks driving them. This feature could be used to set test clocks to non active when the design is normal/mission mode and vice versa. It is important that user sets all active clocks in the design as active, as they can be aggressors to other part of the design.

This command is not supported in **write_sdc**, **characterize_context**, and **extract_model**.

set_active_clocks

The **set_active_clocks** command is used to speed up update_timing for local analysis, and some global functionalities might not behave as you expect. For example, check_timing reports endpoints as unconstrained if they are captured by inactive clocks.

EXAMPLES

The following example sets only two of the existing clocks in the design to be active.

```
pt_shell> set_active_clocks {clk1 clk2}
```

The following example resets all clocks in the design to be active.

```
pt_shell> set_active_clocks [all_clocks]
```

The following example adds *clk3* to the current active clocks list.

```
pt_shell> set foo [get_clock * -filter is_active==true]
pt_shell> set newfoo [add_to_collection $foo [get_clocks clk3]]
pt_shell> set_active_clocks $newfoo
```

The following example removes *clk3* from the current active clocks list.

```
pt_shell> set foo [get_clock * -filter is_active==true]
pt_shell> set newfoo [remove_from_collection $foo \
[get_clocks * -filter "is_active==true && full_name == clk3"]]
pt_shell> set_active_clocks $newfoo
```

SEE ALSO

add_to_collection (2), **all_clocks** (2), **create_clock** (2), **create_generated_clock** (2), **get_clocks** (2), **set_clock_groups** (2), **remove_clock_groups** (2), **remove_from_collection** (2), **report_clock** (2).

set_annotated_check

Sets the setup, hold, recovery, removal, or nochange timing check value between two pins.

SYNTAX

```
string set_annotated_check  
-setup | -hold  
| -recovery | -removal  
| -nochange_high | -nochange_low  
[-rise]  
[-fall]  
[-min]  
[-max]  
[-from from_pins]  
[-to to_pins]  
[-clock clock_check]  
[-cond sdf_expression]  
[-worst]  
check_value  
  
list from_pins  
list to_pins  
string clock_check  
string sdf_expression  
float check_value
```

ARGUMENTS

- setup | -hold | -recovery | -removal | -nochange_high | -nochange_low**
Specifies the type of the timing check. You must specify one of these arguments. There must be a corresponding timing arc between the *from_pins* and the *to_pins*. See the DESCRIPTION section for more information about these arguments.
- rise**
Specifies that the timing check is for the data rise transition. If you do not specify either **-rise** or **-fall**, both values are set.
- fall**
Specifies that the timing check is for the data fall transition. If you do not specify either **-rise** or **-fall**, both values are set.
- min**
Use this option only if the design is in min_max mode (min and max operating conditions). Specifies the minimum timing check for both data rise and data fall transitions.
- max**
Use this option only if the design is in min_max mode (min and max operating conditions). Specifies the maximum timing check for both data rise and data fall transitions.

-from *from_pins*
 Specifies a list of leaf cell clock pins that are the startpoints of the timing arcs for which checks are annotated.

-to *to_pins*
 Specifies a list of leaf cell data pins that are the endpoints of the timing arcs for which checks are annotated.

-clock *clock_check*
 Specifies whether the check is for clock rising or falling. By default, checks for both clock rise and fall are set. The allowable values for *clock_check* are **rise** or **fall**.

-cond *sdf_expression*
 Use this option only if the library has a condition attached to the specified timing check arc; otherwise, an error message is generated. Specifies the condition for which the annotated check is valid. The syntax of the condition must match the condition specified in the library using the construct *sdf_cond*. The syntax is the same one used in the Standard Delay Format (SDF).

-worst
 This option is not currently implemented, so it is ignored.

check_value
 Specifies the timing check value between pins on the same cell, in units consistent with the technology library used during optimization. For example, if the technology library specifies delay values in nanoseconds, *check_value* must be expressed in nanoseconds. The *check_value* can be negative.

DESCRIPTION

Annotates a timing check between two or more pins on a cell or a net in the *current design*. This can be appropriate after place and route, for technologies where the timing check value varies for different instances and the library timing check values do not provide sufficient accuracy.

The **-setup** timing check indicates that data must be stable for the amount of time specified by *check_value* before the closing clock edge.

The **-hold** timing check indicates that data must be stable for the amount of time specified by *check_value* after the closing clock edge.

The **-recovery** timing check indicates that an asynchronous set or clear inactive edge cannot occur within *check_value* before the closing clock edge. This is essentially a setup requirement on an asynchronous pin, but only the inactive transition is considered.

The **-removal** timing check indicates that an asynchronous set or clear inactive edge cannot occur until *check_value* time after the closing clock edge. This is essentially a hold requirement on an asynchronous pin, but only the inactive transition is considered.

The **-nochange_high** timing check indicates that the data must not rise within *check_value* time before the clock becomes active, and must not fall until

check_value time after the clock becomes inactive. A rise data transition corresponds to the check before the activating clock edge. A fall data transition corresponds to the check after the deactivating clock edge. A rise clock transition indicates that the clock is active high. A fall clock transition indicates that the clock is active low.

The **-nochange_low** timing check indicates that the data must not fall within *check_value* time before the clock becomes active, and must not rise until *check_value* time after the clock becomes inactive. A fall data transition corresponds to the check before the activating clock edge. A rise data transition corresponds to the check after the deactivating clock edge. A rise clock transition indicates that the clock is active high. A fall clock transition indicates that the clock is active low.

If the design is not linked, it is linked automatically by **set_annotated_check**.

You can use **set_annotated_check** for pins at lower levels of the design hierarchy. You can specify pins in the format of INSTANCE1/INSTANCE2/PIN_NAME.

To list annotated timing check values, use the **report_annotated_check** command. To remove annotated timing check values from a design, use the **remove_annotated_check** or **reset_design** command. To see the effect of **set_annotated_check** for a specific instance, use the **report_timing** command.

EXAMPLES

The following example annotates a setup time of 2.1 units between clock pin *CP* of cell instance *u1/ff12* and data pin *D* of the same cell instance.

```
pt_shell> set_annotated_check -setup 2.1 -from u1/ff12/CP -to u1/ff12/D
```

SEE ALSO

current_design (2), **link** (2), **read_sdf** (2), **remove_annotated_check** (2), **report_annotated_check** (2), **report_timing** (2), **reset_design** (2).

set_annotated_clock_network_power

Annotate power on clock networks.

SYNTAX

```
string set_annotated_clock_network_power  
[-internal_power internal_power]  
[-switching_power switching_power]  
[-leakage_power leakage_power]  
[-total_power total_power]  
[-clock clock_object]
```

```
float internal_power  
float switching_power  
float leakage_power  
float total_power  
string clock_object
```

ARGUMENTS

- internal_power internal_power
Specifies the annotated internal power value on the clock network.
- switching_power switching_power
Specifies the annotated switching power value on the clock network.
- leakage_power leakage_power
Specifies the annotated leakage power value on the clock network.
- total_power total_power
Specifies the annotated total power value on the clock network. Will be treated as internal power in the reports.
- clock clock_object
Specifies the clock in the related clock network on which the power values are annotated.

DESCRIPTION

The **set_annotated_clock_network_power** command provide the capability to annotate the power values on the clock networks in the reports from PrimeTime-PX. If this command is used before **report_power**, PrimeTime-PX will use the annotated clock network power values in the summary report. There will also be a separate annotated clock network power report embedded in the regular summary power report. The *clock_network* group power in the summary report will exclude the clock network objects whose power values are annotated. An attribute 'e' will mark such a situation in the report.

The power values in the unit of Watt are specified by either *-total_power* option of a combination of *-internal_power*, *-switching_power* and *-leakage_power* options, If *-total_power* option is used, the power values are also treated as internal power in the reports to make the results consistent. If the options of *-switching_power*, -

internal_power and/or *-leakage_power* are used, the values are annotated on internal, switching and/or leakage power respectively. The total power will be the sum of these three components. These three options can be used together or separately. But they can not be used together with the *-total_power* option, otherwise error message **CMD-001** will be issued and the command will fail.

By default, the power values will be annotated on the whole clock network in the design. If *-clock* option is used, the power values will be annotated on the collection of clock network objects of the corresponding clock domain. Only one clock is allowed for one command. Multiple **set_annotated_clock_network_power** commands can be used to specify annotated power of multiple clock domains separately. The annotated clock network power reports will list them separately.

If two **set_annotated_clock_network_power** commands are specified for the same clock domain (or the whole clock network) and the power types (internal, switching, leakage or total) are the same, The power values of the later command will override the earlier one. If they are not the same power type, both power values will be kept and reported.

The **set_annotated_clock_network_power** command can work with **report_power -groups group_list** only when the group_list contains the default 'clock_network' power group, otherwise error message **PWR-296** will be issued and the **report_power** command will fail.

The **set_annotated_clock_network_power** command will cause the cells in the related clock network not to be reported by **report_power -cell** command. The other options in the **report_power** command are not affected.

The annotated clock network power has higher priority than the estimated clock network power. If both features are invoked in the **report_power** command, A warning message **PWR-295** will be issued and annotated clock network power values will be used in power reports.

To remove the annotated clock network power values already specified for the design, use the **remove_annotated_clock_network_power** command. **report_power** will revert back to its original behavior.

In the following example, the internal, switching power are annotated separately for clock domain CKA and CKB.

```
pt_shell> create_clock -name CKA -period 10 clka
pt_shell> create_clock -name CKB -period 5 clka
pt_shell> set_annotated_clock_network_power -internal 1.0e-03 -
switching 2.0e-03 -clock CKA
pt_shell> set_annotated_clock_network_power -switching 2.0e-03 -clock CKB
pt_shell> report_power
```

The following is the output from the above commands

```
*****
Report : Statistical Average Power
Design : my_design
```

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs

	Internal	Switching	Leakage	Total		
Clock	Power	Power	Power	Power	(%)	Attrs

All Annotated Clocks	1.000e-03	2.000e-03	0.0000	3.000e-03	(31.23%)

```
set_annotated_clock_network_power
899
```

set_annotated_delay

Sets the net or cell delay value between two pins.

SYNTAX

```
string set_annotated_delay -cell | -net  
[-rise]  
[-fall]  
[-min]  
[-max]  
[-load_delay load_delay_type]  
[-from from_pins]  
[-to to_pins]  
[-of_objects objects]  
[-cond sdf_expression]  
[-increment]  
[-delta_only]  
[-worst]  
-variation variation_object  
delay_value
```

```
string load_delay_type  
list from_pins  
list to_pins  
string sdf_expression  
float delay_value
```

ARGUMENTS

-cell

Specifies that the delay annotated is a cell delay. The **-cell** and **-net** arguments are mutually exclusive; you must specify one, but not both.

-net

Specifies that the delay annotated is a net delay. The **-net** and **-cell** arguments are mutually exclusive; you must specify one, but not both.

-rise

Indicates that the delay is for the data rise transition. If you do not specify either **-rise** or **-fall**, both values are set.

-fall

Indicates that the timing check is for the data fall transition. If you do not specify either **-rise** or **-fall**, both values are set.

-min

Use this option only if the design is in min_max mode (min and max operating conditions). Specifies the minimum delay for both data rise and data fall

set_annotated_delay

transitions.

-max

Use this option only if the design is in min_max mode (min and max operating conditions). Specifies the maximum delay for both data rise and data fall transitions.

-load_delay *load_delay_type*

Specifies whether load delay is to be included as part of annotated net delays or as part of annotated cell delays. Allowed values are **net** or **cell**. Load delay is the portion of cell delay resulting from the capacitive load of the net the cell is driving. All timing arcs of the same net and of the same cell, must be annotated with the same *load_delay_type*.

-from *from_list*

Specifies a list of leaf cell pins and top level ports that are the startpoints of the timing arcs for which delays are annotated. The **-from/to** and **-of_objects** arguments are mutually exclusive; you must specify one, but not both.

-to *to_list*

Specifies a list of leaf cell pins and/or top level ports that are the endpoints of the timing arcs for which delays are annotated. The **-from/to** and **-of_objects** arguments are mutually exclusive; you must specify one, but not both.

-of_objects *objects_list*

Specifies a list of timing arcs (created with the **get_timing_arcs** command) on which to annotate. The **-of_objects** and **-from/to** arguments are mutually exclusive; you must specify one, but not both.

-cond *sdf_expression*

Use this option only if the library has a condition attached to the specified delay timing arc; otherwise, an error message is generated. Specifies the condition for which the annotated delay is valid. The syntax of the condition must match the condition specified in the library using the construct *sdf_cond*. The syntax is the same one used in the Standard Delay Format (SDF).

-increment

Specifies that the delay is to be incremented to the current delay of the specified timing arc. It should be noted that if this command is used before a timing update, the current delay could be 0.

-delta_only

Specifies that the annotated delay is to be added to the net delay value calculated by PrimeTime. You cannot use this option with **-cell**.

-worst

This option is not yet implemented, so it is ignored.

delay_value

Specifies the delay value between pins on the same cell, in units consistent with the technology library used during optimization. For example, if the technology library specifies delay values in nanoseconds, *delay_value* must be expressed in nanoseconds.

-variation *variation_object*

Specify a variation to annotate on all arcs between the from and to pins. The *variation_object* must be created using the `create_variation` command.

DESCRIPTION

Annotates the cell delay between two or more pins on a cell, or the net delay between two or more pins on the same net, in the *current design*. With the **-net** option, pins in the *from_list* must be cell output or inout pins, and pins in the *to_list* must be cell input or inout pins. With the **-cell** option, both the **-from** and **-to** options are required. Pins in the *from_list* must be cell input or inout pins, and pins in the *to_list* must be cell output or inout pins. To verify that the back-annotation done by **set_annotated_delay** is correct, run the **update_timing** command.

Load delay, also known as extra source gate delay, is the portion of cell delay caused by the capacitive load of the driven net. Some delay calculators consider load delay part of the net delay and others consider it part of the cell delay. If your annotated delay value (for either cell or net) assumes that load delay is part of the cell delay, use **-load_delay cell**. If your delay value assumes that load delay is included in the net delay, use **-load_delay net**. By default, load delays are assumed to be in cell delays, so they appear in **report_timing** path listings.

The specified delay value overrides the internally-estimated cell and net delay value. If the specified pins are not in the same cell or on the same net, when the timing is updated an error message is generated and *delay_value* is discarded for those pins.

If a timing arc has annotated delta delay (**set_annotated_delay -net -delta_only**) and then total arc delay is annotated (**set_annotated_delay -net**), the arc delay is assumed to include the delta. In other words, delta delay is not used if delay is annotated. The annotated delta delay becomes significant once the annotated delay is removed.

The **set_annotated_delay** command can be used for pins at lower levels of the design hierarchy. Pins are specified in the format of `INSTANCE1/INSTANCE2/PIN_NAME`.

To list annotated delay values, use the **report_annotated_delay** command. To remove the annotated cell or net delay values from a design, use the **remove_annotated_delay** or the **reset_design** command.

EXAMPLES

The following example annotates a cell delay of 20 units between input pin A of cell instance U1/U2/U3 and output pin Z of the same cell instance. The delay value of 20 includes the load delay.

```
pt_shell> set_annotated_delay -cell -load_delay cell 20 -from U1/U2/U3/A -to U1/U2/  
U3/Z
```

The following example annotates a rise net delay of 1.4 units between output pin U1/Z and input pin U2/A. The delay value for this net does not include load delay.

```
pt_shell> set_annotated_delay -net -rise 1.4 -load_delay cell -from U1/Z -to U2/A
```

The following example annotates a rise net delay of 12.3 units between the same output pins. In this example, the net delay value includes load delay.

```
pt_shell> set_annotated_delay -net -rise 12.3 -load_delay net -from U1/Z -to U2/A
```

SEE ALSO

`current_design (2)`, `link (2)`, `read_sdf (2)`, `remove_annotated_delay (2)`,
`report_annotated_delay (2)`, `report_timing (2)`, `reset_design (2)`.

set_annotated_power

Annotate power on unresolved black-box cells or leaf cells.

SYNTAX

```
int set_annotated_power  
-internal_power internal_power  
-leakage_power leakage_power  
cell_list
```

```
float internal_power  
float leakage_power  
list cell_list
```

ARGUMENTS

```
-internal_power internal_power  
    Specifies the internal power to be annotated.
```

```
-leakage leakage_power  
    Specifies the leakage power to be annotated.
```

```
cell_list  
    Specifies a list of cells on which power is annotated.
```

DESCRIPTION

The `set_annotated_power` command annotates the internal and leakage power in the unit of Watt on the given cells. The command should be used mostly on unresolved black-box cells, where internal and leakage power cannot be estimated. But the command also works for any leaf cells, where the annotated internal and leakage power overrides the internally estimated internal and leakage power. The internally estimated power, if available, is overridden by the annotated power number, but will resurrect if the annotated power is removed.

Switching power (power of the nets connected to the output pins of the cell) cannot be annotated; instead, one can set switching activities on the nets, thus the switching power can be calculated.

The annotated power is average power, so it has no effect on power waveform.

EXAMPLES

The following example shows how power is annotated, removed and reported.

```
pt_shell> set_annotated_power -int 1 -leak 0.01 u0/*  
1  
pt_shell> report_annotated_power -list_annotated  
*****  
Report : annotated_power
```



```

        -list_annotated
*****

```

Annotated cell powers:

```

-----
1.  u0/u0  (internal: 1  leakage: 0.01)
2.  u0/u1  (internal: 1  leakage: 0.01)

```

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	1	2
	5	2	3

```

1
pt_shell> remove_annotated_power u0/u0
1
pt_shell> report_annotated_power
*****
Report : annotated_power
*****

```

Cell type	Total	Annotated	NOT Annotated
unresolved black-box cell	2	1	1
leaf cell	3	0	3
	5	1	4

```

1

```

SEE ALSO

`remove_annotated_power (2)`, `report_annotated_power (2)`.

set_annotated_transition

Sets the transition time to be annotated on specified pins in the current design.

SYNTAX

```
int set_annotated_transition [-rise][-fall][-min][-max] [-delta_only] slew_value  
pin_list  
float slew_value  
list pin_list
```

ARGUMENTS

-rise

Indicates that *slew_value* represents the data rise transition time.

-fall

Indicates that *slew_value* represents the data fall transition time.

-min

Indicates that *slew_value* represents the minimum transition time. Use this option only if the design is in min-max mode (min and max operating conditions).

-max

Indicates that *slew_value* represents the maximum transition time. Use this option only if the design is in min-max mode (min and max operating conditions).

-delta_only

Indicates that *slew_value* represents a delta transition time to be added to the transition time computed by delay calculation.

slew_value

Specifies the transition time of the specified pins or ports, in units consistent with the technology library used during optimization. For example, if the technology library specifies delay values in nanoseconds, *slew_value* must be expressed in nanoseconds. If used with the **-delta_only** option, *slew_value* can be a negative number.

pin_list

Specifies a list of pins or ports to be annotated with the transition time *slew_value*.

DESCRIPTION

Specifies the transition time to be annotated on pins in the current design. The specified transition time value overrides the internally-estimated transition time value.

set_annotated_transition can be used for pins at lower levels of the design hierarchy. Pins are specified as "INSTANCE1/INSTANCE2/PIN_NAME."

set_annotated_transition

To remove annotated transition times, use **remove_annotated_transition**.

EXAMPLES

The following example annotates a rising transition time of 0.5 units and a falling transition time of 0.7 units on input pin **A** of cell "U1/U2/U3".

```
pt_shell> set_annotated_transition -rise 0.5 [get_pins U1/U2/U3/A] pt_shell>  
set_annotated_transition -fall 0.7 [get_pins U1/U2/U3/A]
```

SEE ALSO

remove_annotated_transition (2), **report_timing** (2), **reset_design** (2),
set_annotated_delay (2).

set_aocvm_coefficient

Specifies the advanced on-chip variation (OCV) random coefficients on library cells, library timing arcs, and cells.

SYNTAX

```
int set_aocvm_coefficient  
coefficient  
object_list
```

Data Types

<i>coefficient</i>	float
<i>object_list</i>	list

ARGUMENTS

coefficient
Specifies a random *coefficient*, which should be a floating-point value greater than zero.

object_list
Specifies a list of library cells, library timing arcs, or leaf cells on which the coefficient is set.

DESCRIPTION

Sets random advanced OCV coefficients on library cells, library timing arcs, and leaf cells. Random coefficients are used to calculate cell path depths; the default depth increment for a cell arc in a path is 1, but this can be changed to *coefficient* using this command. advanced OCV coefficients are not required for an advanced OCV analysis; however, their use might increase the accuracy of the analysis.

When applying random coefficients the following priority is used for cell arcs (in decreasing order of precedence):

- 1) *Cell*
- 2) *Library timing arc*
- 3) *Library cell*
- 4) *1.0* (default)

To display advanced OCV coefficients, use the **report_aocvm -coefficient** command. To remove advanced OCV coefficients from a cell, use the **remove_aocvm -coefficient** or the **reset_design** command.

EXAMPLES

The following examples demonstrate how to specify random advanced OCV coefficients on library cells and library timing arcs.

```
set_aocvm_coefficient  
908
```

```
pt_shell> set_aocvm_coefficient 1.2 [get_lib_cells lib/AN2]  
pt_shell> set_aocvm_coefficient 2.5 [get_lib_timing_arcs -from lib/AN2/A -to lib/  
AN2/Z]
```

SEE ALSO

```
read_aocvm(2)  
remove_aocvm(2)  
report_aocvm(2)  
timing_aocvm_analysis_mode(3)
```

set_case_analysis

Specifies that a port or pin is at a constant logic value 1 or 0, or is considered with a rising or falling transition.

SYNTAX

```
string set_case_analysis value port_or_pin_list  
string0 | 1 | rising | falling  
list port_or_pin_list
```

ARGUMENTS

value

Specifies a constant logic value or a transition to assign to the given pin or port. The valid constant values are **0**, **1**, **zero**, and **one**. The valid transition values are **rising**, **falling**, **rise**, and **fall**.

port_or_pin_list

Lists ports or pins to which the case analysis is assigned. In the case of pins, constant propagation is executed forward only. No backward constant propagation is performed.

DESCRIPTION

Case analysis is a way of specifying a given mode for the design without altering the netlist structure. You can specify for the current timing analysis session, that some signals are at a constant value (**1** or **0**), or that only one type of transition (**rising** or **falling**) is considered.

Pins of a design may be driven by logic values from the design, but if case analysis is used to set a value on such pins, the values set by case analysis will dominate. If the case values are subsequently removed, the value driving the pin from the design will be propagated.

When case analysis is specified as a constant value, this value is propagated through the network as long as the constant value is a controlling value for the traversed logic. For example, if you specify that one of the inputs of a NAND gate is a constant value **0**, it is propagated to the NAND output, which is now considered at a logic constant **1**. This propagated constant value is then propagated to all cells driven by this signal.

Note: When a logic value is propagated onto a net, the associated DRC constraint checks will be disabled. However max_capacitance DRC check can be performed on driver pins for constant nets by setting variable **timing_enable_max_capacitance_set_case_analysis** to **TRUE**. This variable is set to **FALSE** by default.

In the event that the case analysis value is a transition, the given pin or port is considered only for timing analysis with the specified transition. The other transition is disabled. The case analysis information is used by all analysis commands, including the false-path detection algorithm used by the **report_timing**

set_case_analysis

command when issued in conjunction with the **-true** option, and the **report_timing** command when issued in conjunction with the **-justify** option.

Case analysis is used in addition to the mode commands to fully specify the mode of a design. For example, a design that instantiates models with a TESTMODE, is specified so that the TESTMODE is disabled during the timing analysis session by using the **set_mode** command. In addition, if a TESTMODE signal exists on the design, it is specified to a constant logic value, so that all test logic controlled by the TESTMODE signal is disabled.

Set_case_analyis can be used for power analysis for both averaged mode and time-based mode in PrimeTime PX. Values set by set_case_analysis will also be used for the internal logic simulation to get switching activities for un-annotated nets. However, -rising and -falling options are ignored during power calculation. When simulation results from VCD or SAIF conflicts with set_case_analysis values, simulation wins.

EXAMPLES

The following example shows that the port *IN1* is at a constant logic value of **0**.

```
pt_shell> set_case_analysis 0 IN1
```

The following example shows that the pins *U1/U2/A* and *U1/U3/CI* are considered only for a rising transition. The falling transition on these pins is disabled.

```
pt_shell> set_case_analysis rising {U1/U2/A U1/U3/CI}
```

The following example specifies how to disable the TESTMODE of a design for which instances are models having a TESTMODE mode. The design also has a *TEST_PORT* port set to a constant logic value **0**.

```
pt_shell> remove_mode TESTMODE U1/U2
pt_shell> set_case_analysis 0 TEST_PORT
```

SEE ALSO

remove_case_analysis (2), **report_case_analysis** (2), **disable_case_analysis** (3), **disable_case_analysis_ti_hi_lo** (3), **report_timing** (2), **set_mode** (2).

set_clock_gating_check

Specifies the value of setup and hold time for clock gating checks.

SYNTAX

```
string set_clock_gating_check  
[-setup setup_value]  
[-hold hold_value]  
[-rise | -fall]  
[-high | -low]  
[object_list]
```

```
float setup_value  
float hold_value  
list object_list
```

ARGUMENTS

-setup *setup_value*
Specifies the clock gating setup time. The default is 0.0.

-hold *hold_value*
Specifies the clock gating hold time. The default is 0.0.

-rise
Indicates that only rising delays are to be constrained. By default, if neither **-rise** nor **-fall** are specified, both rising and falling delays are constrained.

-fall
Indicates that only falling delays are to be constrained. By default, if neither **-rise** nor **-fall** are specified, both rising and falling delays are constrained.

-high
Indicate that the check is to be performed on the high level of the clock. By default, PrimeTime determines whether to use the high or low level of the clock using information from the cell's logic. That is, for AND and NAND gates PrimeTime performs the check on the high level; for OR and NOR gates, on the low level. For some complex cells (for example, MUX, OR-AND) PrimeTime cannot determine which to use, and does not perform checks unless you specify either **-high** or **-low**. If the user-specified value differs from that derived by PrimeTime, the user-specified value takes precedence, and a warning message is issued. Unlike setup or hold time, this option sets the attribute only on the specified pin or cell and does not affect the transitive fanout pin or cell. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port.

-low
Indicates that the check is to be performed on the low level of the clock. By default, PrimeTime determines whether to use the high or low clock level using information from the cell's logic. That is, for AND and NAND gates

PrimeTime performs the check on the high level; for OR and NOR gates, on the low edge. For some complex cells (for example, MUX, OR-AND) PrimeTime cannot determine which to use, and does not perform checks unless you specify either **-high** or **-low**. If the user-specified value differs from that derived by PrimeTime, the user-specified value takes precedence, and a warning message is issued. Unlike setup or hold time, this option sets the attribute only on the specified pin or cell and does not affect the transitive fanout pin or cell. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port.

object_list

Specifies a list of objects in the current design for which the clock gating check is to be applied. The objects can be clocks, ports, pins, or cells. If a cell is specified, all input pins of that cell are affected. If a pin, cell, or port is specified, all gates in the transitive fanout are affected. If a clock is specified, the clock gating check is applied to all gating gates driven by that clock. If you specify **-high** or **-low** you must also specify *object_list*; in that case, *object_list* must not contain a clock or a port. By default, if *object_list* is not specified, the clock gating check is applied to the current design.

DESCRIPTION

The **set_clock_gating_check** command specifies a setup or hold time clock gating check to be used for clocks, ports, pins, or cells. The gating check is performed on pins that gate a clock signal.

The clock gating setup check is used to ensure the controlling data signals are stable before the clock is active. This check is performed on combinational gates through which the clock signals are propagated. The arrival time of the leading edge of the clock pin is checked against both levels of any data signals gating the clock. A clock gating setup failure can cause either a glitch at the leading edge of the clock pulse, or a clipped clock pulse.

The clock gating hold check is used to ensure that the controlling data signals are stable while the clock is active. The arrival time of the trailing edge of the clock pin is checked against both levels of any data signal gating the clipped clock pulse.

The options **-high** and **-low** are intended to specify clock gating checks that PrimeTime cannot determine. These options apply only to the pins specified and not to the cells or pins in the transitive fanout. Conversely, setup and hold values for clock gating checks apply to the cells and pins in the transitive fanout. If you need to use the **-setup** or **-hold** options and also the **-high** or **-low** options, issue two separate **set_clock_gating_check** commands to avoid confusion as to what is propagated and what is not propagated.

Use the **remove_clock_gating_check** command to remove information set by **set_clock_gating_check**. The **reset_design** command removes all attributes from the design, including those set by **set_clock_gating_check**.

Use the **report_clock_gating_check** command to report the information for the clock gating check.

The order in which clock gating check constraints are applied is the following - clock gating check set on specific pins or cells, clock gating constraint arcs from library, clock gating check constraints applied on gates which are in the transitive fanout of pins, cells or ports on which gating check is set, clock gating constraint set on clocks, and design.

EXAMPLES

The following example specifies a setup time of 0.2 and a hold time of 0.4 for all gates in the clock network of clock CK1.

```
pt_shell> set_clock_gating_check -setup 0.2 -hold 0.4 [get_clocks CK1]
```

The following example specifies a setup time of 0.5 on the gate and1.

```
pt_shell> set_clock_gating_check -setup 0.5 [get_cells and1]
```

SEE ALSO

```
remove_clock_gating_check (2), current_design (2), report_constraint (2),  
reset_design (2). report_clock_gating_check (2).
```

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during the timing analysis.

SYNTAX

Boolean **set_clock_groups**

```
[-physically_exclusive | -logically_exclusive | -asynchronous]  
[-allow_paths]  
[-name name]  
[-group clock_list]*
```

list *clock_list*

ARGUMENTS

Note: Options marked with asterisks (*) in the SYNTAX can be used more than once in the same command.

-physically_exclusive

Specifies that the clock groups are physically exclusive with each other. Physically exclusive clocks cannot co-exist in the design physically. An example of this is multiple clocks that are defined on the same source pin. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-logically_exclusive

Logically exclusive clocks do not have any functional paths between them, but may have coupling interactions with each other. An example of logically-exclusive clocks is multiple clocks, which are selected by a MUX but can still interact through coupling upstream of the MUX cell. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-asynchronous

Specifies that the clock groups are asynchronous to each other. Two clocks are asynchronous with respect to each other if they have no phase relationship at all. Signal integrity analysis uses an infinite arrival window on the aggressor unless all the arrival windows on the victim net and the aggressor net are defined by synchronous clocks. The **-physically_exclusive**, **-logically_exclusive** and **-asynchronous** options are mutually exclusive; you must choose only one.

-allow_paths

Enable the timing analysis between specified asynchronous clock groups. The default behavior is to suppress timing paths between asynchronous clocks. The **-allow_paths** option must be used with **-asynchronous** option.

-name *name*

Specifies a name for the clock grouping to be created. Each command should specify a unique name, which identifies the exclusive or asynchronous

relationship among specified clock groups, and this name is used later on to easily remove the clock grouping defined here. By default, the command creates a unique name.

-group *clock_list*

Specifies a list of clocks. You can use the **-group** option more than once in a single command execution. Each **-group** iteration specifies a group of clocks, which are exclusive or asynchronous with the clocks in all other groups. If only one group is specified, it means that the clocks in that group are exclusive or asynchronous with all other clocks in the design. Thus, a default other group is created for this single group. Whenever a new clock is created, it is automatically included in the default exclusive or asynchronous group. Substitute the list you want for *clock_list*.

DESCRIPTION

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during the timing analysis. This is similar to using the `set_false_path` command. In addition, these relationships also imply the type of crosstalk analysis which should be performed between the clocks. A clock cannot be present in multiple groups during one **set_clock_groups** command execution, but can be included in multiple groups by executing multiple **set_clock_groups** commands. Clock relationships are only implied across the specified clock groups; no relationship is implied across clocks within a group.

For all three relationships, timing paths between the clocks are suppressed. The relationships differ in how crosstalk is handled. If clocks are asynchronous, the clocks are assumed to have an infinite window relationship with each other. If clocks are physically exclusive, only one clock can act as an aggressor or victim during crosstalk analysis; interactions between the clocks will not be explored. If clocks are logically exclusive, the crosstalk computation will be computed normally (synchronously if the clocks are synchronous) even though the timing paths between the clocks are not reported.

Multiple relationship types can exist between two clocks. When this happens, the relationships obey the following precedence:

- * *physically exclusive (highest)*
- * *asynchronous*
- * *logically exclusive (lowest)*

These precedence rules reflect the real physical behavior of the resulting circuit. For example, if two clocks are asynchronous but they are never simultaneously present, then no crosstalk should be computed.

Note that the traditional `set_false_path` exception between clocks will not result in infinite window crosstalk computation between two clocks. If multiple clocks are asynchronous with each other, the asynchronous relationship should be specified with `set_clock_groups`. Failure to specify this relationship for asynchronous clocks could result in an optimistic analysis!

A special `-allow_paths` modifier can be used with the `-asynchronous` option. When used, the clocks are assumed to have an infinite window relationship for crosstalk

purposes, but timing paths between the clocks will be treated normally. When using this option, paths between the clock domains can be manually disabled using the `set_false_path` command.

When a clock pair is defined such that it has false paths (using either physically or logically exclusive or asynchronous) but subsequently modified to have -asynchronous -allow_paths, then this is an error and the first definition holds. In the reverse situation, also the redefinition to false paths from -allow_paths between the clock pairs would also be an error. Please refer message UITE-457 and UITE-458.

This command should be used to specify the relationships among clocks before using the **set_active_clocks** command. You should not also set a false path if you have already specified two clocks as exclusive or asynchronous. An error message is issued if you attempt to set a false path between two clocks which are already exclusive or asynchronous. If a false path was previously set between two clocks when an exclusive or asynchronous relationship is applied, the false path is overwritten by the **set_clock_groups** command. Other exceptions are unaffected.

A clock relationship on a master clock will not automatically propagate to any generated clocks. If the relationship should also apply to generated clocks, they should be explicitly included in the `set_clock_groups` command.

To undo the **set_clock_groups** command, use the **remove_clock_groups** command. To report the clock groups defined in a design, use the **report_clock** command with the **-groups** option.

EXAMPLES

The following example defines two asynchronous clock domains.

```
pt_shell> set_clock_groups -asynchronous -name g1 -group CLK33 -group CLK50
```

The following example defines a clock named *TESTCLK* to be asynchronous with all other clocks in the design:

```
pt_shell> set_clock_groups -asynchronous -group TESTCLK
```

The following example shows how to simultaneously analyze multiple clocks per register without manually specifying false paths. Assume two pairs of mutually-exclusive clocks that are multiplexed:

CLK1 and *CLK2*
CLK3 and *CLK4*.

If each pair of clocks is selected by a different signal, you must execute two **set_clock_groups** commands to specify two independent exclusivity relationships, one for each MUX selection signal:

```
pt_shell> set_clock_groups -physically_exclusive -group CLK1 -group CLK2  
pt_shell> set_clock_groups -physically_exclusive -group CLK3 -group CLK4
```

If each pair of clocks is selected by a single MUX selection signal, you would execute only one **set_clock_groups** command to simultaneously analyze all four clocks.

```
pt_shell> set_clock_groups -physically_exclusive -group {CLK1 CLK3} -  
group {CLK2 CLK4}
```

In this case, we are not implying any type of relationship between CLK1-CLK3 or CLK2-CLK4. For example, if CLK1 and CLK3 were also asynchronous with each other, we would need to specify an additional relationship between them.

SEE ALSO

```
remove_clock_groups (2), report_clock (2), set_active_clocks (2), set_false_path  
(2), create_clock (2), create_generated_clock (2).
```

set_clock_latency

Specifies latency of clock network.

SYNTAX

```
string set_clock_latency
[-clock clock_list]
[-rise][-fall]
[-min][-max]
[-source]
[-late][-early]
[-dynamic dynamic_component_of_delay]
[-pll_shift]
delay
object_list

list clock_list
float delay
float dynamic_component_of_delay
list object_list
```

ARGUMENTS

-clock *clock_list*
Specifies a list of clock objects to be associated with the network latency that is placed on all pin/port objects in *object_list*. If the **-clock** option is supplied when *object_list* refers to clock objects, a warning is issued that the option is not relevant in this case and execution of the command proceeds as if -clock was not given. A more specific clock latency setting overrides a more general one.

-rise
Specifies clock rise latency.

-fall
Specifies clock fall latency.

-min
Specifies clock latency for the minimum operating condition.

-max
Specifies clock latency for the maximum operating condition.

-source
Specifies clock source latency.

-late
Specifies clock late source latency.

-early
Specifies clock early source latency.

`-dynamic` *dynamic_component_of_delay*
Specifies dynamic component of clock latency value.

`-pll_shift`
Specifies that latencies correspond to PLL shifts. This option applies only to PLL output clocks.

`delay`
Specifies clock latency value.

`object_list`
Lists clocks, ports, or pins.

DESCRIPTION

Two types of clock latency can be specified for a design: The clock network latency and the clock source latency.

The clock network latency is the time it takes a clock signal to propagate from the clock definition point to a register clock pin. The rise and fall latencies are the latencies for rising and falling transitions at the register clock pin, respectively. Inversion of the clock waveform, if present in the clock network, is not taken into consideration when computing clock network latencies at register clock pins. Note that this behavior is different than it was in previous releases.

Clock source latency is the time it takes for a clock signal to propagate from its actual ideal waveform origin point to the clock definition point in the design. It can be used to model off-chip clock latency when the clock generation circuit is not part of the current design. You can use clock source latency for generated clocks to model the delay from master-clock to generated-clock definition point.

Dynamic clock latency is the component of the total clock latency which is considered 'dynamic' in nature. A 'dynamic' value may differ along successive clock cycles and hence may only be used to calculate CRP if a zero-cycle path is being considered. A zero-cycle path is one in which the same clock edge both launches and captures.

The dynamic component of any clock latency may be specified using the **-dynamic** switch. It may only be specified if the **-source** switch and total clock latency is also specified with the command.

The phase error for output clocks coming out of phase locked loops (PLLs) can be specified using the **-pll_shift** switch. If **-pll_shift** option is used, the specified delay gets added to the phase adjustment of the PLL.

PrimeTime assumes ideal clocking, which means clocks have a specified network latency (designated by the **set_clock_latency** command) or zero network latency, by default. Propagated clock network latency (designated by the **set_propagated_clock** command) is normally used for post-layout after final clock tree generation. Ideal clock network latency provides an estimate of the clock tree for pre-layout.

You can specify clock source latency for ideal or propagated clocks. The total clock latency at a register clock pin is the sum of clock source latency and clock network latency. If the rise and fall latencies are different, then the source latencies for

a latch are picked based on the sense at the clock port and network latencies are picked based on the sense at the latch clock pin. Thus, source latency takes into account the clock inversions on the clock network, but network latency does not.

In `bc_wc` analysis mode, the `-min` and `-max` options specify the corner for the clock latency value (fast or slow), and the `-early` and `-late` options specify the early or late latency value within the corner. In the `on_chip_analysis` mode, the min/max operating conditions are the variation bounds within the single corner analysis. Therefore, only the min-early and max-late latencies are used for the analysis. In the `on_chip_variation` mode, it is sufficient to specify either `-early/-late` options or `-min/-max` options for `set_clock_latency`.

For internally-generated clocks, PrimeTime automatically computes the clock source latency provided that the master clock of the generated clock has propagated latency and no user-specified value for the generated clock source latency exists. If the master clock is ideal, the master clock has source latency, and there is no user-specified value for the generated clock's source latency, then zero source latency is assumed.

If the **`set_clock_latency`** command is applied to pins or ports, it affects all register clock pins in the transitive fanout of the pins or ports. Directly setting a network latency makes the affected registers ideal. A warning is issued if the **`set_propagated_clock`** command has previously set a propagated attribute on the same object (clock, pin, or port). Clock source latencies are allowed only on clocks and clock source pins.

No check will be performed at the UI level that a clock specified with **`-clock`** passes through the pin or pins referenced in `object_list` or not. If the clock specified does not pass through the pin, the command will have no effect but no warning of this fact is given.

To undo **`set_clock_latency`**, use the **`remove_clock_latency`** command.

To see clock network and source latency information (including dynamic component), use the **`report_clock`** command with the **`-skew`** option.

EXAMPLES

The following example specifies a rise latency of **1.2** and a fall latency of **0.9** for a clock named **CLK1**.

```
pt_shell> set_clock_latency 1.2 \  
-rise [get_clocks CLK1]  
pt_shell> set_clock_latency 0.9 \  
-fall [get_clocks CLK1]
```

The following example specifies an early rise and a fall source latency of **0.8** and a late rise and fall source latency of **0.9** for a clock named **CLK1**.

```
pt_shell> set_clock_latency 0.8 -source \  
-early [get_clocks CLK1]  
pt_shell> set_clock_latency 0.9 -source \  
-late [get_clocks CLK1]
```

The following example specifies an early and late source latency of 3 and 5 respectively with dynamic components of 0.5.

```
pt_shell> set_clock_latency 3 -source \  
-early -dynamic 0.5 [get_clocks CLK1]  
pt_shell> set_clock_latency 5 -source \  
-late -dynamic 0.5 [get_clocks CLK1]
```

SEE ALSO

remove_clock_latency (2), report_clock (2), set_clock_latency (2),
set_clock_transition (2), set_clock_uncertainty (2), set_propagated_clock (2).

set_clock_sense

Specifies unateness propagating forward for pins with respect to clock source.

SYNTAX

```
string set_clock_sense  
[-positive]  
[-negative]  
[-stop_propagation]  
[-logical_stop_propagation]  
[-pulse pulse_type ]  
[-clocks clock_list]  
object_list
```

Data Types

```
clock_listlist  
object_listlist
```

ARGUMENTS

- clocks *clock_list***
Specifies a list of clock objects to be applied with the given unateness that is placed on all pin objects in *object_list*. If the **-clocks** option is not supplied, all clocks passing through the given pin objects will be considered.
- stop_propagation**
Stops the propagation of specified clocks in the *clock_list* from the specified pins or cell timing arcs in *object_list*. Both clock as clock and clock as data propagation is stopped. **-stop_propagation** can not be specified with **-positive**, **-negative** or **-pulse**.
- logical_stop_propagation**
Stops only the propagation of specified clocks as clock in the *clock_list* from the specified pins or cell timing arcs in *object_list*. But clock as data is allowed to propagate. **-logical_stop_propagation** can not be specified with **-positive**, **-negative** or **-pulse**.
- positive**
Specifies positive unateness applied to all pins in *object_list* with respect to clock source. **-positive** can not be specified with **-negative** or **-pulse**.
- negative**
Specifies negative unateness applied to all pins in *object_list* with respect to clock source. **-positive** can not be specified with **-negative** or **-pulse**.
- pulse *pulse_type***
Specifies the type of pulse clock applied to all pins in *object_list* with respect to clock source. The possible values for *pulse_type* are:
'rise_triggered_high_pulse', 'fall_triggered_high_pulse',

'rise_triggered_low_pulse', 'fall_triggered_low_pulse' **-pulse** can not be spefied with **-negative** or **-pulse**.

object_list

Lists of pins or cell timing arcs with specified unateness to propagate. The timing arcs object can be used with the **-stop_propagation** and **-logical_stop_propagation** option only.

DESCRIPTION

This command gives user control to restrict unateness at pin (to positive or negative unate) with respect to clock source. However, the specified unateness only applies within the non-unate clock network. In this case, User defined sense propagates forward from the given pins.

If the **-clocks** option is supplied, only the specified clock domains are applied. Otherwise, all clocks passing through the given pin objects will be considered.

PrimeTime will issue warning messages if the specified sense on given pins can not be respected in case there is predefined unateness for given pins. Hierachical pin is not supported.

To undo **set_clock_sense**, use the **remove_clock_sense** command.

EXAMPLES

The following example specifies a positive unateness for a pin named **XOR/Z** with respect to clock **CLK1**.

```
pt_shell> set_clock_sense -positive -clocks [get_clocks CLK1] XOR/Z
```

The following exmaple specifies negative unateness for a pin named **MUX/Z** for all clocks.

```
pt_shell> set_clock_sense -negative MUX/Z
```

SEE ALSO

remove_clock_sense (2).

set_clock_transition

Specifies transition time of register clock pins.

SYNTAX

```
string set_clock_transition [-rise] [-fall] [-min] [-max] transition clock_list  
float transition  
list clock_list
```

ARGUMENTS

-rise

Specifies clock transition time for rising clock edge.

-fall

Specifies clock transition time for falling clock edge.

-min

Specifies clock transition time for minimum conditions.

-max

Specifies clock transition time for maximum conditions.

transition

Specifies transition time of clock pins.

clock_list

Provides a list of clocks in the design. The transition times on all register clock pins in the transitive fanout of specified clock are affected. You only can specify clocks with ideal latency in the list. When register clock pins in the fanout of a clock are marked with propagated latency, **set_clock_transition** values are ignored.

DESCRIPTION

This command provides the ability to override the calculated transition times on clock pins of registers.

This command is especially useful for pre-layout when clock trees are incomplete and calculated transition times at register clock pins can be highly pessimistic. The transition value specified with this command overrides the transition times of all nets directly feeding a sequential element clocked by the specified clock.

Use the **set_clock_transition** command only with ideal clocks. For propagated clocks the calculated transition times are used. Propagated clocks are only set after the final clock tree is constructed.

If a clock transition is not specified for an ideal clock, and if the variable `timing_ideal_clock_zero_default_transition` is TRUE (by default it is TRUE), then zero transition will be used. Else, the transition time is calculated as it is for other pins in the design.

To undo **set_clock_transition**, use **remove_clock_transition**.

To list all clock transition values which have been set, use **report_clock -skew**.

EXAMPLES

This example specifies a rise transition time of 0.38 and fall transition time of 0.25 for register clock pins clocked by "CLK1".

```
pt_shell> set_clock_transition 0.38 -rise [get_clocks CLK1]
pt_shell> set_clock_transition 0.25 -fall [get_clocks CLK1]
```

SEE ALSO

remove_clock_transition(2), **report_clock(2)**, **set_clock_latency(2)**,
set_clock_uncertainty(2), **set_propagated_clock(2)**.
timing_ideal_clock_zero_default_transition(3)

set_clock_uncertainty

Specifies the uncertainty (skew) of specified clock networks.

SYNTAX

```
string set_clock_uncertainty  
uncertainty  
[object_list |  
  -from from_clock  
    | -rise_from rise_from_clock  
    | -fall_from fall_from_clock  
  -to to_clock  
    | -rise_to rise_to_clock  
    | -fall_to fall_to_clock]  
[-rise]  
[-fall]  
[-setup]  
[-hold]  
  
list from_clock  
list rise_from_clock  
list fall_from_clock  
list rise_to_clock  
list fall_to_clock  
list to_clock  
list object_list  
float uncertainty
```

ARGUMENTS

-from *from_clock* -to *to_clock*

These two options specify the source and destination clocks for interclock uncertainty. You must specify either the pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to**, or *object_list*; you cannot specify both.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to rising edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from *fall_from_clock*

Same as the **-from** option, but indicates that *uncertainty* applies only to falling edge of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_to *rise_to_clock*

Same as the **-to** option, but indicates that *uncertainty* applies only to rising edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-fall_to *fall_to_clock*

Same as the **-to** option, but indicates that *uncertainty* applies only to falling

edge of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

object_list

Specifies a list of clocks, ports, or pins for simple uncertainty; the uncertainty is applied either to capturing latches clocked by one of the clocks in *object list*, or capturing latches whose clock pins are in the fanout of a port or pin specified in *object_list*. You must specify either the pair of **-from** and **-to**, or *object_list*; you cannot specify both.

-rise

Indicates that *uncertainty* applies to only the rising edge of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-rise_to** instead.

-fall

Indicates that *uncertainty* applies to only the falling edge of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty, and is now obsolete. Unless you need this option for backward-compatibility, use **-fall_to** instead.

-setup

Indicates that *uncertainty* applies only to setup checks. By default, *uncertainty* applies to both setup and hold checks.

-hold

Indicates that *uncertainty* applies only to hold checks. By default, *uncertainty* applies to both setup and hold checks.

uncertainty

A floating point number that specifies the uncertainty value. Typically, clock uncertainty should be positive. Negative uncertainty values are supported for constraining designs with complex clock relationships. Setting the uncertainty value to a negative number could lead to optimistic timing analysis and should be used with extreme care.

DESCRIPTION

Specifies the clock uncertainty (skew characteristics) of specified clock networks. This command can specify either interclock uncertainty or simple uncertainty. For interclock uncertainty, use the **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to** options to specify the source clock and the destination clock; all paths between these receive the uncertainty value. For simple uncertainty, use *object_list*; the uncertainty value is either to capturing latches clocked by one of the clocks in *object list*, or capturing latches whose clock pins are in the fanout of a port or pin specified in *object_list*.

Set the uncertainty to the worst skew expected to the endpoint or between the clock domains. You can increase the value to account for additional margin for setup and hold.

When you specify interclock uncertainty, ensure that you specify it for all possible

interactions of clock domains. For example, if you specify paths from CLKA to CLKB and CLKB to CLKA you must specify the uncertainty for both even if the values are the same. For an example, see the EXAMPLES section.

Interlock uncertainty is more specific than simple uncertainty. If a command that specifies interlock uncertainty conflicts with a command that specifies simple uncertainty, the command that specifies interlock uncertainty takes precedence. For an example, see the EXAMPLES section.

If there is no applicable interlock uncertainty for a path, the value for simple uncertainty is used. For an example, see the EXAMPLES section.

To remove the uncertainties set by **set_clock_uncertainty**, use the **remove_clock_uncertainty** command.

To view clock uncertainty information, use the **report_clock -skew** command.

EXAMPLES

The following example specifies that all paths leading to registers or ports clocked by CLK have setup uncertainty of 0.65 and hold uncertainty of 0.45.

```
pt_shell> set_clock_uncertainty -setup 0.65 [get_clocks CLK]
pt_shell> set_clock_uncertainty -hold 0.45 [get_clocks CLK]
```

The following example specifies interlock uncertainties between PHI1 and PHI2 clock domains.

```
pt_shell> set_clock_uncertainty 0.4 -from PHI1 -to PHI1
pt_shell> set_clock_uncertainty 0.4 -from PHI2 -to PHI2
pt_shell> set_clock_uncertainty 1.1 -from PHI1 -to PHI2
pt_shell> set_clock_uncertainty 1.1 -from PHI2 -to PHI1
```

The following example specifies interlock uncertainties between PHI1 and PHI2 clock domains with specific edges.

```
pt_shell> set_clock_uncertainty 0.4 -rise_from PHI1 -to PHI2
pt_shell> set_clock_uncertainty 0.4 -fall_from PHI2 -rise_to PHI2
pt_shell> set_clock_uncertainty 1.1 -from PHI1 -fall_to PHI2
```

The following example shows conflicting **set_clock_uncertainty** commands, one for simple uncertainty and one for interlock uncertainty. The interlock uncertainty value of 2 takes precedence.

```
pt_shell> set_clock_uncertainty 5 [get_clocks CLKA]
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

The following example specifies the uncertainty from CLKA to CLKB and from CLKB to CLKA. Notice that both must be specified even though the value is the same for both.

```
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKA] -to [get_clocks CLKB]  
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

The following example illustrates a situation in which simple uncertainty is used when there is no applicable interclock uncertainty for a path. The first command specifies a simple uncertainty of 5 for CLKA paths, and the second command specifies an interclock uncertainty of 2 for paths from CLKB to CLKA. If there are paths between CLKA and other clocks (for example, CLKC, CLKD, ...) for which interclock uncertainty has not been specifically defined, the simple uncertainty (in this case, 5) is used.

```
pt_shell> set_clock_uncertainty 5 [get_clocks CLKA]  
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] -to [get_clocks CLKA]
```

SEE ALSO

**remove_clock_uncertainty(2), report_clock(2), set_clock_latency(2),
set_clock_transition(2), timing_propagate_interclock_uncertainty(3).**

set_connection_class

Sets the connection class value on ports.

SYNTAX

```
int set_connection_connection_class  
connection_class_value  
object_list
```

```
string connection_class_value  
list object_list
```

ARGUMENTS

connection_class_value
Specifies the desired *connection_class* value of ports contained in the *object_list* option. *connection_class_value* is a single string value. This string can contain any number of space separated connection classes (see the example below).

object_list
Specifies ports whose connection classes are set.

DESCRIPTION

The *connection class* label is an attribute that is used to describe connection requirements for a given technology. Only those loads and drivers with the same connection class label can be legally connected. The labels "universal" and "default" are reserved labels. The "universal" label indicates that a pin or port can legally connect with any other load or driver. The "default" label is used for those library pins that do not otherwise have a connection class assigned to them through any library attributes. For designs being optimized analysed, the "universal" label is assigned to those ports that do not otherwise have a connection class assigned to them. To change the default connection class label for those ports, use the *default_port_connection_class* variable.

Connection classes are placed on ports of designs in a technology library by using Library Compiler. The *set_connection_class* places connection classes on ports of designs being analysed (such as the current design). This is used to indicate a connection class requirement for the network that is connected to the specified port.

To view connection class values on ports, use the *get_attribute* command. To check your design for connection class violations, use the *report_constraint* command.

EXAMPLES

The following example sets a connection class "internal" to the port named "xyz" in the current design.

```
pt_shell> set_connection_class internal xyz
```

The following example sets the connection classes "internal" and "external" on the port "out" in the design "test".

```
dc_shell> set_connection_class "internal  
                                external" test/out
```

In the following example, the connection class on a port is removed.

```
pt_shell> remove_connection_class test/out
```

SEE ALSO

remove_connection_class (2), **report_constraint** (2)

set_context_margin

Specifies the margin by which to tighten or relax constraints.

SYNTAX

```
string set_context_margin [-percent] [-relax] [-min] [-max] value [object_list]  
doublevalue  
list object_list
```

ARGUMENTS

-percent
Considers specified value as a percentage of the delay.

-relax
Relaxes, instead of tightens, the constraint.

-min
Specifies the margin for min constraints.

-max
Specifies the margin for max constraints.

value
Determines the margin in absolute or percentage value.

object_list
Indicates a list of cells or pins.

DESCRIPTION

The **set_context_margin** command specifies a margin to be added to or subtracted from input and output delay values when generated by the **write_context** command. The margin can be specified as an absolute value or as a percentage of the constraint.

The constraints are first created using the **characterize_context** or **create_timing_context** commands then written out using the **write_context** command.

By default, the input and output delays are adjusted such that they are more constraining; that is, the specified margin is added to the max delay values and subtracted from the min delay values. If you specify the **-relax** option, the margin is subtracted from the max delay values and added to the min delay values.

The margin applies to the current design if no object list is specified. When determining the margin for a pin, the value set for the pin is used if specified. If you don't specify this value, the value set for the parent cell is used. If neither value is set, the value set for the current design is used. If no margin is specified, the default value of 0.0 is used.

EXAMPLES

The following command specifies that a margin 0.5 should be added to all max values of input and output delays constrains and subtracted from all min values of input and output delay constraints. The margin applies to all objects in the current design.

```
pt_shell> set_context_margin 0.5
```

The following command specifies that the max value input delays on I2/IN should be relaxed by 10 percent; that is, input delay values will be 10 percent less.

```
pt_shell> set_context_margin -max -percent 10.0 I2/IN
```

SEE ALSO

characterize_context (2), **create_timing_context** (2), **report_context** (2), **write_context** (2).

set_coupling_separation

Create a separation constraint on nets.

SYNTAX

```
int set_coupling_separation [-pairwise pair_nets] nets  
list pnets  
list nets
```

ARGUMENTS

-pairwise

When **-pairwise *pnets*** is applied, all coupling capacitances between only ***pnets*** and ***nets*** are excluded.

DESCRIPTION

By default, all nets with coupling capacitors (non-filtered) are included in crosstalk and noise analysis as both victim and aggressor. This command creates a separation constraint on nets which means that all coupling capacitors on each net in ***nets*** are excluded for noise and crosstalk analysis.

From an analysis point of view, this command has the same effect of applying both **set_si_delay_analysis** and **set_si_noise_analysis** with the **-exclude** option and specifying ***nets*** as both victims and aggressors.

However, this constraint takes precedence over the constraints set by **set_si_delay_analysis** and **set_si_noise_analysis**.

When **-pairwise *pnets*** is applied, all coupling capacitances only between ***pnets*** and ***nets*** are excluded. This is referred to as "pairwise exclusion."

Instances of this command are recorded for output by the **write_changes** command. This feature allows the user to communicate the separation constraint to other implementation tools along with netlist changes.

The **set_coupling_separation** command returns a **1** if successful and a **0** if unsuccessful.

To remove the result of this command, use the **remove_coupling_separation** command.

To view the result of this command, use the **report_si_delay_analysis** or **report_si_noise_analysis** with the **-coupling_separated** option.

EXAMPLES

The following example sets a separation constraint on all nets referred to by **CLK_NET_***.

```
pt_shell> set_coupling_separation [get_nets CLK_NET*]  
1
```

The following example sets a separation constraint between net **n3** and **n1** and **n2**. It does not affect analysis of cross-coupling between **n1** and **n2** nor **n3** and any other net.

```
pt_shell> set_coupling_separation -pairwise [get_nets {n1 n2}] [get_nets n3]
1
```

SEE ALSO

```
remove_coupling_separation (2), report_si_delay_analysis (2),
report_si_noise_analysis (2), write_changes (2), set_si_delay_analysis (2),
set_si_noise_analysis (2).
```


set_current_power_domain

Sets the specific power domains defined by the UPF **create_power_domain** command to be included in the power analysis. As default (if not specified), the whole design covered by all the defined power domains are included in the power analysis. The commands can take one single power domain or a combination of power domains in the design.

This command has no effect on power results in non-UPF mode.

SYNTAX

```
int set_current_power_domain  
list_of_power_domains
```

Data Types

```
list_of_power_nets    list
```

ARGUMENTS

```
list_of_power_domains  
    specifies the power domains defined in the design to be included in the power  
    analysis. If power_domain_name refers to a power domain that is not defined  
    in the design, an error is issued and the current power domain setting remains  
    unchanged.
```

DESCRIPTION

The *current_power_domain* command provides the control of calculating power consumption for the domains of interest. Domain-based power consumption data can be generated for a multi power domain design. Only the power dissipated in the blocks covered by the current power domain list will be calculated and reported.

For both dynamic (internal and switching) and static (leakage) power, the domain-based power numbers are calculated based on which domain a cell belongs to. Before using this command, user has to define power domains by using UPF command *create_power_domain*. Command *get_current_power_domain* returns the power domains being included in the power analysis.

The following power analysis results will be affected by this command in UPF mode:

- average power report
- time-based power report
- power related attributes: *total_power*
 - dynamic_power*
 - internal_power*
 - switching_power*
 - leakage_power*
 - intrinsic_leakage_power*
 - gate_leakage_power*
 - x_transition_power*

```
glitch_power
peak_power
peak_time
```

Use the **set_current_power_domain [get_power_domain *]** command to resume the default behavior, which includes the whole design covered by all the defined power domains.

EXAMPLES for UPF mode

The following example shows generating domain-based power analysis results. The design has two sub-blocks "InstA" and "InstB", which are covered by power domain "PD1" and "PD2" respectively. The first **report_power** generates the power analysis results for the whole design, which is the default behavior. The second **report_power** generates the power analysis results for the power consumed in power domain "PD1". The third **report_power** generates the results for the power consumed in power domain "PD2".

```
pt_shell> ...
pt_shell> create_power_domain PD1 -elements {InstA}
pt_shell> create_power_domain PD2 -elements {InstB}
pt_shell> ...
pt_shell> report_power
pt_shell> set_current_power_domain PD1
pt_shell> report_power
pt_shell> get_current_power_domain
pt_shell> set_current_power_domain PD2
pt_shell> report_power
pt_shell> get_current_power_domain
```

SEE ALSO

```
get_current_power_domain(2) create_power_domain(2) get_power_domain(2)
report_power_domain(2) report_power(2)
```

set_current_power_net

Sets the specific power net(s) defined by UPF **create_supply_net** command(s) to be included in the power analysis. As default (if not specified), the whole design covered by all the defined supply nets are included in the power analysis. The commands can take one single power net or a combination of power nets in the design.

This command has no effect on power results in non-UPF mode.

SYNTAX

```
int set_current_power_net list_of_power_nets  
list_of_power_nets
```

Data Types

list_of_power_nets list

ARGUMENTS

list_of_power_nets
specifies the power nets defined in the design to be included in the power analysis. If *list_of_power_nets* refers to a supply net that is not defined in the design, an error is issued and the current power net setting remains unchanged.

DESCRIPTION

The *set_current_power_net* command provides the control of calculating power consumption for the power nets of interest. Power net-based power consumption data can be generated for a multi-supply design. Only the power dissipated on the cell or part of cell which is supplied by the current power net list will be calculated and reported. If a power net is specified in the command *set_current_power_net*, all the connected power nets are also included in the current power net list. The input supply net and output supply net of a power switch are not considered to be connected.

For both dynamic (internal and switching) and static (leakage) power, the power net-based power numbers are calculated based on the PG connection of the cell.

If using CCS Power model in the technology library:

- Dynamic_currents are based on *pg_pin* association;
- Leakage_currents are based on *pg_pin* association;
- Gate_leakages are based on the *related_power_pin* for the input related pins;
- Switching powers are based on the *related_power_pin* for the output pins.

If using NLPM Power model in the technology library with PG pin definitions:

- internal powers are based on *related_pg_pin* attribute;
- leakage powers are based on *related_pg_pin* attribute;
- switching powers are based on the *related_power_pin* for the output pins.

Before using this command, user has to define power nets by using UPF command `create_supply_net`. Command `report_supply_net` can be used to show the defined power nets. Command `report_pg_pin_info` and `report_power_network` can be used to show the power connections. Command `get_current_power_net` returns the supply nets being included in the power analysis.

The following power analysis results will be affected by this command in UPF mode same as command `set_current_power_domain`:

- average power report
- time-based power report
- power related attributes: `total_power`
 - `dynamic_power`
 - `internal_power`
 - `switching_power`
 - `leakage_power`
 - `intrinsic_leakage_power`
 - `gate_leakage_power`
 - `x_transition_power`
 - `glitch_power`
 - `peak_power`
 - `peak_time`

Use **`set_current_power_net [get_supply_net *]`** to resume the default behavior, which includes the whole design covered by all the defined supply nets.

EXAMPLES for UPF mode

The following example shows generating power net-based power analysis results. The design has two sub-blocks "InstA" and "InstB", which is covered by power domain PD1 and PD2 respectively. There are total 6 supply nets defined. Among them, four are power nets and two are ground nets. The primary power net for power domain "PD1" is "VDDIS", and the primary power net for power domain "PD2" is "VDDGS". The first `<b>report_power</b>` will generate the power analysis results for the whole design, which is the default behavior. The second `<b>report_power</b>` will generate the power consumption associated with power net "VDDI". The third `<b>report_power</b>` will generate the power consumption associated with power net "VDDIS", which is the output of power switch "top_header". The fourth and fifth reports are for "VDDG" and "VDDGS" respectively.

```
pt_shell> ...
pt_shell> create_power_domain PD1 -elements {InstA}
pt_shell> create_power_domain PD2 -elements {InstB}
pt_shell> create_supply_net VDDI -domain PD1
pt_shell> create_supply_net VDDIS -domain PD1
pt_shell> create_supply_net VSS -domain PD1
pt_shell> set_domain_supply_net PD1 -primary_power_net VDDIS -
primary_ground_net VSS
pt_shell> connect_supply_net -ports InstA/LS_u1/VDDL VDDI
pt_shell> connect_supply_net -ports InstA/LS_u1/VDD VDDIS
pt_shell> connect_supply_net -ports InstA/LS_u1/VSS VSS
pt_shell> create_power_switch top_header
                    -domain PD1
                    -output_supply_port {NSLEEPOUT VDDIS}
```

`set_current_power_net`

```

        -input_supply_port {NSLEEPIN VDDI}
        -on_state {state1 NSLEEPIN {CNTL}}
        -control_port { CNTL ctrl }

pt_shell> ...
pt_shell> create_supply_net VDDG -domain PD2
pt_shell> create_supply_net VDDGS -domain PD2
pt_shell> create_supply_net VSS -domain PD2 -reuse
pt_shell> set_domain_supply_net PD2 -primary_power_net VDDGS -
primary_ground_net VSS
pt_shell> create_power_switch gprs_header
        -domain PD2
        -output_supply_port {NSLEEPOUT VDDGS}
        -input_supply_port {NSLEEPIN VDDG}
        -on_state {state1 NSLEEPIN {CNTL}}
        -control_port { CNTL ctrl }

pt_shell> ...
pt_shell> report_power
pt_shell> set_current_power_net { VDDI }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDIS }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDG }
pt_shell> report_power
pt_shell> get_current_power_net
pt_shell> set_current_power_net { VDDGS }
pt_shell> report_power
pt_shell> get_current_power_net

```

SEE ALSO

get_current_power_net(2), set_current_power_domain(2), get_current_power_domain(2), create_supply_net(2), get_supply_net(2), report_supply_net(2), report_pg_pin_info(2), report_power_network(2), report_power(2).

set_data_check

Sets data-to-data checks using the specified values of setup and hold time.

SYNTAX

```
string set_data_check  
{-from from_object  
  | -rise_from from_object  
  | -fall_from from_object}  
{-to to_object  
  | -rise_to to_object  
  | -fall_to to_object}  
[-setup | -hold]  
[-clock clock_object]  
[check_value]
```

```
object from_object  
object to_object  
object clock_object  
float check_value
```

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the related pin of the data-to-data check to be set. Both rising and falling delays are checked. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

-to *to_object*

Specifies a pin or port in the current design as the constrained pin of the data-to-data check to be set. Both rising and falling delays are constrained. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising delays at the related pin. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

-fall_from *from_object*

Similar to the **-from** option, but applies only to falling delays at the related pin. You must specify one of **-from**, **-rise_from**, or **-fall_from**.

-rise_to *to_object*

Similar to the **-to** option, but applies only to rising delays at the constrained pin. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

-fall_to *to_object*

Similar to the **-to** option, but applies only to falling delays at the constrained pin. You must specify one of **-to**, **-rise_to**, or **-fall_to**.

-setup

Indicates that the data check value is for setup analysis only. If neither -

setup nor **-hold** is specified, the value applies to both setup and hold.

-hold

Indicates that the data check value is for hold analysis only. If neither **-setup** nor **-hold** is specified, the value applies to both setup and hold.

-clock *clock_object*

Specifies the name of a single clock to be used for the data check. Use this option only if your design has multiple clocks per register and you want to select a single clock to be used. For more details about multiple clocks per register, see the DESCRIPTION section.

check_value

Specifies the value of the setup and/or hold time for the check.

DESCRIPTION

The **set_data_check** command specifies a data-to-data check to be performed between the from object and to object using the specified setup and/or hold value.

The pulse relation between clocks of related pin and constrained pin is considered to be zero cycles. The normal sequential check uses one cycle.

The data check is treated as non-sequential; that is, the path goes through the related and constrained pins and is not broken at these pins. To report the constraint related to the data check, use **report_timing -to data_check_constrained_pin**.

The data signals arriving at the constrained pin could come from startpoints with multiple clocks, in one of these ways:

1. Any given latch has multiple clocks propagating to the clock or gate pin.
2. When each latch has only one clock propagating to the clock or gate pin, different latches have different clocks propagating to their clock or gate pins. For such cases, these clock relations are checked separately. Similarly, data signals arriving at the related pin could also come from startpoints with multiple clocks. You use the **-clock** option to select a specific clock at the related pin for the analysis. If a single clock is not selected, by default multiple clocks are analyzed and grouped into respective clock groups. You can change this default behavior by setting the **timing_enable_multiple_clocks_per_reg** variable to false (the default is true), which causes PrimeTime to select a single clock at random for analysis. The value of this variable has no effect on the **-clock** option.

In some cases, the signals that arrive at the related pin are launched by a clock signal that goes through several levels of generation circuitry. If multiple levels of flip-flops are involved in the clock generation, and you do not define generated clocks, signal propagation is blocked by the flip-flops or latches if the **timing_propagate_through_unlocked_registers** variable is set to its default value of false. In this case, set the variable to true; then, the signals propagate through each unlocked flip-flop or latch, enabling the analysis of the data check.

Even though PrimeTime does not complain when the user defines a data-to-data check in the clock network (mainly because the tool has not yet analyzed what is clock and

what is data,) the application and correct reporting of these checks may be incorrect. Data-to-data checks expect to operate on two data signals and not expected to be used in lieu of clock skew checks. So, for correct, supported behavior, the user should refrain from setting data-to-data checks in the clock network.

Note that if best-case/worst-case (bc_wc) operating conditions are preset, the same operating type is used for both the constrained and the reference data paths. That is, when determining the setup slack for a data-to-data check, PrimeTime uses the max operating condition for both data paths. For sign-off, the user is encouraged to use on-chip variations operating mode to resolve data-to-data check setup and hold slacks.

To remove information set by **set_data_check**, use the **remove_data_check** command.

EXAMPLES

The following example creates a data-to-data check from and1/B to and1/A with respect to the rising edge of signal at and1/B, using a setup and hold time of 0.4.

```
pt_shell> set_data_check -rise_from and1/B -to and1/A 0.4
```

The following example creates a data-to-data check from and1/B to and1/A with respect to the rising edge of signal at and1/B, coming from the clock domain of CK1, constraining only the falling edge of signal at and1/A, using a setup time of 0.5 and no hold check.

```
pt_shell> set_data_check -rise_from and1/B -fall_to and1/A -setup \  
-clock [get_clock CK1] 0.5
```

SEE ALSO

```
remove_data_check (2), report_timing (2), update_timing (2);  
timing_enable_multiple_clocks_per_reg (3),  
timing_propagate_through_unclocked_registers (3).
```


set_delcalc_resource

Sets the value of an arbitrary variable to be used by the delay calculators.

SYNTAX

```
int set_delcalc_resource resource_name resource_value  
string resource_name  
string resource_value
```

ARGUMENTS

resource_name
Specifies the name for the resource.

resource_value
Specifies the value for the resource.

DESCRIPTION

Sets the value of an arbitrary variable to be used by the delay calculators. Currently, only the DCM delay calculator needs this capability. The first argument is the resource name, and the second argument is the resource value.

EXAMPLES

This example sets the variable of "resource1" to "value1".

```
pt_shell> set_delcalc_resource resource1 value1
```

SEE ALSO

remove_delcalc_resource (2), **list_delcalc_resources** (2),

set_design_top

Sets or gets the current design in PrimeTime. It is a synonym for the **current_design** command.

SYNTAX

```
string set_design_top [design_name]
```

Data Types

design_name string

ARGUMENTS

design_name
Specifies the working or focal design for many PrimeTime commands. If the *design_name* option is not specified, the **current_design** command returns a collection containing the current design. If *design_name* refers to a design that cannot be found, an error is issued and the working design remains unchanged.

DESCRIPTION

The UPF specification defines the **set_design_top** command that is in PrimeTime - implemented as a synonym to the **current_design** command.

Note: The **current_design** command removes all assertions (such as SDC, SPEF, and UPF). Therefore, the **set_design_top** command cannot be arbitrarily used in UPF scripts. It has to be used as the first command in UPF script, which is also used before any SDC or other assertions.

SEE ALSO

`current_design(2)`

set_disable_clock_gating_check

Disables the clock gating check for specified objects in the current design.

SYNTAX

```
string set_disable_clock_gating_check object_list  
list object_list
```

ARGUMENTS

object_list
Specifies a list of cells and pins for which the clock gating check is to be disabled.

DESCRIPTION

Disables the clock gating check for specified cells and pins in the current design. This command will only disable auto-inferred clock gating checks. Clock gating checks from library will not be disabled.

Any cell or pin in the current design or its subdesigns can be disabled. Cells at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name*. Pins at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name/pin_name*.

Note: For subdesigns in the hierarchy, you must specify instance names instead of design names.

When the checking through a cell is disabled, all gating checks in the cell are disabled. When the checking through a pin is disabled, any gating check is disabled if it uses the disabled pin as a gating clock pin or a gating enable pin.

To restore gating checks disabled by **set_disable_clock_gating_check**, use **remove_disable_clock_gating_check**. To globally disable all clock gating checks, set the **timing_disable_clock_gating_checks** variable to *true*.

EXAMPLES

The following example disable gating checks on the specified cell or pin.

```
pt_shell> set_disable_clock_gating_check an2/Z  
pt_shell> set_disable_clock_gating_check or1
```

SEE ALSO

remove_disable_clock_gating_check (2), **timing_disable_clock_gating_checks** (3).

set_disable_timing

Disables timing arcs in a circuit.

SYNTAX

```
string set_disable_timing  
[-from from_pin_name -to to_pin_name]  
object_list
```

```
string from_pin_name  
string to_pin_name  
list object_list
```

ARGUMENTS

-from "*from_pin_name*"

Specifies that arcs only from this pin on the specified cell are to be disabled. The *from_pin_name* value must be a valid library pin name corresponding to the cell in *object_list*. When used, the **-from** and **-to** options must be specified together. The *object_list* must contain only cells, and the command specifies that arcs only between these two pins on the specified cells are to be disabled.

-to "*to_pin_name*"

Specifies that arcs only to this pin on the specified cell are to be disabled. The *to_pin_name* value must be a valid library pin name corresponding to the cell in *object_list*. When used, the **-from** and **-to** options must be specified together. The *object_list* must contain only cells, and the command specifies that arcs only between these two pins on the specified cells are to be disabled.

object_list

Specifies a list of cells, pins, lib-cells, lib-pins, ports, or timing-arcs to be disabled. This list can include library objects.

DESCRIPTION

Disables timing through the specified cells, pins, ports, or timing arcs in the **current_design**.

Any cell, pin, port, or timing arc in the **current_design** or its subdesigns can be disabled. Cells at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name*. Pins at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name/pin_name*. Ports at lower levels in the design hierarchy are specified as *instance1/instance2/cell_name/port_name*. Timing arcs have to be collected using the *get_timing_arcs* command.

Note: For subdesigns in the hierarchy, you must specify instance names instead of design names.

When the timing through a cell is disabled, only those cell arcs are disabled that

lead to the output pins of the cell. When the timing through a pin or port is disabled, only those cell arcs that lead from a load pin and to a driver pin are disabled. For bidirectional pins or ports the cell arcs from the load side and to the driver side are disabled.

When **-from** and **-to** pins are specified, all arcs between these two pins on the cell are disabled.

To view disabled timing arcs in the current design, use the **report_disable_timing** command. To restore timing arcs disabled by **set_disable_timing**, use the **remove_disable_timing** command. The **reset_design** command removes all user-specified attributes from the current design, including those set by **set_disable_timing**.

EXAMPLES

Consider the path reported below.

```
pt_shell> report_timing -input_pins
```

```
*****
Report : timing
        -path full
        -delay max
        -input_pins
        -max_paths 1
Design : counter
Version: 2002.03-Beta3
Date   : Wed Feb 13 12:05:38 2002
*****
```

Startpoint: ffb (rising edge-triggered flip-flop)
Endpoint: CO (output port)
Path Group: (none)
Path Type: max

Point	Incr	Path

ffb/CP (FD3)	0.00	0.00 r
ffb/QN (FD3)	2.42	2.42 r
m/C (AN4)	0.00	2.42 r
m/Z (AN4)	1.12	3.54 r
CO/A (AN2)	0.00	3.54 r
CO/Z (AN2)	0.48	4.02 r
CO (out)	0.00	4.02 r
data arrival time		4.02

(Path is unconstrained)

Assuming you want to disable timing arcs through cell **CO**, you can view all timing arcs of **CO** by examining its library cell, **AN2**.

```
pt_shell> report_lib -timing lsi_10k { AN2 }
```

```

*****
Report : library
        -timing_arcs
Library: lsi_10k
Version: 2002.03-Beta3
Date   : Thu Feb 14 09:19:43 2002
*****

```

Library Cells:

```

Attributes:
b - black box (function unknown)
d - dont_touch
s - state table
u - dont_use
A - abstracted timing model
E - extracted timing model
I - Interface timing spec model (ITS)
S - Stamp timing model
Q - Quick timing model (QTM)

```

Lib	Cell	Attributes	Arc		Arc Pins		When
			#	Type/Sense	From	To	

AN2			0	positive_unate	A	Z	
			1	positive_unate	B	Z	

To disable, the timing arc used in the path reported above, use `set_disable_timing` as follows:

```
pt_shell> set_disable_timing -from A -to Z [ get_cells { CO } ]
```

As a result of the above disabling, the `report_timing` command now finds a new longest path.

```
pt_shell> report_timing
```

```

*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : counter
Version: 2002.03-Beta3
Date   : Thu Feb 14 09:21:12 2002
*****

```

```

Startpoint: ffa (rising edge-triggered flip-flop)
Endpoint: QA (output port)
Path Group: (none)
Path Type: max

```

Point	Incr	Path

ffa/CP (FD2)	0.00	0.00 r
ffa/Q (FD2)	1.42	1.42 f
QA/Z (IV)	0.38	1.80 r
QA (out)	0.00	1.80 r
data arrival time		1.80

(Path is unconstrained)		

Alternatively, the same arc can be disabled as follows:

```
pt_shell> set_disable_timing [get_timing_arcs -from { CO/A } -to { CO/Z } ]
```

SEE ALSO

```
remove_disable_timing (2), report_disable_timing (2), report_lib (2), report_timing (2), reset_design (2), get_timing_arcs (2).
```

set_distributed_parameters

Configures the distributed environment.

SYNTAX

Boolean **set_distributed_parameters**

```
[-script script]
[-shell rsh | remsh | ssh]
[-collection_levels collection_level]
```

ARGUMENTS

[-script script]

The user defined / customized wrapper script to be used when launching remote processes. If this option is not specified then the wrapper is taken by default to be \$synopsys_root/bin/pt_shell. This applies to both DMSA and multi_core distributed environments.

[-shell rsh | remsh | ssh]

The type of remote shell to be used when launching remote processes.

[-collection_levels collection_level]

Specifies the number of collection levels to traverse when retrieving collection attributes from the master. The higher this collection level will be set, the more memory will be potentially consumed at the master.

DESCRIPTION

The **set_distributed_parameters** command allows the user to configure the distributed environment. The command should be issued before the **create_distributed_farm** or the **start_hosts** command for user configurations to take effect.

EXAMPLES

In the following example a distributed processor is allocated from a workstation called platinum1. The set_distributed_parameters command is used to configure the shell type script for the launching of the processor. The processor is launched using the **create_distributed_farm** command.

```
pt_shell> add_distributed_processors -farm now -32bit platinum1
1
pt_shell> set_distributed_parameters -shell ssh -script /usr/pt_shell
1
pt_shell> create_distributed_farm
1] Launched: ssh -n -l user platinum1 /usr/pt_shell -multi_scenario -
32bit  -__zdpp_slave platinum1 16874 16859 1 /usr/work PrimeTime 4106 4108 53205
      Status: Forking successful
      Stdout: your job
      Stderr: **<<EMPTY>>**
```



```
-----  
-----  
Waiting for 1 (of 1) distributed processors  
-----  
-----
```

In the following example we use the `-collection_levels` option to restrict the amount of information returned by the `get_distributed_variable` command. By setting the collection level to 4 only 4 levels of collections can be traversed.

```
pt_shell> set_distribtued_parameters -collection_level 4  
pt_shell> remote_execute { set m_pins [get_pins A]}  
pt_shell> get_distributed_variables m_pins -attr {clocks sources}  
pt_shell> echo [set clock1 [get_attribute $m_pins(scen1) clocks]]  
_sel12  
pt_shell> echo [set clock_source1 [get_attribute $clock1 sources]]  
_sel13  
pt_shell> echo [set clock2 [get_attribute $clock_source1 clocks]]  
_sel14  
pt_shell> echo [set clock_source2 [get_attribute $clock2 sources]]  
_sel15  
pt_shell> echo [set clock3 [get_attribute $clock_source2 clocks]]  
Warning: Attribute 'clocks' does not exist on port 'CLK' (ATTR-3)
```

SEE ALSO

`create_distributed_farm` (2), `add_distributed_processors` (2).

set_distributed_variables

Passes Tcl variables from the master to the slaves.

SYNTAX

Boolean **set_distributed_variables**
[-pre_commands pre_command_string]
[-post_commands post_command_string]

Data Types

<i>pre_command_string</i>	string
<i>post_command_string</i>	string

ARGUMENTS

-pre_commands *pre_command_string*
A string of commands to be executed in the slave context before the setting of variables. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

-post_commands *post_command_string*
A string of commands to be executed in the slave context after the setting of variables. Commands are grouped using the ";" character. The maximum size of a command is 1000 chars.

DESCRIPTION

The **set_distributed_variables** command sets variables at the slaves given a set of variables at the master. If the variable to be sent is a simple Tcl list, the Tcl list is recreated at the slaves for all scenarios in focus. If the variable to be sent to the master is an array indexed by the scenario name, a variable is set at the slaves for each scenario in focus that has an entry in the array at the master. In this case, the data can be either a Tcl list or a Tcl array.

EXAMPLES

In the following example, you set a **scenario_id** variable for each scenario and a variable **max_fanout** in both scenarios. On scenario **scen1** the **scenario_id** variable is created and is set to a value of 1. On scenario **scen2** the **scenario_id** variable is created and is set to a value of 2. On both scenario **scen1** and scenario **scen2**, the **max_fanout** variable is created and is set to a value of 10.

```
pt_shell> set max_fanout 10
pt_shell> array set scenario_id {scen1 1 scen2 2}
pt_shell> set_distributed_variables {scenario_id max_fanout}
```

SEE ALSO

`get_distributed_variables(3)`

set_domain_supply_net

Set the primary power net and primary ground net of an existed power_domain.

SYNTAX

```
int set_domain_supply_net  
domain_name  
-primary_power_net supply_net_name  
-primary_ground_net supply_net_name
```

Data Types

<i>domain_name</i>	string
<i>supply_net_name</i>	string

ARGUMENTS

domain_name
Specify the name of the power domain to be set.
If there is not such a power domain with the specified name in current scope, the command fails.

-primary_power_net *supply_net_name*
Specify the power net of the power domain.
If there is not such a supply net with the specified name in current scope, the command fails. If the supply net is not associated with specified power domain, the command also fails.

-primary_ground_net *supply_net_name*
Specify the ground net of the power domain.
If there is not such a supply net with the specified name in current scope, the command fails. If the supply net is not associated with specified power domain, the command also fails.

DESCRIPTION

The *set_domain_supply_net* command enables you to set the primary power net and the primary ground net of a power domain. All extent of the power domains shares the same power/ground supply until explicitly excluded. Here "explicitly excluded" means it is explicitly connected with another supply_net (non primary power/ground supply_net) by commands *connect_supply_net*, *set_isolation*, *set_retention*. This command fails if domain already has a primary power and ground net.

The supply_net must be created in the same power_domain at first before it could be set as the primary power/ground supply_net. Use command *create_supply_net* to create a supply_net at a specified power_domain.

Command returns 1 if succeed, and returns 0 otherwise.

EXAMPLES

The following example creates two `supply_net` on power_domain PD1, then set them as primary power/ground net.

```
prompt> create_power_domain PD1 -elements INST_1
PD1
ptompt> create_supply_net A_VDD -domain PD1
A_VDD
ptompt> create_supply_net A_VSS -domain PD1
A_VSS
ptompt> set_domain_supply_net PD1 -primary_power_net A_VDD
-primary_ground_net A_VSS
1
```

SEE ALSO

```
create_power_domain(2)
create_supply_net(2)
connect_supply_net(2)
help UPF
help "power domains"
```

set_dont_touch

Sets the **dont_touch** attribute on cells, nets, designs, and library cells to prevent synthesis from replacing or modifying them during optimization.

SYNTAX

```
string set_dont_touch  
object_list [value]
```

```
list object_list  
Booleanvalue
```

ARGUMENTS

object_list
Specifies a list of cells, nets, designs, or library cells on which to place the **dont_touch** attribute.

value
Specifies the value with which to set the **dont_touch** attribute. Allowed values are *true* (the default) or *false*.

DESCRIPTION

Sets the **dont_touch** attribute on cells and nets in the current design, or on designs and library cells, to prevent these objects from being modified or replaced by Design Compiler during optimization.

This attribute is characterized by the **characterize_context** and **create_timing_context** commands, and is exported to synthesis using the **write_context** command.

When the **dont_touch** attribute on a design or library cell is written to scripts, the target objects vary based on the script format. A design **dont_touch** is applied to a design for PrimeTime and to a reference for Design Compiler. Similarly, a library cell **dont_touch** is applied to a lib_cell for PrimeTime, and to a reference for Design Compiler. For an example, see the EXAMPLES section.

If a **dont_touch** attribute is on a design that is removed, the attribute continues to apply to any cells that were instances of that design, until the next time the current design is linked.

The **dont_touch** attribute on cells, nets, and designs is imported from db files.

For a complete description of the **dont_touch** attribute and its effect, see the man page of the **set_dont_touch** command in Design Compiler.

Inheritance

The **dont_touch** attribute is inherited by cells if the cell is an instance of a design or library cell that has the **dont_touch** attribute. In these cases,

get_attribute returns *true* for the **dont_touch** attribute for that cell. This affects the result of **get_attribute** but does not actually transfer the **dont_touch** attribute to the cell, so the output of **write_script** does not contain extra **set_dont_touch** commands applied to cells. For examples, see the EXAMPLES section.

Limitations

Currently, if your database contains a **dont_touch** attribute on an object, and you explicitly use **set_dont_touch** to remove the attribute by setting the value to *false* for that object, the output of **write_script** is inaccurate because this information is not written. This limitation will be addressed in an upcoming release.

EXAMPLES

The following commands specify not to modify the "block1" and "analog1" cells during synthesis, but allow the net "N1" to be modified.

```
pt_shell> set_dont_touch [get_cells {block1 analog1}]
1
pt_shell> set_dont_touch [get_nets N1] false
1
```

In the following example, applying a **dont_touch** attribute to design d1 affects its instances in that **get_attribute** returns *true* for **dont_touch**. Note the difference in **write_script** output for PrimeTime and Design Compiler. Note that **write_script** output is excerpted.

```
pt_shell> get_attribute [get_cells u1] ref_name
d1
pt_shell> get_attribute [get_cells u1] dont_touch
false
pt_shell> set_dont_touch [get_designs d1]
1
pt_shell> get_attribute [get_cells u1] dont_touch
true
pt_shell> write_script
set_dont_touch [get_designs "design1.db:d1"]
pt_shell> write_script -format dcsh
set_dont_touch find(reference, "d1")
pt_shell>
```

SEE ALSO

characterize_context (2), **create_timing_context** (2), **get_attribute** (2),
set_dont_touch_network (2), **write_context** (2), **write_script** (2).

set_dont_touch_network

Sets the dont_touch attribute on clock networks for synthesis.

SYNTAX

```
string set_dont_touch_network object_list  
list object_list
```

ARGUMENTS

object_list
Specifies a list of clocks, pins, or ports.

DESCRIPTION

Sets the dont_touch_network attribute on clocks, pins, or ports in the current design. When a design is optimized, synthesis assigns dont_touch attributes to all cells and nets in the transitive fanout of dont_touch_network objects so that they are not modified or replaced during optimization.

The **set_dont_touch_network** command is intended primarily for clock circuitry. Placing a dont_touch_network on a clock object prevents synthesis from modifying the clock buffer network.

This attribute is characterized by the **characterize_context** and **create_timing_context** commands and is exported to synthesis using the **write_context** command.

Refer to the man page of the **set_dont_touch** command in Design Compiler for a complete description of the dont_touch attribute and its effect.

Use the **reset_design** command to remove the dont_touch_network attribute.

EXAMPLES

The following command adds a dont_touch_network attribute to the port named "clock_in".

```
pt_shell> set_dont_touch_network [get_ports clock_in]
```

SEE ALSO

creating_timing_context (2), **characterize_context** (2), **report_context** (2), **write_context** (2), **set_dont_touch** (2).

set_drive

Sets the resistance to a specified value on specified input or inout ports in the current design.

SYNTAX

```
string set_drive
    [-rise] [-fall]
    [-min] [-max]
    resistance_value port_list
```

```
float resistance_value
list port_list
```

ARGUMENTS

-rise
Indicates that *resistance_value* is to be used to drive the ports only for the rising case.

-fall
Indicates that *resistance_value* is to be used to drive the ports only for the falling case.

-min
Applies only to designs in min-max mode (min and max operating conditions). Indicates that the *resistance_value* is the minimum resistance.

-max
Applies only to designs in min-max mode (min and max operating conditions). Indicates that the *resistance_value* is the maximum resistance.

resistance_value
Specifies a nonnegative port drive resistance value for the ports in *port_list*. The value must be ≥ 0 ; units must be the same as those in the technology library.

port_list
Specifies a list of input or inout ports in the current design, on which the *resistance_value* is to be set.

DESCRIPTION

Sets the resistance to a specified value on specified input or inout ports in the current design. PrimeTime models the driver as a resistance and uses that value to calculate the wire delay of the port. To view the drive that is set, use **report_port -drive**.

Depending on the options used, **set_drive** sets these attributes on the specified ports:

drive_resistance_fall_max
drive_resistance_fall_min
drive_resistance_rise_max
drive_resistance_rise_min

If **set_drive** is issued without any options, the **drive_resistance_fall_max** and **drive_resistance_rise_max** attributes are set; in min-max mode, the other two attributes are also set.

If **set_drive** is issued with only the **-rise** option, only the **drive_resistance_rise_max** attribute is set; in min-max mode, **drive_resistance_rise_min** is also set.

If **set_drive** is issued with only the **-fall** option, only the **drive_resistance_fall_max** attribute is set; in min-max mode, **drive_resistance_fall_min** is also set.

Drive resistance is the output resistance of the cell that drives the port, so that a higher drive resistance means less drive capability and longer delays. Thus, a *drive_resistance_value* of 0 is infinite drive, or no delay between the ports and all ports connected to them. Setting *resistance_value* to 0 is the same as removing the drive resistance with **remove_drive_resistance**.

There are two other methods of describing port drive capability. **set_driving_cell** command or the **set_input_transition** command. The most recent drive command has precedence.

EXAMPLES

The following example sets the drive resistance to 5 on all input ports in the current design, and displays the ports on which the drive resistance has been set.

```
pt_shell> set_drive 5 [all_inputs]
1
pt_shell> report_port -drive
*****
Report : port
        -drive
Design : counter
*****
```

Input Port	Resistance		Transition	
	Rise	Fall	Rise	Fall
A	5.00	5.00	--	--
B	5.00	5.00	--	--
C	5.00	5.00	--	--
CL	5.00	5.00	--	--
CLK	5.00	5.00	--	--
D	5.00	5.00	--	--
JOKE	5.00	5.00	--	--
L	5.00	5.00	--	--
P	5.00	5.00	--	--
RESET	5.00	5.00	--	--

T	5.00	5.00	--	--

SEE ALSO

`remove_drive_resistance (2), report_port (2), set_load (2).`

set_drive_resistance

Sets drive resistance for input or inout ports.

Note: This command is obsolete, and has been replaced by the **set_drive** command. Use **set_drive** instead.

SYNTAX

```
string set_drive_resistance [-rise]
[-fall]
[-min]
[-max]
resistance
port_list
```

```
float resistance
list port_list
```

ARGUMENTS

-rise
Specifies to use the resistance to drive the ports for the rising case.

-fall
Specifies to use the resistance to drive the ports for the falling case.

-min
Specifies to use the resistance to drive the ports for the minimum case.

-max
Specifies to use the resistance to drive the ports for the maximum case.

resistance
Specifies a nonnegative resistance value for the ports. Port drive resistance (value ≥ 0).

port_list
Specifies a list of input or inout port names of the current design on which the drive attributes are to be set.

DESCRIPTION

Model the driver as a resistance and use that value to calculate the wire delay of the port. **report_port -drive** can be used to view the drive which is set.

Drive resistance is the output resistance of the cell that drives the port; such that a higher drive resistance means less drive capability and longer delays. Thus, a drive *resistance* of 0 is infinite drive, or no delay between the ports and all connected to them. This is the same as removing the drive resistance with **remove_drive_resistance**.

Drive resistance must be in units with the technology library.

EXAMPLES

```
pt_shell> set_drive_resistance 5 [all_inputs]
1
```

```
pt_shell> report_port -drive
```

```
*****
```

```
Report : port
        -drive
```

```
Design : counter
```

```
*****
```

Input Port	Resistance		Transition	
	Rise	Fall	Rise	Fall
A	5.00	5.00	--	--
B	5.00	5.00	--	--
C	5.00	5.00	--	--
CL	5.00	5.00	--	--
CLK	5.00	5.00	--	--
D	5.00	5.00	--	--
JOKE	5.00	5.00	--	--
L	5.00	5.00	--	--
P	5.00	5.00	--	--
RESET	5.00	5.00	--	--
T	5.00	5.00	--	--

SEE ALSO

```
remove_drive_resistance (2), report_port (2), set_drive (2), set_load (2).
```

set_driving_cell

Sets the port driving cell.

SYNTAX

```
string set_driving_cell  
[-lib_cell lib_cell_name]  
[-rise]  
[-fall]  
[-min]  
[-max]  
[-library lib_name]  
[-pin pin_name]  
[-from_pin from_pin_name]  
[-multiply_by factor]  
[-dont_scale]  
[-no_design_rule]  
[-input_transition_rise rtrans]  
[-input_transition_fall ftrans]  
[-clock clock_name]  
[-clock_fall]  
port_list
```

```
string lib_cell_name  
string lib_name  
string pin_name  
string pin_name  
float factor  
float rtran  
float ftran  
[-clock clock_name]  
[-clock_fall]  
list port_list
```

ARGUMENTS

-lib_cell *lib_cell_name*
Specifies the name of the library cell used to drive the ports. You must use the **-pin** option if the cell has more than one output pin. If different cells are needed for the rising and the falling cases, use separate commands with the **-rise** or **-fall** option. Use the **-from_pin** option to choose between multiple input pins with arcs to this output pin. This option is required.

-rise
Sets driving_cell information for only the rising port transition.

-fall
Sets driving_cell information for only the falling port transition.

-min
Sets driving_cell information for only the minimum analysis. This option is valid only in min-max mode.

-max
Sets driving_cell information for only the maximum analysis. This option is valid even when not in min-max mode. When not in min-max mode, the option is not required because it is the default.

-library *lib_name*
Specifies the name of the library containing the *library_cell_name* value. This is the library of the driving cell. By default, the libraries in **link_library** are searched for the cell.

-pin *pin_name*
Specifies the output pin whose drive is used. This is the driving pin name. If you do not include the **-from_pin** option, the command uses an arbitrary pin arc ending at the specified pin.

-from_pin *from_pin_name*
Specifies an input pin on the specified cell so the command uses the drive of the timing arc from this pin to the specified pin.

-multiply_by *factor*
Multiplies the calculated transition time by the specified multiplier. The valid transition multiplier range is greater than or equal to 0.

-dont_scale
Prevents operating condition scaling. Indicates that the timing analyzer is not to scale the drive capability of the ports according to the current operating conditions. By default, the port drive capability is scaled for operating conditions exactly as the driving cell itself would have been scaled.

-no_design_rule
Specifies not to transfer design rules from the driving cell to the port. If you do not include this option, the design rules (such as max_capacitance) of the library pin are applied to the port.

-input_transition_rise *rtran*
Specifies the input rising transition time associated with the **-from_pin** option. If you do not include this option, the default value is 0. Use the **-input_transition_rise** and **-input_transition_fall** options to capture the accurate transition time associated with the *from_pin_name* value. This can obtain more accurate information on the transition time and delay time at the output pin.

-input_transition_fall *ftran*
Specifies the input falling transition time associated with the *from_pin_name* value. If you do not include this option, the default value is 0.

-clock *clock_name*
The driving cell is set relative the specified clock.

-clock_fall
Specifies that driving cell is relative to the falling edge of the clock. The default is the rising edge.

port_list

Provides a list of input ports. The list contains input or inout port names in the current design on which the driving cell information is set.

DESCRIPTION

Sets the port driving cell. Sets attributes on the specified input or inout ports in the current design to associate an external driving cell with the ports. The drive capability of the port is the same as if the specified driving cell were connected in the same context to allow accurate modeling of port drive capability for nonlinear delay models. If you do not include the **-dont_scale** option, the drive capability of the port is scaled according to the current operating conditions. Use the **report_port** command with the **-drive** option to view drive information on ports.

The driving cell can be relative to a clock by using **-clock** option. This means the driving cell apply only to those external paths driven by the clock. Using **-clock** or **-clock_fall** option can be meaningful only when **timing_slew_propagation_mode** is in **worst_arrival** mode. Note if it is in **worst_slew** mode, there must be a driving cell without any clock specified to see any driving cell on the port. If a clock-relative driving cell is set on a port, these driving-cell attributes will not be accessible via **get_attribute** until **-clock** and **-clock_fall** options are supported for **set_driving_cell** in SDC.

Use the **characterize_context** command to automatically set driving cell information on subdesign ports based on their context in the entire design. Use the **remove_driving_cell** command to remove driving cell information from ports.

Note: There are two other methods of describing port drive capability. **set_drive** command or the **set_input_transition** command. The most recent drive command has precedence. If possible, always use the **set_driving_cell** command instead of the **set_drive** command, because **set_driving_cell** allows accurate calculation of port delay and transition time for library cells with nonlinear dependence on capacitance.

EXAMPLES

```
pt_shell> set_driving_cell \  
-lib_cell AND [get_ports A]  
pt_shell> report_port -drive A
```

```
***** Report : port -drive Design : counter  
Version: 1997.08 Date : Tue 1996 *****
```

```
Resistance Transition Input Port Rise Fall Rise Fall -----  
----- A -- -- -- --
```

```
Driving Cell Input Port Rise Fall Mult Attrs -----  
----- A AND AND --
```

SEE ALSO

all_inputs (2), **characterize_context** (2), **remove_driving_cell** (2), **report_port** (2), **reset_design** (2), **set_capacitance** (2), **set_drive** (2), **set_input_transition** (2).

set_equal

Sets two ports to be logically equivalent.

SYNTAX

```
string set_equal port1 port2
stringport1
stringport2
```

ARGUMENTS

port1
Specifies the first input port.

port2
Specifies another input port.

DESCRIPTION

Defines two input ports in the current design as logically equivalent. This information is used during synthesis to eliminate redundant ports; improving optimization quality.

Logical inconsistencies are checked for in relationships between ports. Specifying an inconsistent logical relationship between ports is not allowed.

Use the **reset_design** command to remove this property from a port.

This attribute is characterized by the **characterize_context** and **create_timing_context** commands and is exported to synthesis using the **write_context** command.

EXAMPLES

The following example sets two input ports "A" and "B" logically equal.

```
pt_shell> set_equal A [get_ports B]
1
```

The following example sets up a contradictory relationship between two ports and creates an error.

```
pt_shell> set_equal A B
1
pt_shell> set_opposite A B
Error: Can't set equal ports opposite in design 'top': 'A' 'B'. (DDB-22)
0
```

SEE ALSO

`set_opposite` (2), `creating_timing_context` (2), `characterize_context` (2),
`write_context` (2).

set_false_path

Identifies paths in a design that are to be marked as false, so that they are not considered during timing analysis.

SYNTAX

```
Boolean set_false_path
[-setup] [-hold]
[-rise] [-fall]
[-reset_path]
[-from from_list
    | -rise_from rise_from_list
    | -fall_from fall_from_list]
[-through through_list]
[-rise_through rise_through_list]
[-fall_through fall_through_list]
[-to to_list
    | -rise_to rise_to_list
    | -fall_to fall_to_list]

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
```

ARGUMENTS

-setup

Indicates that setup (maximum) paths are to be marked as false. This option disables setup checking for specified paths. If you do not specify either **-setup** or **-hold**, both setup and hold timing are marked false.

-hold

Indicates that hold (minimum) paths are to be marked as false. This option disables hold checking for specified paths. If you do not specify either **-setup** or **-hold**, both setup and hold timing are marked false.

-rise

Indicates that rising delays are to be marked as false, as measured on the path endpoint. If you do not specify either **-rise** or **-fall**, both rise and fall timing are marked false.

-fall

Indicates that falling delays are to be marked as false, as measured on the path endpoint. If you do not specify either **-rise** or **-fall**, rise and fall timing are marked false.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise/fall setup/hold type is reset. This is equivalent to using the **reset_path** command with similar arguments before the **set_false_path** command is issued.

-from from_list

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from rise_from_list

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from fall_from_list

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-through through_list

Specifies a list of pins, ports, cells or nets through which the disabled paths must pass. Nets are interpreted to imply the leaf-level driver pins. If you do not specify any type of **through** option, all timing paths specified using the **-from** and **-to** options are affected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

set_false_path

`-rise_through` *rise_through_list*

The same as the **-through** option, but the paths must have a rising transition at the through points.

`-fall_through` *fall_through_list*

The same as the **-through** option, but the paths must have a falling transition at the through points.

`-to` *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, one path endpoint on that cell is affected.

You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

`-rise_to` *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

`-fall_to` *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

DESCRIPTION

The **set_false_path** command marks startpoint/endpoint pairs as false timing paths; that is, paths that cannot propagate a signal. The command removes timing constraints on these false paths, so that they are not considered during timing analysis. Path startpoints are input ports or register clock pins; path endpoints are register data pins or output ports. The command disables maximum delay (setup) checking and minimum delay (hold) checking for the specified paths.

The **set_false_path** command is a point-to-point timing exception command, and overrides the default single-cycle timing relationship for one or more timing paths. False path information always takes precedence over multicycle path information. Also, **set_false_path** commands override **set_max_delay** and **set_min_delay** commands.

To disable the timing at a particular cell along a path, use the **set_disable_timing** command.

To remove the false path designations set by **set_false_path**, use the **reset_path** command.

For crosstalk analysis the false paths are treated as sensitizable. So, the

aggressors on the false paths could still effect its victims. When the clocks are asynchronous to each other, **set_clock_groups -async** command should be used instead of **set_false_paths** between the async clocks. With out **set_clock_groups -async** command, the clocks will be treated to be sync to each other.

EXAMPLES

The following example disables timing paths from ff12 to ff34.

```
pt_shell> set_false_path -from ff12 -to ff34
```

The following example disables rising timing paths from U14/Z to ff29/Reset.

```
pt_shell> set_false_path -rise_from U14/Z -to ff29/Reset
```

The following examples disables hold checking for endpoints clocked by CLK1.

```
pt_shell> set_false_path -hold -to [get_clocks CLK1]
```

The following example disables all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C}.

```
pt_shell> set_false_path -from ff1/CP -through {U1/Z U2/Z}
          -through {U3/Z U4/C} -to ff2/D
```

The following example disables all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall to one or more of {U3/Z U4/C}.

```
pt_shell> set_false_path -from ff1/CP
          -rise_through {U1/Z U2/Z}
          -fall_through {U3/Z U4/C} -to ff2/D
```

SEE ALSO

current_design(2), reset_design(2), reset_path(2), set_clock_groups(2),
set_disable_timing(2), set_max_delay(2), set_min_delay(2), set_multicycle_path(2).

set_fanout_load

Sets fanout_load for output ports in the current design.

SYNTAX

```
string set_fanout_load value port_list  
float value  
list port_list
```

ARGUMENTS

value

Shows **fanout_load** value, in units consistent with the **fanout_load** and **max_fanout** values in the technology library.

port_list

Provides a list of output ports.

DESCRIPTION

Specifies a **fanout_load** for output ports in the current design. By default, ports are considered to have fanout_load of 0.0. Fanout load for a net is the sum of **fanout_load** attributes for all input pins and output ports connected to the net. Output pins may have maximum fanout limits, specified in the library or with the **set_max_fanout** command.

The **report_constraint -max_fanout** command shows maximum fanout constraint evaluations. **report_port -design_rule** shows port **fanout_load** values.

To remove **fanout_load** information from ports, use **remove_fanout_load**.

EXAMPLES

This example sets the fanout_load on ports matching "OUT*" to 3.0.

```
pt_shell> set_fanout_load 3.0 "OUT*"
```

SEE ALSO

remove_fanout_load (2), **report_constraint** (2), **report_port** (2), **set_max_fanout** (2), **set_min_fanout** (2).

set_host_options

Specifies host options for compute resources.

SYNTAX

```
int set_host_options
[-local_process]
[-max_cores      number]
[-name           name]
[-submit_command command]
[-terminate_command command]
[-num_processes  number]
[-32bit]
[host_names]

string name
string command
integer number
list   host_names
```

ARGUMENTS

-local_process

Use this option to override the default resource usage limits for the current process. If neither the **-local_process** option nor the **-num_processes** option are specified then **-local_process** is the implied default.

-max_cores *num_cores*

Use the **-max_cores** option to modify the CPU cores usage limit for the current run if **-local_process** is specified, or the subsequently launched remote process otherwise. By default, the cores usage limit would be dictated by the limit indicated by license usage as well as the physical number of cores.

-name *name*

Use the **-name** option to specify the unique name for the host options being defined. Any valid user definable string can be specified; however, if options of the same name already exist, the previously defined options are removed and replaced with the new options. Any remote processes started using the previous options is shutdown. This option cannot be specified in conjunction with the **-local_process** option.

-submit_command *command*

Use the **-submit_command** option to specify the command to call when submitting/launching the PrimeTime processes associated with the host options. This option specifies the path and command to call to perform this operation. When it is called the job/process ID of the job will be captured and associated with the host options. The job and process ID can be reported using the **report_host_usage** command. If the **submit_command** and **host_name** options are not specified or the **host_name** is the same as where the master process is running or is localhost, the processes are launched locally in the operating system of the master. However, if the **host_name** is specified and does not meet the above criterion, rsh is used to make remote connections to

the host when setting up the processes. If a command is specified, it should be non-interactive in nature. This option cannot be specified in conjunction with the **-local_process** option.

-terminate_command *command*

Use the **-terminate_command** option to specify the command to call to terminate the jobs submitted by calling the submit command. When it is necessary to terminate the jobs (e.g. at exit) if the PrimeTime processes have not come online then the master will use the terminate command to terminate the jobs by calling the command and passing the job ID captured from the submit command. This option cannot be specified in conjunction with the **-local_process** option.

-num_processes *num_proc*

Use the **-num_processes** option to specify the number of processes to launch. If neither the **-num_processes** option nor the **-local_process** option are specified then **-local_process** is the implied default.

-32bit

Use this option to specify that the 32-bit binary will be used when launching the processes. By default, the processes is launched using the same architecture as the master. For example, if the master is 32-bit, the processes launched is also 32-bit, even without specifying this option. This option cannot be specified in conjunction with the **-local_process** option.

host_names

Use this option to specify a list of names of hosts on which to launch the slave processes. If nothing is specified then by default a list containing a single entry, "localhost" is assumed. The name "localhost" is the default name for the host on which the current process is running and hence refers to the master process. If host names are specified and if any of the names are not "localhost" or the name of the host on which the master process is running, then the **-submit_command** option can be used to specify the command to setup a remote connection to the hosts in order to launch the PrimeTime slave processes. This option cannot be specified in conjunction with the **-local_process** option.

DESCRIPTION

This command is used to specify the host options of the current process in conjunction with the **-local_process** option.

This command is also used in conjunction with the **-num_processes** option to set up the host options to use when launching PrimeTime processes. The command specifies the options but does not launch the processes. See the **start_hosts** command for more information about launching the processes.

In order to use a custom wrapper for launching slave processes see the **set_distributed_parameters** command.

EXAMPLES

In the following example, the master process is running on the host named ptopt018

using a 64-bit binary and specifies five different sets of host options. The first host options, named "my_opts1", specifies 1 process with the same architecture as the master on the same host as the master. The second host options, named "my_opts2", specifies 2 processes using the 32-bit binary on the same host as the master but explicitly mentions the name of the master's host. The third host options, named "my_opts3", specifies 4 processes (with the same architecture as the master) on the host named ptopt010 using rsh to connect to ptopt010. The fourth host options named "my_opts4", specifies 5 processes (with the same architecture as the master) on an LSF farm requesting 500MB of memory on the compute servers and also specifies the terminate command to use to kill jobs in the farm. The fifth host options, named "my_opts5", specifies 2 processes (with the same architecture as the master) on a Grid farm requiring the jobs to be launched using the project named bnormal and also specifies the terminate command to use to kill jobs in the farm.

Note: Grid requires the "-b y" option.

```
pt_shell> set_host_options -name my_opts1
1
pt_shell> set_host_options -name my_opts2 -32bit -num_processes 2 \
ptopt018
1
pt_shell> set_host_options -name my_opts3 -num_processes 4 \
-submit_command "/usr/bin/rsh -n" ptopt010
1
pt_shell> set_host_options -name my_opts4 -num_processes 5 \
-submit_command "/lsf/bin/bsub -R rusage\[mem=500\]" \
-terminate_command "/lsf/bin/bkill"
1
pt_shell> set_host_options -name my_opts5 -num_processes 2 \
-submit_command "/grid/bin/qsub -b y -P bnormal" \
-terminate_command "/grid/bin/qdel"
1
pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:07:42 2009
*****
```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	Action	Command

my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
	TERMINATE	/lsf/bin/bkill
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal
	TERMINATE	/grid/bin/qdel

1

SEE ALSO

set_distributed_parameters(2), remove_host_options(2), report_host_usage (2),
start_hosts (2), stop_hosts (2)

set_ideal_latency

Specifies ideal latency values for the pins in an ideal network.

SYNTAX

```
int set_ideal_latency
    [-rise] [-fall]
    [-min] [-max]
    value
    object_list
```

```
float value
list object_list
```

ARGUMENTS

-rise

Indicates that the *value* option represents the rise latency time. If you do not specify the **-rise** or **-fall** options, both values are set.

-fall

Indicates that the *value* option represents the fall latency time. If you do not specify the **-rise** or **-fall** option, both values are set.

-min

Indicates that the *value* option represents the minimum latency time. If you do not specify the **-min** or **-max** option, both values are set.

-max

Indicates that the *value* option represents the maximum latency time. If you do not specify the **-min** or **-max** option, both values are set.

value

Specifies an ideal latency value on leaf cell pins or top-level ports in an ideal network.

object_list

Specifies a list of leaf cell pins and top-level ports on which ideal latency is set.

DESCRIPTION

Sets an ideal latency value on leaf cell pins and top-level ports of an ideal network.

PrimeTime uses ideal timing for ideal networks, which means that pins and ports have a specified ideal latency (from the **set_ideal_latency** command) or zero ideal latency by default. Ideal networks are normally used during pre-layout to avoid unnecessary DRCs. The specified ideal latency value provides an estimate of the latency time in the ideal network for pre-layout.

The specified latency value overrides the internally-estimated cell and net delay value and any other delay annotations. If the specified pins do not belong to an ideal network, a warning message is generated by the **report_ideal_network** command and the ideal latency value is ignored for those pins. Ideal latency values do not have any effect in ideal clock networks (for example, non-propagated clock networks).

The **set_ideal_latency** command affects all pins in the transitive fanout of the pins or ports on which ideal latency is specified. The total ideal latency at an ideal boundary pin is the sum of latency on the ideal network source and all ideal latencies on the path.

You can use the **set_ideal_latency** command for pins at lower levels of the design hierarchy.

To list ideal latency values, use the **report_ideal_network** command.

To remove the ideal latency values from a design, use the **remove_ideal_latency** or **reset_design** command.

EXAMPLES

The following example specifies a rise latency of *1.2* and a fall latency of *0.9* for ports named *A*, *B*, and *C*.

```
pt_shell> set_ideal_latency 1.2 -rise {A B C}
pt_shell> set_ideal_latency 0.9 -fall {A B C}
```

SEE ALSO

```
remove_ideal_latency (2),
remove_ideal_network (2),
remove_ideal_transition (2),
report_ideal_network (2),
report_timing (2),
set_ideal_transition (2),
set_ideal_network (2).
```

set_ideal_network

Marks a set of ports or pins in the design as sources of an ideal network. This disables timing update of cells and nets in the transitive fanout of the specified objects.

SYNTAX

```
int set_ideal_network
    [-no_propagate]
    object_list
```

```
list    object_list
```

ARGUMENTS

-no_propagate

Indicates that the ideal network is not propagated through leaf cells. Ideal properties are enabled on all nets that are electrically connected to the ideal network sources.

object_list

Specifies a list of objects to mark as the sources of an ideal network. Ideal network source objects may be ports or pins of leaf cells at any hierarchical level of the design. If nets are specified in the *object_list*, all of the nets' global driver pins are marked as ideal network sources. The **set_ideal_network** command accepts nets only when you specify the **-no_propagate** option.

DESCRIPTION

This command marks a set of ports or pins in the design as sources of an ideal network. You specify only the source of the network. PrimeTime marks all nets, cells, and pins in the transitive fanout of ideal sources as ideal. The ideal property is automatically spread by the tool and respread as necessary after netlist changes. The criteria for propagating the ideal property, starting at the source pins and ports, are as follows:

- A pin is marked as ideal if you specify it by using the **set_ideal_network** command, if it is either a driver pin and its cell is ideal or if it is a load pin attached to a net that is ideal.
- A net is marked as ideal if all its driving pins are ideal.
- A combinational cell is marked as ideal if at least one of its input pins is ideal and all the other input pins are either ideal, unconnected or attached to a logic constant nets. Objects with the case analysis attribute set are not treated as constant.

Propagation traverses through combinational cells but stops at sequential cells.

Design rule checks (DRCs) are disabled on ideal network objects.

Ideal networks can be specified in the clock or data network.

The latency and transition times of an ideal network are zero by default, but you can override them by using the **set_ideal_latency** and **set_ideal_transition** commands. Ideal timing values do not have any effect in ideal clock networks (for example, non-propagated clock networks).

To reverse the effect of the **set_ideal_network** command, use the **remove_ideal_network** command and specify the source pins or ports for the network. All ideal networks are removed using the **reset_design** command.

Ideal sources, non-source ideal pins with ideal timing values, boundary pins of ideal networks and ideal nets and cells are displayed using the **report_ideal_network** command.

EXAMPLES

The following example creates an ideal network on ports in a design called 'TOP'.

```
pt_shell> current_design {TOP}  
pt_shell> set_ideal_network {IN1 IN2}
```

The following example creates an ideal network on the multi-driven net named 'netA'.

```
pt_shell> set_ideal_network -no_propagate {netA}  
Warning: Transferring ideal net attribute onto driver pin 'U1/  
Z' of net 'netA'. (UITE-450)  
Warning: Transferring ideal net attribute onto driver pin 'FFD/  
Q' of net 'netA'. (UITE-450)
```

SEE ALSO

```
remove_ideal_network(2),  
report_ideal_network(2),  
reset_design(2),  
set_ideal_latency(2),  
set_ideal_transition(2).
```

set_ideal_transition

Specifies ideal transition values for the pins in an ideal network.

SYNTAX

```
int set_ideal_transition
    [-rise] [-fall]
    [-min] [-max]
    value
    object_list
```

```
float value
list object_list
```

ARGUMENTS

-rise

Indicates that *value* represents the rise transition time. If you do not specify **-rise** or **-fall**, both values are set.

-fall

Indicates that *value* represents the fall transition time. If you do not specify **-rise** or **-fall**, both values are set.

-min

Indicates that *value* represents the minimum transition time. If you do not specify **-min** or **-max**, both values are set.

-max

Indicates that *value* represents the maximum transition time. If you do not specify **-min** or **-max**, both values are set.

value

Specifies an ideal transition value on leaf cell pins or top-level ports in an ideal network.

object_list

Specifies a list of leaf cell pins and top-level ports on which ideal transition is set.

DESCRIPTION

Sets an ideal transition value on leaf cell pins and top-level ports of an ideal network.

PrimeTime uses ideal timing for ideal networks, which means that pins and ports have a specified ideal transition (from the **set_ideal_transition** command) or zero ideal transition by default. Ideal networks are normally used during pre-layout to avoid unnecessary DRCs. The specified ideal transition value provides an estimate of the transition time in the ideal network for pre-layout.

The specified transition value overrides the internally-estimated cell and net transition value and any other transition annotations. If the specified pins do not belong to an ideal network, a warning message is generated by the **report_ideal_network** command and the ideal transition value is ignored for those pins. Ideal transition values do not have any effect in ideal clock networks (i.e. non-propagated clock networks).

You can use **set_ideal_transition** for pins at lower levels of the design hierarchy.

To list ideal transition values, use the **report_ideal_network** command.

To remove the ideal transition values from a design, use the **remove_ideal_transition** or **reset_design** command.

EXAMPLES

The following example specifies a rise transition of 1.2 and a fall transition of 0.9 for ports named A, B, and C.

```
pt_shell> set_ideal_transition 1.2 -rise {A B C}
pt_shell> set_ideal_transition 0.9 -fall {A B C}
```

SEE ALSO

```
remove_ideal_latency (2),
remove_ideal_network (2),
remove_ideal_transition (2),
report_ideal_network (2),
report_timing (2),
set_ideal_latency (2),
set_ideal_network (2).
```

set_input_delay

Defines the arrival time relative to a clock.

SYNTAX

```
string set_input_delay [-clock clock_name]  
[-reference_pin pin_port_name]  
[-clock_fall]  
[-level_sensitive]  
[-rise]  
[-fall]  
[-max]  
[-min]  
[-add_delay]  
[-network_latency_included]  
[-source_latency_included]  
delay_value  
port_pin_list  
  
list clock_name  
list pin_port_name  
float delay_value  
list port_pin_list
```

ARGUMENTS

-clock *clock_name*

Specifies the clock to which the specified delay is related. If you use **-clock_fall**, you must specify **-clock** *clock_name*. If you do not specify **-clock**, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with a period determined by considering the sequential cells in the transitive fanout of each port.

-reference_pin *pin_port_name*

Specifies the clock pin or port to which the specified delay is related. If you use this option, and if propagated clocking is being used, the delay value is related to the arrival time at the specified reference pin, which is clock source latency plus its network latency from the clock source to this reference pin. The options **-network_latency_included** and **-source_latency_included** cannot be used at the same time as the **-reference_pin** option. For ideal clock network, only source latency is applied. The pin specified with the **-reference_pin** option should be a leaf pin or port in a clock network, in the direct or transitive fanout of a clock source specified with the **-clock** option. If multiple clocks reach the port or pin where you are setting the input delay, and if the **-clock** option is not used, the command considers all of the clocks.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock. When it is used with **-reference_pin** option, the delay is relative to the falling transition of the reference pin. The default is the rising edge or rising

transition of a reference pin.

-level_sensitive

Specifies that the source of the delay is a level-sensitive latch. This allows the tool to derive setup and hold relationship for paths from this port as if it were a level-sensitive latch. If you do not use **-level_sensitive**, the input delay is treated as if it were a path from a flip-flop.

-rise

Specifies that *delay_value* refers to a rising transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-fall

Specifies that *delay_value* refers to a falling transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-max

Specifies that *delay_value* refers to the longest path. If you do not specify **-max** or **-min**, maximum and minimum input delays are assumed to be equal.

-min

Specifies that *delay_value* refers to the shortest path. If you do not specify **-max** or **-min**, maximum and minimum input delays are assumed to be equal.

-add_delay

Specifies whether to add delay information to the existing input delay or to overwrite. Use the **-add_delay** option to capture information about multiple paths leading to an input port that are relative to different clocks or clock edges.

For example, **set_input_delay 5.0 -max -rise -clock phil {A}** removes all other maximum rise input delay from "A", because **-add_delay** is not specified. Other input delays with different clocks or with clock_fall are removed.

In another example, **-add_delay** is specified as **set_input_delay 5.0 -max -rise -clock phil -add_delay {A}**. If there is an input maximum rise delay for "A" relative to clock "phil" rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

-network_latency_included

Specifies whether the clock network latency should not be added to the input delay value. If this option is not specified, the clock network latency of the related clock will be added to the input delay value. It has no effect if the clock is propagated or the input delay is not specified with respect to any clock.

-source_latency_included

Specifies whether the clock source latency should not be added to the input delay value. If this option is not specified, the clock source latency of the related clock will be added to the input delay value. It has no effect if the input delay is not specified with respect to any clock.

`delay_value`

Specifies the path delay. The *delay_value* must be in units consistent with the technology library used during analysis. The *delay_value* represents the amount of time that the signal is available after a clock edge. This usually represents a combinational path delay from the clock pin of a register.

`port_pin_list`

Provides a list of input port or internal pin names in the current design to which *delay_value* is assigned. If you specify more than one object, the objects are enclosed in braces ({}).

DESCRIPTION

Sets input path delay values for the current design. Used with **set_load** and **set_driving_cell**, the input and output delays characterize the operating environment of the current design.

The **set_input_delay** command sets input path delays on input ports relative to a clock edge. Unless specified, input ports are assumed to have zero input delay. For inout (bidirectional) ports, you can specify the path delays for both input and output modes.

To describe a path delay from a level-sensitive latch, use the **-level_sensitive** option. If the latch is positive-enabled, set the input delay relative to the rising clock edge. If the latch is negative enabled, set the input delay relative to the falling clock edge. If time is borrowed at that latch, add that time borrowed to the path delay from the latch when determining input delay.

The **characterize_context** command automatically sets input and output delay, drive, and load values based on the environment of a cell instance.

If for a specific clock the max delay is lesser than min delay, PrimeTime will make max delay equal to that of min delay.

PrimeTime adds input delay to path delay for paths starting at primary inputs, and to output delay for paths ending at primary outputs.

PrimeTime allows an input port to behave simultaneously as a clock and data port. You can use the `timing_simultaneous_clock_data_port_compatibility` variable to enable or disable the simultaneous behavior of the input port as a clock and data port. When this variable is false, the default, simultaneous behavior is enabled and you can use the `set_input_delay` command to define the timing requirements for input ports relative to a clock. In this situation, the following applies:

If you specify the `set_input_delay` command relative to a clock defined at the same port and the port has data sinks, the command is ignored and an error message is issued. There is only one signal coming to port, and it cannot be at the same time data relative to a clock and the clock signal itself.

If you specify the `set_input_delay` command relative to a clock defined at a different port and the port has data sinks, the input delay is set and controls data edges launched from the port relative to the clock.

Regardless of the location of the data port, if the clock port does not fanout to data sinks, the input delay on the clock port is ignored and you receive an error message.

`set_input_delay`

When you set the `timing_simultaneous_clock_data_port_compatibility` variable to true, the simultaneous behavior is disabled and the `set_input_delay` command defines the arrival time relative to a clock. In this situation, when an input port has a clock defined on it, PrimeTime considers the port exclusively as a clock port and imposes restriction on the data edges that are launched. PrimeTime also prevents setting input delays relative to another clock.

To control the clock source latency for any clocks defined on an input port, you must use the `set_clock_latency` command.

If a reference pin is not reachable by any given clock, zero arrival time is assumed. If no clock is specified with a reference pin and this reference pin is not reachable by any clock, an unconstrained path is reported. This behavior is changed after X0506. The new behavior is these invalid constraints will be ignored and path will be constrained by whatever it should be as if no such constraints are defined at all. The **`check_timing`** command generates a warning if the reference pin is not reachable by any active clock, or if multiple clocks reach the reference pin and no clock is specified in the command.

To get a report on the latency calculation using a reference pin, use **`report_timing -path_type full_clock`** or **`report_timing -path_type full_clock_expanded`**.

To list input delays associated with ports, use **`report_port`**. To list input delays of internal pins, use **`report_design`**. To remove input delay values, use **`remove_input_delay`** or **`reset_design`**.

EXAMPLES

The following example sets an input delay of 2.3 for ports "IN1" and "IN2" on a combinational design. Because the design is combinational, no clock is needed.

```
pt_shell> set_input_delay 2.3 { IN1 IN2 }
```

The following example sets input delay of 1.2 relative to the rising edge of "CLK1" for all input ports in the design.

```
pt_shell> set_input_delay 1.2 -clock CLK1 [all_inputs]
```

The following example sets the input and output delays for the bidirectional port "INOUT1". The input signal arrives at "INOUT1" 2.5 units after the falling edge of "CLK1". The output signal is required at "INOUT1" at 1.4 units before the rising edge of "CLK2".

```
pt_shell> set_input_delay 2.5 -clock CLK1 -clock_fall { INOUT1 }  
pt_shell> set_output_delay 1.4 -clock CLK2 { INOUT1 }
```

The following example models the situation where there are three paths to input port "IN1". The first path is relative to the rising edge of "CLK1". The second path is relative to the falling edge of "CLK1". The third path is relative to the falling edge of "CLK2". The **`-add_delay`** option is used to indicate that new input delay information does not cause old information to be removed.

```
pt_shell> set_input_delay 2.2 -max clock CLK1 -add_delay { IN1 }  
pt_shell> set_input_delay 1.7 -max clock CLK1 -clock_fall -add_delay { IN1 }  
pt_shell> set_input_delay 4.3 -max clock CLK2 -clock_fall -add_delay { IN1 }
```

In the following example, two different maximum delays and two minimum delays for port "A" are specified using **-add_delay**. Since the information is relative to the same clock and clock edge, only the largest of the maximum values and the smallest of the minimum values are maintained (in this case, 5.0 and 1.1). If **-add_delay** is not used, the new information overwrites the old information.

```
pt_shell> set_input_delay 3.4 -max -clock CLK1 -add_delay { A }
pt_shell> set_input_delay 5.0 -max -clock CLK1 -add_delay { A }
pt_shell> set_input_delay 1.1 -min -clock CLK1 -add_delay { A }
pt_shell> set_input_delay 1.3 -min -clock CLK1 -add_delay { A }
```

SEE ALSO

all_inputs (2), **characterize_context** (2), **create_clock** (2), **current_design** (2), **remove_input_delay** (2), **report_design** (2), **report_port** (2), **report_timing** (2), **check_timing** (2), **reset_design** (2), **set_driving_cell** (2), **set_load** (2), **set_output_delay** (2).

set_input_noise

Sets a noise bump for a pin or port.

SYNTAX

```
int set_input_noise  
[-add_noise]  
[-above]  
[-below]  
[-low]  
[-high]  
[-height width_value]  
[-width height_value]  
object_list
```

```
float width_value  
float height_value  
list object_list
```

ARGUMENTS

-add_noise
Specifies whether to add noise information to the existing input noise or to overwrite. For example, if `set_input_noise` is used previously, another `set_input_noise` with `-add_noise` option will sum the noise information with value previously set by `set_input_noise`.

-above
Specifies a noise bump for above ground or power rail noise analysis region.

-below
Specifies a noise bump for below ground or power rail noise analysis region.

-low
Specifies a noise bump for ground rail noise.

-high
Specifies a noise bump for power rail noise.

-height *height_value*
Specifies the height of the noise bump. The height is in the voltage units of the library.

-width *width_value*
Specifies the width of the noise bump. The width is in the time units of the library. 1.0.

object_list
Specifies a list of pins or ports.

DESCRIPTION

Rather than allow PrimeTime SI to calculate propagated noise bumps, you can explicitly specify a noise bump at any port or pin. In this command, you specify the port or pin in the design on which to apply the noise and the characteristics of the noise bump: the type (above/below high/low), the width in library time units, the height in library voltage units (always entered as a positive number).

PrimeTime SI treats this injected noise as propagated noise from the driver of the net connected to the pin. This noise bump overrides any propagated noise that would otherwise be calculated from the driver of the net.

You can specify propagated noise on an output pin rather than a load pin. In that case, the specified noise bump overrides any propagated noise that would otherwise be calculated from the driver. The specified noise bump is propagated through the subnets and attenuated by the parasitic capacitance and resistance specified for the net, until the propagated noise reaches each of load pin on the net.

report_noise_calculation will show whether input noise information was calculated or annotated by this command.

EXAMPLES

This example specifies a noise bump height of 0.58, width of 1.70 for pin B of cell U1 in the current design.

```
pt_shell> set_input_noise -above -low -width 1.70 -height 0.58\  
[get_pins U1/B]
```

SEE ALSO

report_noise_calculation (2), **remove_input_noise** (2).

set_input_transition

Sets a fixed transition time on input or inout ports.

SYNTAX

```
string set_input_transition [-rise] [-fall] [-min] [-max] [-clock clock_name] [-  
clock_fall] transition port_list  
float transition  
list port_list
```

ARGUMENTS

-rise

Sets rise transition only.

-fall

Sets fall transition only.

-min

Sets transition for minimum conditions.

-max

Sets transition for maximum conditions.

-clock *clock_name*

The input transition is set relative the specified clock.

-clock_fall

Specifies that the transition is relative to the falling edge of the clock.
The default is the rising edge.

transition

Port transition value. This is a floating point number greater than or equal to zero.

port_list

A list of input or inout ports.

DESCRIPTION

The **set_input_transition** command specifies a fixed transition time for a list of input or inout ports. This transition time is not affected by the capacitance of the net connected to the port. The transition time calculates delays for nets and cells in the transitive fanout of the port. The port itself has no cell delay.

The transition time can be relative to a clock by using -clock option. This means the transition apply only those external paths driven by the clock. Using -clock or -clock_fall option can be meaningful only when *timing_slew_propagation_mode* is in *worst_arrival* mode. Note If it is in *worst_slew* mode the worst of input transitions specified with this command will be used, disregard of the clock(s) specified. If a clock-relative input transition is set on a port, these transitions will not be

accessible via **get_attribute** until **-clock** and **-clock_fall** options are supported for **set_input_transition** in SDC.

To display port transition or drive capability information, use **report_port -drive**.

There are two other methods of describing port drive capability. The **set_driving_cell** command causes the port to have transition time calculated as if a given library cell was driving the net. The **set_driving_cell** command also has a cell delay equal to the load-dependent portion of the delay driving the net of the library cell. The driving cell approach is accurate for nonlinear delay models even if the capacitance is changed. Another method is to use **set_drive**, which models the driver as a linear resistance. The most recent drive command has precedence.

EXAMPLES

This example specifies that ports matching the pattern "DATA_IN*" has a transition time of 0.75 units.

```
pt_shell> set_input_transition 0.75 [get_ports DATA_IN*]
```

SEE ALSO

set_driving_cell(2), **set_drive(2)**, **report_port(2)**, **set_clock_transition(2)**.

set_isolation

Defines the UPF isolation strategy for the power domains in the design.

SYNTAX

```
int set_isolation  
isolation_strategy  
-domain power_domain  
[-isolation_power_net isolation_power_net]  
[-isolation_ground_net isolation_ground_net]  
[-clamp_value 0 | 1 | Z | latch]  
[-applies_to inputs | outputs | both]  
[-elements objects]  
[-no_isolation]
```

Data Types

<i>isolation_strategy</i>	string
<i>power_domain</i>	string
<i>isolation_power_net</i>	string
<i>isolation_ground_net</i>	string
<i>applies_to</i>	string
<i>clamp_value</i>	string
<i>objects</i>	list

ARGUMENTS

isolation_strategy
Specifies the UPF isolation strategy name. The isolation strategy name should be unique within the specified power domain.

-domain *power_domain*
Specifies the power domain name that this UPF isolation strategy will be applied to.

-isolation_power_net *isolation_power_net*
Specifies the isolation power net for the isolation cells that will be created based on this UPF isolation strategy.

-isolation_ground_net *isolation_ground_net*
Specifies the isolation ground net for the isolation cells that will be created based on this UPF isolation strategy. At least one of the -isolation_power net or the -isolation_ground_net should be specified if -no_isolation is not specified

-applies_to *inputs* | *outputs* | *both*
Specifies whether the given set_isolation command applies to all inputs or outputs or both kind of ports of the power domain. The default value for this options is outputs. This option can not be used with -elements option

-elements *objects*
Specifies the objects that this UPF isolation strategy will be applied to.

The objects can be pins of the root cells of the power domain or ports of the top level design for top level power domains

-clamp_value

Specifies the clamp value of the isolation cells that should be created based on this strategy. Default value for this option is 0

-no_isolation

Specifies that elements under the influence of this `set_isolation` command should not be isolated.

DESCRIPTION

This command defines the UPF isolation strategy for the ports of the specified power domain.

If `-elements` and `-applies_to` options are not specified, the isolation strategy is applied to all the outputs ports of the power domain.

PrimeTime uses the **set_isolation** command to build virtual power ground connectivity. The command creates explicit connection for isolation power and ground nets to power and ground pins of isolation cells. If a cell matches multiple isolation rules (**set_isolation** and **set_isolation_control** commands) the last entered **set_isolation** command rule is used to derive PG connectivity of the cell. Use cell attribute `upf_isolation_strategy` to check which isolation rule was applied to the cell.

EXAMPLES

The following example shows how to define a UPF isolation strategy for all the output ports of the specified power domain PD1. Power domain PD1 is defined on instance `shutdown_inst`.

```
pt_shell> set_isolation isolation_1 -domain PD1 \  
          -isolation_power_net PN1 \  
          -isolation_ground_net GN1
```

In the following example, it shows how to define a UPF isolation strategy for specific ports of the specified power domain.

```
pt_shell> set_isolation isolation_2 -domain PD1 \  
          -isolation_power_net PN1 \  
          -isolation_ground_net GN1 \  
          -elements shutdown_inst/special_port
```

To find all isolation cells in the design and the strategy applied to each cell:

```

pt_shell> foreach_in_collection c [sort_collection [get_cells -hier *
-
filter "upf_isolation_strategy != {}"] full_name] { echo [format { Cell %-
8s isolation %s} [get_attribute $c full_name] [get_attribute $c upf_isolation_stra
tegy]]
}

```

SEE ALSO

set_isolation_control.2

set_isolation_control

Provides additional options needed for creating isolation cells.

SYNTAX

```
status set_isolation_control  
isolation_strategy  
-domain power_domain  
-isolation_signal isolation signal  
[-isolation_sense low | high]  
[-location self | parent]
```

Data Types

<i>isolation_strategy</i>	string
<i>power_domain</i>	string
<i>location</i>	string
<i>isolation_sense</i>	string
<i>isolation_signal</i>	string

ARGUMENTS

isolation_strategy
Specifies the UPF isolation strategy name. The isolation strategy should already have been defined in the given *power_domain*

-domain *power_domain*
Specifies the power domain name that this UPF isolation strategy will be applied to.

-isolation_signal *isolation signal*
Specifies the isolation signal (net, pin, port) to used as the control signal of the isolation cells that should be created as a result of this isolation strategy.

-isolation_sense *low / high*
Specifies the isolation sense of the isolation cells that should be created as a result of this isolation strategy. Default value for this option is high

-location *self / parent*
Specifes the hierarchical location of the isolation cells that should be created as a result of this isolation strategy. "self" means that isolation cell will be put in the parent cell of the port being isolated. "parent" value means that isolation cell will be placed in the parent cell of the parent cell of the port being isolated. Default value for this option is self

DESCRIPTION

This command applies to an existing isolation strategy specified by `set_isolation`. It provides extra parameters needed for creating isolation cells.

PrimeTime uses additional information about the isolation strategy provided by this command to properly match isolation strategies to individual leaf cells (PrimeTime does not create isolation cells). Isolation strategy is used for deriving PG connectivity from UPF description.

EXAMPLES

The following example shows how to define a UPF isolation strategy for all the output ports of the specified power domain PD1. Power domain PD1 is defined on instance shutdown_inst. And isolation strategy isolation_1 is already defined

```
prompt> set_isolation_control isolation_1 -domain PD1 \  
        -location parent \  
        -isolation_signal en \  
        -isolation_sense high \  
        -isolation_type output
```

SEE ALSO

set_isolation.2

set_lcd_pulse_width_multipliers

Sets LCD multipliers specific to pulse width check, when using LCD operating conditions.

SYNTAX

```
int set_lcd_pulse_width_multipliers  
[-mult_worst op_worst_mult]  
[-mult_nominal op_nom_mult]  
[-mult_best op_best_mult]  
[-library lib_name]  
[name]
```

Data Types

<i>op_worst_mult</i>	float
<i>op_nom_mult</i>	float
<i>op_best_mult</i>	float
<i>lib_name</i>	string
<i>name</i>	string

ARGUMENTS

-mult_worst *op_worst_mult*
Specifies the LCD multiplier to apply to worst case operating condition in pulse width check.

-mult_nominal *op_nom_mult*
Specifies the LCD multiplier to apply to nominal operating condition in pulse width check.

-mult_best *op_best_mult*
Specifies the LCD multiplier to apply to best case operating condition in pulse width check.

-library *lib_name*
Specifies the name of the library for the LCD operating condition.

name
Specifies the name for the LCD operating condition.

DESCRIPTION

When using Linear Combination of Delays (LCD) operating conditions, this command sets LCD multipliers specific to pulse width check. By default, LCD multipliers for pulse width check have the values set by the **create_lcd_operating_condition** command. LCD multipliers specific to pulse width checks are needed to mimic process variations within the same cell for the opposite edge.

To see the operating conditions and the LCD multipliers defined for a library, use the **report_lib** command.

```
set_lcd_pulse_width_multipliers  
1000
```


EXAMPLES

The following example creates a LCD operating condition named `lcd_late` and sets the LCD multipliers for pulse width check for LCD operating condition `lcd_late`.

```
pt_shell> create_lcd_operating_condition -op_worst wccom -op_nominal nocom -  
op_best bccom  
-mult_worst 0.7 -mult_nominal 0.3 -mult_best 0.1 -library library_name lcd_late  
  
pt_shell> set_lcd_pulse_width_multipliers -mult_worst 0.6 -mult_nominal 0.2  
-mult_best 0.1 -library library_name lcd_late
```

SEE ALSO

```
create_lcd_operating_condition(2)  
report_lib(2)
```

set_level_shifter_strategy

Sets the type of strategy to use for reporting the signal level mismatches in the design.

SYNTAX

```
int set_level_shifter_strategy  
-rule all | low_to_high | high_to_low
```

ARGUMENTS

```
-rule all | low_to_high | high_to_low
```

Specifies types of source-sink pairs of mismatching signal levels that **check_timing -include signal_level** reports.

Using **-rule low_to_high** reports the voltage level mismatches when a source at a lower voltage drives a sink at a higher voltage.

Using **-rule high_to_low** reports the voltage level mismatches when a source at a higher voltage drives a sink at a lower voltage.

The **all** value is the default strategy.

DESCRIPTION

This command specifies the strategy for reporting nets that need insertion of level shifters.

EXAMPLES

The following example sets the level shifter strategy to the **high_to_low** option:

```
prompt> set_level_shifter_strategy -rule high_to_low
```

SEE ALSO

```
check_timing (2),  
set_level_shifter_threshold (2).
```

set_level_shifter_threshold

Sets the minimum threshold beyond which the voltage adjustment is required.

SYNTAX

```
int set_level_shifter_threshold  
-voltage volt  
-percent diff
```

Data Types for All Modes

```
volt      float  
diff      float
```

ARGUMENTS

```
-voltage volt  
    The absolute difference between the source and sink voltages.  
  
-percent diff  
    The percentage by which the source and sink voltages must differ. The  
    percentage is determined as follows:  
  
     $\text{abs}(\text{driver}(v) - \text{load}(v)) / \text{driver}(v) * 100$ 
```

DESCRIPTION

This command specifies the minimum threshold value for the voltage difference between a source and a sink. Voltage differences above the minimum threshold are reported by **check_timing -include signal_level**. The threshold can be specified as the absolute difference in voltages or a percentage difference or both.

When both **-voltage** and the **-percent** options are both specified: Since physical tools insert level shifters when either absolute or relative threshold is exceeded the **check_timing -include signal_level** only skips signal level mismatches that are smaller than both the absolute and the relative thresholds.

The default threshold voltage and percentage is zero. Therefore you typically must specify both absolute and relative thresholds to suppress reporting of below-threshold level mismatches.

EXAMPLES

The following example sets the level shifter threshold value to the absolute difference of 0.1 (and sets large relative threshold):

```
psyn_shell-t> set_level_shifter_threshold -voltage 0.1 -percent 100
```

The following example sets the threshold value 5 percent (and sets large absolute

threshold):

```
prompt> set_level_shifter_threshold -percent 5 -voltage 10
```

The following example sets the threshold value to 0.1 difference and 5 percent, so if either value is reached, the signal level mismatch is reported:

```
prompt> set_level_shifter_threshold -voltage 0.1 -percent 5
```

SEE ALSO

```
check_timing (2),  
set_level_shifter_strategy (2).
```

set_lib_rail_connection

Sets a physical power pin on a library cell.

SYNTAX

```
int set_lib_rail_connection  
-lib_cells lib_cells  
-lib_rail_name name of power rail (NOT SUPPORTED)  
-lib_pin name of library pin
```

Data Types

<i>lib_cells</i>	string
<i>name of library pin</i>	string

ARGUMENTS

```
-lib_cell_names lib_cells  
    Specifies a list of library cells on power_rails are to be recognized.  
  
-lib_rail_names names of power rail  
    This argument is accepted, but not currently supported in PrimeTime.  
  
-lib_pin_names names of library pin  
    Specifies the names of library pin to be be annotated.
```

DESCRIPTION

The **set_lib_rail_connection** command associates the name of a power pin with a lib cell. When linking physical Verilog to a logical library, pins specified by this command may be present in the Verilog, but not in the logical library.

EXAMPLES

The following example shows how to set a physical power pin on a library cell to link with to logical library.

```
pt_shell>set_lib_rail_connection -lib_cell_names typ/INVX1 -lib_pin VDD1
```

When subsequently linking physical Verilog, pins named VDD1 on the *INVX1* library cell are ignored when linking to the (logical) library *typ*. This allows successful linking of physical Verilog to logical libraries in which power pins not specified.

set_library_driver_waveform

Sets the driver waveform used to characterize the timing library.

SYNTAX

```
int set_library_driver_waveform  
-type type  
[lib_objs]
```

list *lib_objs*

ARGUMENTS

-type "*type*"

The type of the waveforms. It has two possible values, "ramp" : linear waveform and "standard": waveform commonly known as Synopsys pre-driver.

lib_objs

List of libraries or library cells on which the driver waveform applies.

DESCRIPTION

The data tables in the timing libraries are indexed by slew. The library characterization process could use the standard Synopsys predriver waveform or a simple ramp waveform to characterize the library. The library data reflects the type of waveform used for characterization. The shape of the waveform generally has only a small effect on the timing results. However, for some forms of advanced analysis, the waveform shape can become significant:

- o For advanced crosstalk delay analysis using a CCS Noise library, using the exact waveform shape in PrimeTime SI is necessary to get the best possible accuracy.

- o When write_spice_deck writes out a SPICE deck for a given path or arc, each input to the cell should use the same waveform that was used to characterize the library, to get the best possible correlation between PrimeTime SI and SPICE simulation results.

This command specifies the waveform shape for the library objects. This command allows two type of waveforms "ramp" and "standard" (Synopsys pre-driver waveform). If the library object is not provided the waveform is applied to the whole design. When waveform specified on a library cell it gets higher priority than the waveform set on library. And the waveform set on library gets higher priority than the design. This command triggers update_timing.

EXAMPLES

This example specifies how to set ramp for the whole design. When write_spice_deck generates the spice deck it uses ramp for voltage sources if no other waveform is specified on the library or library cell.

```
pt_shell> set_library_driver_waveform -type ramp
```

```
set_library_driver_waveform  
1006
```

In the following example the "libOld" is characterized with ramp and the lib "libNew" is characterized with standard waveforms. It shows how to set these waveforms after the libraries read to PrimeTime. The lib_pin attribute **driver_waveform_fall/driver_waveform_rise** could be used to get the name of the characterization waveforms.

```
pt_shell> set_library_driver_waveform -type ramp \  
                                                [get_lib libOld]  
pt_shell> set_library_driver_waveform -type standard \  
                                                [get_lib libNew]  
pt_shell> get_attr [get_lib_pin libOld/INV/A] driver_waveform_fall  
pt_shell> get_attr [get_lib_pin libOld/INV/A] driver_waveform_rise
```

SEE ALSO

write_spice_deck (2), **pba_enable_ccs_waveform_propagation** (3),
si_ccs_use_gate_level_simulation (3).

set_load

Sets the capacitance to a specified value on the specified ports and nets in the current design.

SYNTAX

```
Boolean set_load
[-min]
[-max]
[-rise]
[-fall]
[-subtract_pin_load]
[-pin_load]
[-wire_load]
capacitance
objects

capacitance    float
objects        list
```

ARGUMENTS

-min

Indicates that the *capacitance* is the minimum capacitance. Applies only to designs in min-max mode (minimum and maximum operating conditions).

-max

Indicates that the *capacitance* is the maximum capacitance. Applies only to designs in min-max mode (minimum and maximum operating conditions).

-rise

Indicates that the *capacitance* is the rise capacitance. This option can be used in combination with **-min** or **-max**. Use this option only with ports. An error message is generated if the *objects* list contains nets.

-fall

Indicates that the *capacitance* is the fall capacitance. This option can be used in combination with **-min** or **-max**. Use this option only with ports. An error message is generated if the *objects* list contains nets.

-subtract_pin_load

Indicates that the current pin capacitances of the specified net are to be subtracted from *capacitance* before the net capacitance value is set. Any resulting negative net capacitance values are set to zero. With this option, the total capacitance computed on these nets during **update_timing** does not include pin capacitances. Use this option only if *capacitance* includes pin capacitances.

-pin_load

Indicates that the specified *capacitance* is a pin capacitance. Pin capacitance is not subject to "scaling" using *tree_type*. Use this option only with ports. An error message is generated if the *objects* list contains nets.

If you do not specify either the **-pin_load** or the **-wire_load** option, or both, **-pin_load** is the default.

-wire_load

Indicates that the specified *capacitance* is a wire capacitance. Pin capacitance is subject to "scaling" using *tree_type*. Use this option only with ports. An error message is generated if the *objects* list contains nets. If you do not specify either the **-pin_load** or the **-wire_load** option, or both, **-pin_load** is the default.

capacitance

Specifies the capacitance value to be set on the ports and nets in *objects*. It is in the units of the technology library.

objects

Specifies a list of ports and nets in the current design, on which the *capacitance* is to be set. Note that if you specify a name (for example, *net_name*) instead of a real object_list (for example, by using the **get_port** command with the *port_name* option or the **get_net** command with the *net_name* option), the tool first searches the list of all ports and then searches the list of all nets for a match.

DESCRIPTION

This command sets the capacitance to a specified value on specified ports and nets in the current design. If the current design is hierarchical, you must link it using the **link** command. Depending on the options used, **set_load** sets these attributes on the specified objects:

wire_capacitance_max
wire_capacitance_min
pin_capacitance_max
pin_capacitance_min

If this command is issued without any options, the **pin_capacitance_max** attribute is set; in min-max mode, the **pin_capacitance_min** attribute is also set.

For ports, if **set_load** is issued with only the **-wire_load** option, then the **wire_capacitance_max** attribute is set on the specified ports; in min-max mode, the **wire_capacitance_min** attribute is also set. However, the value is actually counted as part of the total wire capacitance and not as part of the pin/port capacitance.

In min-max mode, using either the **-min** or **-max** option sets only the ***_min** or ***_max** attributes, respectively.

If **-rise** is specified, the **pin_capacitance_max_rise** and **pin_capacitance_min_rise** attributes are set. You can use this option in conjunction with **-min** or **-max** to set only one of the two attributes. The same conditions apply for the **-fall** option and the **pin_capacitance*_fall** attributes.

The total capacitance on a net is the sum of all pin capacitances, port capacitances, and wire capacitances associated with that net. The specified *capacitance* overrides both the internally-estimated net capacitance value and any net capacitance set by the **read_parasitics** command.

You can also use the **set_load** command for nets at lower levels of the design hierarchy. These nets are specified in the "BLOCK1/BLOCK2/NET_NAME" format. The tool treats capacitances in such a way that timing on a subblock should be the same as timing on the higher-level design, since pin/wire caps on ports represent a part of the higher-level design.

To view capacitance values on ports and nets, use the **report_port** and **report_net** commands, respectively. To remove a capacitance value, use the **remove_capacitance** command; you can remove annotated capacitances from the full design by using the **reset_design** command.

EXAMPLES

The following example sets a capacitance of 2 units on the port named *the_answer*.

```
pt_shell> set_load 2 the_answer
```

The following example sets a capacitance of 2.5 units (the estimated wire capacitance) plus the capacitance of three inverter input pins from the library named *tech_lib* on all output ports. It uses the user-defined **pt_shell** variable **port_capacitance** to store this capacitance value.

```
pt-shell> set pin_cap \  
?[get_attribute [get_lib_pins tech_lib/IV/A] pin_capacitance]  
2.0  
pt_shell> set port_capacitance [expr 2.5 + (3 * $pin_cap)]  
8.5  
pt_shell> set_load $port_capacitance [all_outputs]  
1
```

The following example uses the **get_attribute** command to get the value of the **pin_capacitance** attribute on pin *Z* of *IV* from the *tech_lib* library and sets that value on the *input_1* and *input_2* ports.

```
pt_shell> set_load \  
?[get_attribute [get_lib_pins tech_lib/IV/Z] pin_capacitance] \  
?[get_ports {input_1 input_2}]
```

The following example sets a capacitance of 3 on the **U1/U2/NET3** net. The wire capacitance is set to 3. (Total net capacitance is 3 + the sum of the pin and port capacitances.)

```
pt_shell> set_load 3 U1/U2/NET3
```

The following example sets a total net capacitance (wire capacitance + pin capacitances) of 3 on the *U1/U2/NET3* net. If the pin and port capacitances equal 2, the wire capacitance is annotated with 1; if the pin and port capacitances are 3 or more, the wire capacitance is annotated with 0.

```
pt_shell> set_load -subtract_pin_load 3 U1/U2/NET3
```

The following example sets a wire capacitance of 5 units on the port named *the_answer*.

```
pt_shell> set_load -wire_load 5 the_answer
```

The following example removes the back-annotated capacitance on a port and on a net.

```
pt_shell> remove_capacitance [get_ports the_answer]
```

```
pt_shell> remove_capacitance [get_nets net12]
```

SEE ALSO

```
all_outputs(2)  
current_design(2)  
remove_capacitance(2)  
report_net(2)  
report_port(2)  
reset_design(2)  
set_drive(2)  
set_wire_load(2)
```

set_max_area

Sets the **max_area** attribute on the current design to a specified value.

SYNTAX

```
int set_max_area area_value
float area_value
```

ARGUMENTS

area_value
Specifies the value to which the **max_area** attribute is to be set. The value must be ≥ 0 . The units of *area_value* must be consistent with the units in the technology library used during optimization.

DESCRIPTION

The **set_max_area** command sets the **max_area** attribute on the current design to the specified *area_value*. This attribute represents the target area of the design.

Use **remove_max_area** or **reset_design** to remove the **max_area** attribute on the current design.

Use **report_constraint** to show area cost of the design.

EXAMPLES

The following example sets the target area for the current design to 250.

```
pt_shell> set_max_area 250.0
1
pt_shell> get_attribute [current_design] max_area
250.000000
```

SEE ALSO

current_design (2), **get_attribute** (2), **remove_max_area** (2), **report_constraint** (2), **reset_design** (2).

set_max_capacitance

Sets maximum capacitance for pins, ports, clocks or designs.

SYNTAX

```
string set_max_capacitance capacitance_value  
[-clock_path] [-data_path]  
[-rise] [-fall]  
object_list
```

```
float capacitance_value  
list object_list
```

ARGUMENTS

-clock_path
Specifies all pins that are in the network of the specific clocks in the *object_list*.

-data_path
Specifies all pins that are in the data paths launched by the specific clocks in the *object_list*.

-rise
Specifies rising capacitance for all selected pins.

-fall
Specifies falling capacitance for all selected pins.

capacitance_value
Capacitance limit (Value >= 0). This is the maximum total capacitance (pin plus wire capacitance) in library capacitance units.

object_list
Provides a list of pins, ports, clocks or designs on which to set maximum capacitance.

DESCRIPTION

Specifies a maximum capacitance on pins, ports or design. If maximum capacitance is set on a pin or port, the net connected to that pin or port is expected to have a total capacitance less than the specified *capacitance_value*. If specified on a design, the default maximum capacitance for that design is set. Library cell pins also can have a **max_capacitance** value specified.

If maximum capacitance is set on a clock, the maximum capacitance is applied to all pins in this specified clock domain. Within a clock domain, you can optionally restrict the constraint further to clock paths only or data paths only, and to rising or falling capacitance only. Note that *clock_path*, *data_path*, *rise* and *fall* options can only be used when the max capacitance limit is set on a clock and does not apply to design, pin or port.

The most restrictive of the design limit, pin, clock, library or port limit is used.

The **report_constraint -max_capacitance** command shows maximum capacitance constraint evaluations. The **report_port -design_rule** command shows port maximum capacitance limits. The **report_design** command shows the default maximum capacitance setting for the current design.

To remove maximum capacitance limits from designs or ports, use **remove_max_capacitance**.

EXAMPLES

The following example sets a maximum capacitance limit of 2.0 units on ports "OUT*".

```
pt_shell> set_max_capacitance 2.0 [get_ports "OUT*"]
```

The following example sets the default maximum capacitance limit of 5.0 units on the current design.

```
pt_shell> set_max_capacitance 5.0 [current_design]
```

The following example sets a maximum capacitance limit of 0.8 units on all pins of the clock network of CLK.

```
pt_shell> set_max_capacitance 0.8 [get_clocks CLK] -clock_path
```

The following example sets a maximum capacitance limit of 0.7 units on all pins of the data path of CLK.

```
pt_shell> set_max_capacitance 0.7 [get_clocks CLK] -data_path
```

SEE ALSO

current_design(2), remove_max_capacitance(2), report_constraint(2),
report_design(2), report_port(2), get_ports(2), set_min_capacitance(2).

set_max_delay

Specifies a maximum delay for timing paths.

SYNTAX

```
Boolean set_max_delay
[-rise] [-fall]
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
[-reset_path]
delay_value

list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
float delay_value
```

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise or fall setup or hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_max_delay**.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of

a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through *through_list*

Specifies a list of pins, ports, cells, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the leaf-level driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

-rise_through *rise_through_list*

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects.

-fall_through *fall_through_list*

This option is similar to the **-through** option, but applies only to paths with a falling transition at the specified objects.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a

sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

delay_value

Specifies a floating point number that represents the required maximum delay value for specified paths. *delay_value* must have the same units as the technology library used during analysis. If a path startpoint is on a sequential device, clock skew is included in the computed delay. If a path startpoint has an input delay specified, that delay value is added to the path delay. If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay. If the endpoint has an output delay specified, that delay is added into the path delay.

DESCRIPTION

This command specifies a required maximum delay for timing paths in the current design. The path length for any startpoint in *from_list* to any endpoint in *to_list* must be less than *delay_value*.

The value of a **max_rise_delay** attribute cannot be less than that of a **min_rise_delay** attribute on the same path (and similarly for fall attributes). If this condition occurs, the older attribute is removed.

Individual maximum delay targets are automatically derived from clock waveforms and port input or output delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_max_delay** command is a point-to-point timing exception command. For example, the command overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_multicycle_path**, **set_min_delay**, and **set_false_path**. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path. For example, either **-from** or **-to** is used (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following list of commands is arranged in order, from the highest to the lowest precedence (more specific to more general).

1. `set_max_delay -from pin -to pin`
2. `set_max_delay -from clock -to pin`
3. `set_max_delay -from pin -to clock`
4. `set_max_delay -from pin`
5. `set_max_delay -to pin`
6. `set_max_delay -from clock -to clock`
7. `set_max_delay -from clock`
8. `set_max_delay -to clock`

To list the **max_delay**, **min_delay**, **multicycle_path**, and **false_path** information for the design, use the **report_exceptions** command. To list the defined path groups, use **report_path_group**.

To remove information set by **set_max_delay**, use the **reset_path** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port "Y" must be less than 10.0 units.

```
pt_shell> set_max_delay 10.0 -to {Y}
```

The following command specifies that all paths from ff1a or ff1b to ff2e must have delays less than 15.0 units.

```
pt_shell> set_max_delay 15.0 -from {ff1a ff1b} -to {ff2e}
```

The following example specifies that all paths to endpoints clocked by PHI2 must have delays less than 8.5 units.

```
pt_shell> set_max_delay 8.5 -to [get_clocks PHI2]
```

The following example sets a requirement that all paths leading to ports named "busA[*]" must have delays less than 5.0.

```
pt_shell> set_max_delay 5.0 -to "busA[*]"
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
pt_shell> set_max_delay 8.0 -from ff1/CP -through {U1/Z U2/Z} -through {U3/Z U4/C} -to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
pt_shell> set_max_delay 8.0 -from ff1/CP -rise_through {U1/Z U2/Z} -fall_through
```

{U3/Z U4/C} -to ff2/D

SEE ALSO

**create_clock (2), current_design (2), group_path (2), report_constraint (2),
report_path_group (2), reset_design (2), reset_path (2), set_false_path (2),
set_input_delay (2), set_min_delay (2), set_multicycle_path (2), set_output_delay
(2).**

set_max_fanout

Sets maximum fanout for input ports or designs.

SYNTAX

```
string set_max_fanout fanout_value  
object_list
```

```
float fanout_value  
list object_list
```

ARGUMENTS

fanout_value
Fanout limit (Value >= 0). This is the maximum fanout load in library fanout units.

object_list
Provides a list of input ports or designs on which to set maximum fanout.

DESCRIPTION

Specifies a maximum fanout on input ports or designs. If maximum fanout is set on a port, the net connected to that port is expected to have **fanout_load** less than the specified *fanout_value*. Fanout load for a net is the sum of **fanout_load** attributes for all input pins on the net. If specified on a design, the default maximum fanout for that design is set. Library cell pins may also have a **max_fanout** value specified. The most restrictive of the design limit and the pin or port limit will be used.

The **report_constraint -max_fanout** command shows maximum fanout constraint evaluations. The **report_port -design_rule** shows port maximum fanout limits. The **report_design** command shows the default maximum fanout setting for the current design.

To remove maximum fanout limits from designs or ports, use **remove_max_fanout**.

EXAMPLES

The following example sets a maximum fanout limit of 2.0 units on ports "IN*".

```
pt_shell> set_max_fanout 2.0 [ge_ports "IN*"]
```

The following example sets the default maximum fanout limit of 5.0 units on the current design.

```
pt_shell> set_max_fanout 5.0 [current_design]
```

SEE ALSO

`current_design` (2), `remove_max_fanout` (2), `report_constraint` (2), `report_design` (2),
`report_port` (2), `get_ports` (2), `set_fanout_load` (2), `set_min_fanout` (2).

set_max_time_borrow

Limits time borrowing for latches.

SYNTAX

```
string set_max_time_borrow value  
object_list
```

```
float value  
list object_list
```

ARGUMENTS

value

Specifies the value to which the **max_time_borrow** attribute is set. Defines the desired limit of time borrowing on the latches specified in **object_list**. *delay_value* must be between zero and the default maximum derived from the waveform. By default, the maximum is derived from the ideal clock waveform driving each latch, and is equal to (closing_edge - open_edge). Library setup and data-to-Q propagation times are automatically taken into account. *delay_value* is in the same units as those in the technology library used during analysis.

object_list

Specifies a list of objects in the **current_design** for which time borrowing is to be limited to *value*. The objects can be clocks, latch cells, data pins, or clock (enable) pins. If a cell is specified, all enable pins on that cell are affected.

DESCRIPTION

Sets the **max_time_borrow** attribute to **value** to constrain the amount of time borrowing possible for level-sensitive latches. To meet delay targets, **set_max_time_borrow** prevents automatic use of all or part of the enabling clock pulse on a latch.

If **max_time_borrow** is set on both data and clock (enable) pins of a level-sensitive latch, the value set on the data pin takes higher priority.

To get **max_time_borrow** attributes on the design, use **get_attribute** or **report_exceptions**.

To undo **set_max_time_borrow**, use **remove_max_time_borrow**.

EXAMPLES

The following example restricts time borrowing on latches **latch1a** and **latch2f** to 4.0 units.

```
pt_shell> set_max_time_borrow 4.0 { latch1a latch2f }
```

The following example specifies that no time borrowing will take place on **latch1c**.

```
pt_shell> set_max_time_borrow 0.0 { latch1c }
```

SEE ALSO

current_design (2), **remove_max_time_borrow** (2), **get_attribute** (2), **report_exceptions** (2).

set_max_transition

Sets maximum transition for pins, ports, clocks or designs with respect to the main library trip-points.

SYNTAX

```
string set_max_transition transition_value  
[-clock_path] [-data_path]  
[-rise] [-fall]  
object_list
```

```
float transition_value  
list object_list
```

ARGUMENTS

-clock_path
Specifies all pins that are in the network of the specific clocks in the *object_list*.

-data_path
Specifies all pins that are in the data paths launched by the specific clocks in the *object_list*.

-rise
Specifies rising transition for all selected pins.

-fall
Specifies falling transition for all selected pins.

transition_value
Sets the transition limit (Value >= 0). This is the maximum transition time in library time units.

object_list
Provides a list of pins, ports, clocks or designs on which to set maximum transition.

DESCRIPTION

Specifies a maximum transition on pins, ports or designs. If maximum transition is set on a pin or port, the pin or port is expected to have transition time less than the specified *transition_value*. If specified on a design, the default maximum transition for that design is set. Library cell pins can also have a **max_transition** value specified. The most restrictive of the design limit and the pin or port limit is used. If user entered limit is the most restrictive limit, it is scaled to that of the pin trip-points during slack computation.

If maximum transition is set on a clock, the maximum transition is applied to all pins in this specified clock domain. Within a clock domain, you can optionally restrict the constraint further to clock paths only or data paths only, and to

rising transitions only or falling transitions only.

The `-clock_path`, `-data_path`, `-rise`, and `-fall` options can be used only if the list of objects is a clock list, not a pin or port list or design. If `clock_list` is specified without `-clock_path`, `-data_path`, `-rise` or `-fall`, both clock path and data path, both rise and fall are considered by default.

The **`report_constraint -max_transition`** command shows maximum transition constraint evaluations. The actual limit used for slack computation is reported. If user entered limit is used for slack computation, the reported limit will be different for pins with different trip-points. The minimum of the slack for rise and fall transition is reported. When there are violations of both design/pin/port limit and clock limit in different clock domains, the worst slack is reported.

The **`report_port -design_rule`** shows port maximum transition limits. The **`report_design`** command shows the default maximum transition setting for the current design.

To remove maximum transition limits from designs or ports, use **`remove_max_transition`**.

EXAMPLES

The following example sets a maximum transition limit of 2.0 units on ports "OUT*".

```
pt_shell> set_max_transition 2.0 [get_ports "OUT*"]
```

The following example sets the default maximum transition limit of 5.0 units on the current design.

```
pt_shell> set_max_transition 5.0 [current_design]
```

The following example sets a maximum transition limit of 2.0 units on all pins in the clock network of CLK.

```
pt_shell> set_max_transition 2.0 [get_clocks CLK] -clock_path
```

SEE ALSO

`current_design` (2), **`remove_max_transition`** (2), **`report_constraint`** (2), **`report_design`** (2), **`report_port`** (2), **`get_ports`** (2),

set_min_capacitance

Sets minimum capacitance for ports or designs.

SYNTAX

```
string set_min_capacitance capacitance_value  
object_list
```

```
float capacitance_value  
list object_list
```

ARGUMENTS

capacitance_value

Sets capacitance limit (Value ≥ 0). This is the minimum total capacitance (pin plus wire capacitance) in library capacitance units.

object_list

Provides a list of input or inout ports or designs on which to set minimum capacitance.

DESCRIPTION

Specifies a minimum capacitance on input/inout ports or designs. If minimum capacitance is set on a port, the net connected to that port is expected to have total capacitance greater than the specified *capacitance_value*. If specified on a design, the default minimum capacitance for that design is set. Library cell pins can also have a **min_capacitance** value specified. The most restrictive of the design limit and the pin or port limit is used.

The **report_constraint -min_capacitance** command shows minimum capacitance constraint evaluations. The **report_port -design_rule** shows port minimum capacitance limits. The **report_design** command shows the default minimum capacitance setting for the current design.

To remove minimum capacitance limits from designs or ports, use **remove_min_capacitance**.

EXAMPLES

The following example sets a minimum capacitance limit of 0.2 units on ports "IN*".

```
pt_shell> set_min_capacitance 0.2 [get_ports "IN*"]
```

The following example sets the default minimum capacitance limit of 0.1 units on the current design.

```
pt_shell> set_min_capacitance 0.1 [current_design]
```

SEE ALSO

`current_design` (2), `remove_min_capacitance` (2), `report_constraint` (2), `report_design` (2), `report_port` (2), `get_ports` (2), `set_min_capacitance` (2).

set_min_delay

Specifies a minimum delay for timing paths.

SYNTAX

Boolean **set_min_delay**

```
[-rise] [-fall]
[-reset_path]
[-from from_list
  | -rise_from rise_from_list
  | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
  | -rise_to rise_to_list
  | -fall_to fall_to_list]
delay_value
```

```
list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
float delay_value
```

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither **-rise** nor **-fall** is specified, both rising and falling delays are constrained.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise or fall setup or hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_min_delay**.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of

set_min_delay

a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through *through_list*

Specifies a list of pins, ports, cells, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the leaf-level driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

-rise_through *rise_through_list*

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects.

-fall_through *fall_through_list*

This option is similar to the **-through** option, but applies only to paths with a falling transition at the specified objects.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a

sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

delay_value

Specifies a floating point number that represents the required minimum delay value for specified paths. *delay_value* must have the same units as the technology library used during analysis. If a path startpoint is on a sequential device, clock skew is included in the computed delay. If a path startpoint has an input external delay specified, that delay value is added to the path delay. If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay. If the endpoint has an output external delay specified, that delay is added into the path delay.

DESCRIPTION

This command specifies a required minimum delay for timing paths in the current design. The path length for any startpoint in *from_list* to any endpoint in *to_list* must be greater *delay_value*.

Use **set_min_delay** in the following two cases:

1. To set a target minimum delay for output ports in a combinational design.
2. To override the default single cycle timing for paths, where **set_multicycle_path** is not sufficient.

The value of a **min_rise_delay** attribute cannot be greater than that of a **max_rise_delay** attribute for the same path (and similarly for fall attributes). If this condition occurs, the older attribute is removed.

Individual minimum delay targets are automatically derived from clock waveforms and port input or output delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_min_delay** command is a point-to-point timing exception command; it overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_multicycle_path**, **set_max_delay**,

and **set_false_path**. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path; either **-from** or **-to** is used (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following list of commands is arranged in order, from highest to lowest precedence (more specific to more general):

1. **set_min_delay** -from *pin* -to *pin*
2. **set_min_delay** -from *clock* -to *pin*
3. **set_min_delay** -from *pin* -to *clock*
4. **set_min_delay** -from *pin*
5. **set_min_delay** -to *pin*
6. **set_min_delay** -from *clock* -to *clock*
7. **set_min_delay** -from *clock*
8. **set_min_delay** -to *clock*

To remove information set by **set_min_delay**, use the **reset_path** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port "Y" must be greater than 12.5 units.

```
pt_shell> set_min_delay 12.5 -to Y
```

The following command specifies that all paths from A1 and A2 to Z5 must have delays greater than 4.0 units.

```
pt_shell> set_min_delay 4.0 -from {A1 A2} -to Z5
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays greater than 3.0 units.

```
pt_shell> set_min_delay 3.0 -from ff1/CP -through {U1/Z U2/Z} -through {U3/Z U4/C} -  
to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays greater than 3.0 units.

```
pt_shell> set_min_delay 3.0 -from ff1/CP -rise_through {U1/Z U2/Z} -fall_through  
{U3/Z U4/C} -to ff2/D
```

SEE ALSO

`current_design` (2), `report_constraint` (2), `reset_design` (2), `set_input_delay` (2),
`set_output_delay` (2), `reset_path` (2), `set_false_path` (2), `set_multicycle_path` (2),
`set_max_delay` (2).

set_min_library

Sets the library to be used for minimum delay analysis

The **set_min_library** command is used to relate a minimum conditions library to a maximum conditions library.

SYNTAX

```
string set_min_library  
[-min_version min_library]  
[-none]  
max_library
```

```
stringmin_library  
stringmax_library
```

ARGUMENTS

-min_version *min_library*
The library for min analysis. This library is not to be used in the **link_path**.

-none
Dissociate *max_library* from its min library

max_library
The library for max analysis. This library should be used in the **link_path**.

DESCRIPTION

The **set_min_library** command creates a max/min relationship between two libraries. Once the relationship is established, the *max_library* is used for maximum delay analysis and the *min_library* is used for minimum delay analysis.

For each library cell in the *max_library*, the **set_min_library** command searches for a library cell of the same name in the *min_library*. If it is not found, a warning is issued, and there will be no max/min relationship for that library cell. If the *min_library* does have the named library cell, the command will compare the two library cells to verify that they have the same pins (in the same order, with the same directions), and the same timing arcs. If any of these conditions fails, a warning is issued, and there will be no max/min relationship for that library cell.

At least one library cell must match for the command to succeed.

When PrimeTime needs to compute a minimum delay value, if there is a max/min relationship for the library cell, the timing information from the min library is used. Otherwise, the information is taken from the max library.

Both libraries and objects in them can be referenced in commands such as **set_operating_conditions**, **set_wire_load_model**, and so on.

Libraries used as the *min_library* in a **set_min_library** command should never be

placed in the link path. Only the *max_library*, which is the root of the relationship, is used in the link path. A warning is issued if you use a min library in the link path.

A library in use as a min library can be removed with **remove_lib** as long as no environment information (operating conditions, wire load models, and so on) is being used from that library. When you remove a library in this way, any designs using the max/min library will be scheduled for a full timing update.

The **list_libraries** command will show max/min relationships. In addition, the **report_lib** command will indicate when a max library has a relationship to a min library. See the examples or the man pages for these commands.

EXAMPLES

The following is a typical usage of **set_min_library**.

```
pt_shell> set_min_library LIB_WC_COM.db -min_version LIB_BC_COM.db
Loading db file '/u/libraries/LIB_WC_COM.db'
Loading db file '/u/libraries/LIB_BC_COM.db'
Created max/min library relationship:
  Max: /u/libraries/LIB_WC_COM.db:LIB_WC_COM
  Max: /u/libraries/LIB_BC_COM.db:LIB_BC_COM
1
pt_shell> list_libraries
Library Registry:
  m LIB_BC_COM      /u/libraries/LIB_BC_COM.db:LIB_BC_COM
  *M LIB_WC_COM     /u/libraries/LIB_WC_COM.db:LIB_WC_COM
1
pt_shell> set_operating_conditions -max WC_COM -max_library LIB_WC_COM \
    -min BC_COM -min_library LIB_BC_COM
1
pt_shell> report_timing -delay min
```

SEE ALSO

list_libraries (2), **remove_lib** (2), **report_lib** (2), **set_operating_conditions** (2), **set_wire_load_model** (2), **link_path** (3).

set_min_pulse_width

Sets a minimum pulse width constraint for specified design objects.

SYNTAX

```
string set_min_pulse_width [-low] [-high] value [object_list]  
float value  
list object_list
```

ARGUMENTS

-low

Indicates that the minimum pulse width constraint specified by *value* is to apply only to low clock signal levels. If you do not specify the **-low** or **-high** option, both low and high pulses of clock signals are constrained.

-high

Indicates that the minimum pulse width constraint specified by *value* is to apply only to high clock signal levels. If you do not specify the **-low** or **-high** option, both low and high pulses of clock signals are constrained.

value

A positive floating point value that specifies the minimum pulse width check to be applied to clock signals.

object_list

Specifies a list of clocks, cells, pins or ports in the current design, to which the minimum pulse width check is to be applied. If you specify a cell or clock, all pins on the cell or clock are affected. If you do not specify any objects, the minimum pulse width check is applied to the current design.

DESCRIPTION

The **set_min_pulse_width** command specifies a minimum pulse width check to be applied to clock signals in the clock tree or at sequential devices. This value overrides the default values for clock tree minimum pulse width checks. Use this command for sequential clock pin pulse width checks, to add a more restrictive value than the one specified in library.

A clock pulse width problem occurs if the negation of the clock signal happens too soon after the assertion of the clock signal. Clock pulse width violations can cause the following problems:

- Sequential devices (flip-flops and latches) might not capture data properly.
- In some logic circuits, the entire pulse could disappear.

Two types of minimum pulse width checks are performed:

- Clock Pulse Width Check at Sequential Devices: This check is performed on clock

pins of sequential elements. This check verifies that a minimum separation exists between the trailing edge and leading edge of the clock that is clocking the sequential elements. The library defines the constraints, which are environmentally scalable. The **set_min_pulse_width** command overrides the library value. The most restrictive value of pulse width (library-specified or user-asserted) is used. No default value is assumed.

- **Clock Tree Pulse Width Check at Logic Circuits:** The clock tree pulse width check ensures that the clock pulse is propagated reliably through the logic. This check is performed on the combinational logic circuits through which the clock signals are propagated. No default value is assumed for the clock tree pulse width check. The **set_min_pulse_width** command overrides the library value.

Note that if the variable **timing_enable_pulse_clock_constraints** is set to **TRUE** (default value) then the constraint set by this command donot apply to pulse clock networks. To apply the constraints to pulse clock networks use command **set_pulse_clock_min_width**. To constrain pulse clock network using **set_min_pulse_width** command from UI, set the variable **timing_enable_pulse_clock_constraints** to **FALSE**.

To generate a report of pulse width constraints, use **report_constraint -min_pulse_width** or **report_min_pulse_width**. To remove minimum pulse width constraints set by **set_min_pulse_width**, use the **remove_min_pulse_width** command.

The **reset_design** command removes all user-specified attributes from a design, including those set by **set_min_pulse_width**.

EXAMPLES

The following example sets a minimum pulse width requirement of 2.0 for both low and high pulses of clock signals.

```
pt_shell> set_min_pulse_width 2.0 [get_clocks CK1]
```

The following example sets a minimum pulse width requirement of 2.5 for the low pulse of the clock signal on the pin U1/Z.

```
pt_shell> set_min_pulse_width -low 2.5 U1/Z
```

SEE ALSO

current_design (2), **remove_min_pulse_width** (2), **report_constraint** (2), **report_min_pulse_width** (2), **reset_design** (2), **set_pulse_clock_min_width** (2), **report_pulse_clock_min_width** (2), **timing_enable_pulse_clock_constraints** (2).

set_mode

Selects the active mode of cell mode groups or design mode groups

SYNTAX

```
Boolean set_mode [-type cell | design]
[mode_list]
[instance_list]
list mode_list
list instance_list
```

ARGUMENTS

-type design | cell

Indicates the type of mode to be made active. This option has the following mutually-exclusive valid values: **design** and **cell**. The **cell** value specifies that cell modes are to be made active. Cell modes are defined on library cells in the library. The **design** value specifies that design modes are to be made active. Design modes are user specified using **define_design_mode_group** and **map_design_mode**. If the **-type** option is omitted from the command then this is equivalent to specifying **-type cell**

mode_list

Specifies a list of modes, each of which is to be made the active mode for its mode group. If **-type** has a cell value then the **mode_list** must contain only cell modes. If **-type** has a design value then the mode list must contain only design modes.

instance_list

Specifies a list of instances for which the specified cell modes are to be made active. This list must only be included with **-type cell**.

DESCRIPTION

Selects the active mode for a mode group or for several mode groups and disables modes in the same group as the selected active mode. Mode groups must either have all modes enabled (default setting) or have one of their modes enabled and all others disabled. This command sets either cell modes or design modes depending on the value of the type option.

Cell mode groups are defined in the library. Each library cell can have a set of cell mode groups. Each of these cell mode groups can have two or more cell modes. Each of these cell modes can be mapped to a set of timing arcs of the library cell. When a cell mode is made active for a given instance of the library cell, the cell mode is enabled and all of its timing arcs are enabled for that cell. All other cell modes are automatically disabled. The timing arcs of the disabled cell modes are then automatically disabled. To view what modes have been enabled or disabled use **report_mode -type cell**. To view what arcs have been disabled, use **report_disable_timing**.

In the special case of an arc having multiple cell modes mapped to it, the arc will

be enabled if any of the modes are enabled.

Cell modes can be made active in three different ways, using **set_mode -type cell**, using **set_mode -type design** or through evaluation of mode conditions. When conflict arises the following precedence rule is employed **set_mode -type cell** > **set_mode -type design** > evaluation of mode conditions

Design modes and design mode groups are defined by the user with the **define_design_group** and **map_design_mode** commands. Design modes can be mapped to a set of cell modes and/or a set of paths. When a design mode of a particular design mode group is made active all cell modes mapped to the design mode are made active and all paths mapped to the design mode are reset. All other design modes in the design mode group are disabled. All paths associated with these disabled design modes will be set to false path. No action is taken to any cell modes mapped to the disabled design modes.

EXAMPLES

In the following example, the first command defines two design modes, DM1 and DM2. (Since no mode group name was specified, the mode group is named "default".) The second command specifies that for design mode DM1, the READ cell mode is active for the RAM instance Uram1. The third command specifies that for design mode DM2, the WRITE component mode is active for the RAM instance Uram1. The fourth command maps all paths ending at Uram1/D0 to the design mode DM2.

```
pt_shell> define_design_mode_group {DM1 DM2}
pt_shell> map_design_mode DM1 READ Uram1
pt_shell> map_design_mode DM2 WRITE Uram1
pt_shell> map_design_mode DM2 -to Uram1/D0
```

The following example selects the design mode DM1 as the active mode for the current design, which in turn causes the cell mode READ to be selected as the active mode for the instance Uram1. Since design mode DM2 is automatically disabled all paths ending at Uram1/D0 become false paths.

```
pt_shell> set_mode -type design DM1
```

The following command directly selects the READ cell mode as the active mode for the RAM instance Uram1.

```
pt_shell> set_mode READ Uram1
```

The following command is equivalent to the previous command

```
pt_shell> set_mode -type cell READ Uram1
```

SEE ALSO

define_design_mode_group (2), **remove_design_mode** (2), **reset_mode** (2), **report_mode**

(2), `map_design_mode` (2), `timing_through_compatibiity` (3).

set_multi_scenario_license_limit

Specifies the upper limit on the number of licenses, of a particular feature, the master in multi-scenario analysis can checkout for dynamic allocation among its slave processes.

SYNTAX

```
int set_multi_scenario_license_limit  
-feature feature_name  
[-force_checkout]  
numberfp
```

```
string feature_name  
int    number
```

ARGUMENTS

-feature

Select the feature to be dynamically managed for multi-scenario analysis. See the help -v output of the command for the currently supported features.

number

Specifies the maximum number of licenses to be checked out for slave usage.

-force_checkout

Forces licenses to be checked out straight away. If this option is not specified licenses will only be checked out when needed.

DESCRIPTION

This command is only available when the user invokes PrimeTime with the -multi_scenario option.

In multi-scenario analysis, it is necessary to specify up-front, using the **set_multi_scenario_license_limit** command, the maximum number of PrimeTime/PrimeTime-SI etc. licenses for slave usage during the analysis. Licenses can be checked out up front using the -force_checkout option or by default licenses will only be checked out on an as-needed basis. Once the master checks out the licenses, it does not release them until it exits, it dynamically allocates them among its slave processes on an as-needed basis. Hence, more slave processes can be started than there are licenses checked out. However, at any point in time, the number of executing slave processes will never exceed the number of licenses checked out by the master for slave usage.

If the -force_checkout option is issued the master tries to checkout immediately up to the limit specified for the particular feature. It may get less than this but will never attempt to get more.

If licenses are being checked out on an as-needed basis and a request for a license checkout succeeds, a message is printed to the master shell and the analysis

continues.

If SI analysis is to be performed at the slave and the license limit for the PrimeTime-SI feature has not been set then SI analysis will not be performed at the slave. The same applies for other features e.g. VA, PX etc.

When the master checks out N licenses of a feature this implies that N slave processes can operate concurrently as the master does not require a license itself during the analysis.

EXAMPLES

In the following example, the upper limit on the number of PrimeTime licenses is set to seven and the number of PrimeTime-SI licenses set to four. No licenses will be checked out at the point at which the commands are called, the licenses will be checked out as and when the slaves need them during the subsequent analysis.

```
pt_shell> set_multi_scenario_license_limit -feature PrimeTime 7
1
pt_shell> set_multi_scenario_license_limit -feature PrimeTime-SI 4
1
```

SEE ALSO

report_multi_scenario_design (2),

set_multicycle_path

Defines the multicycle path.

SYNTAX

Boolean **set_multicycle_path**

```
[-setup] [-hold]
[-rise] [-fall]
[-start] [-end]
[-reset_path]
[-from from_list
    | -rise_from rise_from_list
    | -fall_from fall_from_list]
[-through through_list]*
[-rise_through rise_through_list]*
[-fall_through fall_through_list]*
[-to to_list
    | -rise_to rise_to_list
    | -fall_to fall_to_list]
path_multiplier
```

```
list from_list
list rise_from_list
list fall_from_list
list through_list
list rise_through_list
list fall_through_list
list to_list
list rise_to_list
list fall_to_list
int path_multiplier
```

ARGUMENTS

-setup

Indicates that setup (maximum delay) calculations are to use the specified *path_multiplier*. Note that changing the *path_multiplier* for setup affects the default hold check as well. If neither **-setup** nor **-hold** is specified, *path_multiplier* is used for setup calculations and 0 is used for hold calculations.

-hold

Indicates that hold (minimum delay) calculations are to use the specified *path_multiplier*. Note that changing the *path_multiplier* for setup affects the default hold check as well. If neither **-setup** nor **-hold** is specified, *path_multiplier* is used for setup calculations and 0 is used for hold calculations.

-rise

Indicates that only rising path delays are to use *path_multiplier*. If neither **-rise** nor **-fall** is specified, both rising and falling delays are affected. Rise refers to a rising value at the path endpoint.

set_multicycle_path

-fall

Indicates that only falling path delays are to use *path_multiplier*. If neither **-rise** nor **-fall** is specified, both rising and falling delays are affected. Fall refers to a falling value at the path endpoint.

-start

Indicates that the multicycle information is relative to the period of the start clock. The **-start** and **-end** options are needed only for multifrequency designs; otherwise, start and end are equivalent. The start clock is the clock source related to the register or primary input at the path startpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with **-start** moves the relation backward one cycle of the start clock. A hold multiplier of 1 with **-start** moves the relation forward one cycle of the start clock.

-end

Indicates that the multicycle information is relative to the period of the end clock. The **-start** and **-end** options are needed only for multifrequency designs; otherwise, start and end are equivalent. The end clock is the clock source related to the register or primary output at the path endpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with **-end** moves the relation forward one cycle of the end clock. A hold multiplier of 1 with **-end** moves the relation backward one cycle of the end clock.

-reset_path

Indicates that existing point-to-point exception information is to be removed from the specified paths. If used with **-to** only, all paths leading to the specified endpoints are reset. If used with **-from** only, all paths leading from the specified startpoints are reset. If used with **-from** and **-to**, only paths between those points are reset. Only information of the same rise/fall setup/hold type is reset. Using this option is equivalent to using the **reset_path** command with similar arguments before issuing **set_multicycle_path**.

-from from_list

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. If a cell is specified, one path startpoint on that cell is affected. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-rise_from rise_from_list

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-fall_from fall_from_list

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of

the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of **-from**, **-rise_from**, and **-fall_from**.

-through *through_list*

Specifies a list of pins, ports, cells and nets through which the multicycle paths must pass. Nets are interpreted to imply the leaf-level driver pins. If you omit **-through**, all timing paths specified using the **-from** and **-to** options are affected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

-rise_through *rise_through_list*

This option is similar to the **-through** option, but applies only to paths with a rising transition at the specified objects.

-fall_through *fall_through_list*

This option is similar to the **-through** option, but applies only to paths with a fall transition at the specified objects.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, one path endpoint on that cell is affected. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock

path. You can use only one of **-to**, **-rise_to**, and **-fall_to**.

path_multiplier

An integer that specifies the number of cycles the data path must have for setup or hold, relative to the startpoint or endpoint clock, before data is required at the endpoint. For example, specifying a *path_multiplier* of 2 for setup implies a 2-cycle data path. If you use **-setup**, *path_multiplier* is applied to setup path calculations. If you use **-hold**, *path_multiplier* is applied to hold path calculations. If you do not specify either **-setup** or **-hold**, *path_multiplier* is used for setup and 0 is used for hold. Note that changing the multiplier for setup affects the hold check as well.

DESCRIPTION

This command specifies the number of cycles that the data path must have for setup or hold, so that that designated timing paths in the current design have no default setup or hold relations.

PrimeTime applies certain rules to determine single cycle timing relationships for paths between clocked elements; the rules are based on active edges. For flip-flops, a single active edge both launches and captures data. For latches, the open edge launches data and the close edge latches data.

The setup check ensures that the correct data signal is available on destination registers in time to be correctly latched. The default setup behavior, called single-cycle setup, is as follows:

For multifrequency designs, there can be multiple setup relations between two clocks. For every latch edge of the destination clock, the setup check determines the nearest launch edge that precedes each latch edge. The smallest value of (setup_latch_edge - setup_launch_edge) determines the maximum delay requirement for this path.

You can override this default relationship by applying **set_multicycle_path** or **set_max_delay** to clocks, pins, ports, or cells. For example, setting the setup path multiplier to 2 with the **set_multicycle_path** command delays the latch edge one clock pulse. Note that changing the setup multiplier also affects the default hold check.

Most often, the setup check moves relative to the end clock. This move changes the data latch time at the path endpoint. In multi-frequency designs, use **set_multicycle_path -setup -start** to move the data launch time backward. The hold check ensures that data from the source clock edge that follows the setup launch edge is not latched by the setup latch edge, and also ensures that data from the setup launch edge is not latched by the destination clock edge that precedes the setup latch edge.

The hold check is determined relative to each valid setup relationship, after applying multicycle path multipliers. The most restrictive (largest) difference of (hold_latch_edge - hold_launch_edge) is used as a minimum delay requirement for the path.

Important Note: The hold relation is conservative. In some cases you need to override the default hold relation. For multifrequency designs, it is more straightforward to use the **set_min_delay** command to override the default instead of

using **set_multicycle_path -hold**.

There are separate setup and hold multipliers for rise and fall. In most cases, these are set to the same value. You can use the **-rise** or **-fall** option to apply different values.

The **set_multicycle_path** command is a point-to-point timing exception command. The command can override the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include **set_max_delay**, **set_min_delay**, and **set_false_path**. False path information always takes precedence over multicycle path information. A specific **set_max_delay** or **set_min_delay** command overrides a general **set_multicycle_path** command.

The more general commands apply to more than one path; either **-from** or **-to** (but not both), or clocks are used in the specification. Within a given point-to-point exception command, the more specific command overrides the more general. The following lists commands from highest to lowest precedence (more specific to more general):

1. **set_multicycle_path -from *pin* -to *pin***
2. **set_multicycle_path -from *pin* -to *clock***
3. **set_multicycle_path -from *pin***
4. **set_multicycle_path -from *clock* -to *pin***
5. **set_multicycle_path -to *pin***
6. **set_multicycle_path -from *clock* -to *clock***
7. **set_multicycle_path -from *clock***
8. **set_multicycle_path -to *clock***

To undo a **set_multicycle_path** command, use the **reset_path** command. The **reset_design** command removes all attributes from the design, including those set by **set_multicycle_path**.

To disable setup and hold calculations for paths, use **set_false_path**. To list the point-to-point exceptions on a design, use **report_exceptions**.

EXAMPLES

The following example sets all paths between latch1b and latch2d to 2-cycle paths for setup. Hold is measured at the previous edge of the clock at latch2d.

```
pt_shell> set_multicycle_path 2 -from { latch1b } -to { latch2d }
```

The following example moves the hold check to the preceding edge of the start clock.

```
pt_shell> set_multicycle_path -1 -from { latch1b } -to { latch2d }
```

The following example is a two-phase level-sensitive design, where the designer expects paths from phil to phil to be zero cycles.

```
pt_shell> set_multicycle_path 0 -from phil -to phil
```

The following example uses **-start** to specify a 3-cycle path relative to the clock at the path startpoint. Clock sources are specified, affecting all sequential elements clocked by that clock, or ports with input or output delay relative to that clock.

```
pt_shell> set_multicycle_path 3 -start -from { clk50mhz } -to { clk10mhz }
```

The following example sets all rising paths to ff12/D to 2 cycles, and falling paths to 1 cycle.

```
pt_shell> set_multicycle_path 2 -rise -to { ff12/D }
```

```
pt_shell> set_multicycle_path 1 -fall -to { ff12/D }
```

The following multifrequency example shows a 20ns clock to 10ns clock with multicycle path of 2. Assume a path from ff1 (clocked by CLK2) to ff2 (clocked by CLK1).

```
pt_shell> create_clock -period 20 -waveform {0 10} CLK2
pt_shell> create_clock -period 10 -waveform {0 5} CLK1
pt_shell> set_multicycle_path 2 -setup -from ff1/CP -to ff2/D
```

The single-cycle setup relation is from the CLK1 edge at 0ns to the CLK2 edge at 10ns. With the multicycle path, the setup relation is now 20ns (from CLK1 edge at 0ns to CLK2 edge at 20ns). The hold relations are determined according to that setup relation.

1. Data from the source clock edge that follows the setup launch edge must not be latched by the setup latch edge. This implies a hold relation of 0ns (from CLK1 edge at 20ns to CLK2 edge at 20ns).

2. Data from the setup launch edge must not be latched by the destination clock edge that precedes the setup latch edge. This implies a hold relation of 10ns (from CLK1 edge at 0ns to CLK2 edge at 10ns).

The most restrictive (largest) hold relation is used, so the minimum delay requirement for this path is 10ns. This is a conservative check which might not apply to certain designs. Often you know that the destination register is disabled for the clock edge at 10ns. You can modify this default hold relation with the following to get an 0ns requirement.

```
pt_shell> set_multicycle_path 1 -hold -end -from ff1/CP -to ff2/D
```

However, a simpler approach is to use the following:

```
pt_shell> set_min_delay 0 -from ff1/CP -to ff2/D
```

The following example sets all timing paths from ff1/CP to ff2/D which passes through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} to 2 cycle paths for setup.

```
pt_shell> set_multicycle_path 2 -from ff1/CP -through {U1/Z U2/Z} -through
{U3/Z U4/C} -to ff2/D
```

SEE ALSO

`current_design` (2), `report_exceptions` (2), `reset_design` (2), `reset_path` (2),
`set_false_path` (2), `set_max_delay` (2), `set_min_delay` (2).

set_noise_derate

Sets noise derate information for the current design.

SYNTAX

```
int set_noise_derate  
[-above]  
[-below]  
[-low]  
[-high]  
[-height_offset hoffset]  
[-height_factor hfactor]  
[-width_factor wfactor]  
[object_list]  
  
float hoffset  
float hfactor  
float wfactor  
list object_list
```

ARGUMENTS

- above
Specifies a derate setting for above ground or power rails noise analysis region.
- below
Specifies a derate setting for below ground or power rails noise analysis region.
- low
Specifies a derate setting for ground rail noise.
- high
Specifies a derate setting for power rail noise.
- height_offset *hoffset*
Specifies an offset voltage in ratio of Vdd, that will be directly added to each aggressor and total noise height of a pin in the design. The value of this offset must be between -1.0 to 1.0. Default value for this offset is 0.0.
- height_factor *hfactor*
Specifies a scale factor that could increase or decrease the calculated height of the noise waveform of each aggressor and the total noise height of a pin in the design. The value for this scale factor must be larger than 0.0. The default for this scale factor is 1.0.
- width_factor *soffset*
Specifies a scale factor that could increase or decrease the calculated width of the noise waveform of each aggressor and the total noise width of a pin in the design. The value for this scale factor must be larger than 0.0. The default for this scale factor is 1.0.

`object_list`
Specifies a list of input pins or output ports.

DESCRIPTION

If the *object_list* is not specified, sets the noise derate offset and scale factors for the current design. If the *object_list* is specified, sets the noise derate offset and scale factors for the specified pins and ports. Noise derating offset and scale factors affect the noise height and width values shown in noise reports. Also the noise propagated through stages will be impacted by these settings. If offset is not specified for a noise analysis region, 0.0 offset is assumed. If scale factors are not specified for a noise analysis region, 1.0 scale factors are assumed. To show derating offsets and scale factors settings, use **report_noise_calculation**.

EXAMPLES

This example specifies a voltage offset of 0.10 and noise height and width scale factors of 0.90 for above the ground rail noise:

```
pt_shell> set_noise_derate -above -low -height_offset 0.10 \  
-height_factor 0.90 -width_factor 0.90
```

To restore the settings to their original values:

```
pt_shell> set_noise_derate -above -low -height_offset 0.0 \  
-height_factor 1.0 -width_factor 1.0
```

SEE ALSO

report_noise (2), **report_noise_calculation** (2), **set_noise_parameters** (2).

set_noise_immunity_curve

Sets noise immunity curve for a library pin or port.

SYNTAX

```
int set_noise_immunity_curve  
[-above]  
[-below]  
[-low]  
[-high]  
[-height height_value]  
[-width width_value]  
[-area area_value]  
object_list
```

```
float height_value  
float width_value  
float area_value  
list object_list
```

ARGUMENTS

-above
Specifies an immunity curve for above ground or power rail noise analysis region.

-below
Specifies an immunity curve for below ground or power rail noise analysis region.

-low
Specifies an immunity curve for ground rail noise.

-high
Specifies an immunity curve for power rail noise.

-height *height_value*
Specifies the height of the input voltage bump at the threshold of logic failure. The height is in the voltage units of the library.

-width *width_value*
Specifies the width of the input voltage bump at the threshold of logic failure. The width is in the time units of the library.

-area *area_value*
Specifies an area value for the curve. This parameter specifies the size of the hyperbolic curve. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

Each cell input can tolerate a certain amount of noise without causing a failure at the cell output. This characteristic is called noise immunity. This command specifies noise immunity at library pins to determine whether noise failures occur as well as the amount of noise slack.

Noise immunity in the library can be specified as noise immunity curves, polynomials, or tables or in terms of noise margins that consider only the bump heights at the cell inputs.

This command specifies a noise immunity curve for a library pin or design port. It can be used in the absence of library-specified noise immunity characteristics, or to override the library-specified characteristics by replacing them with a noise immunity curve. This command will also override noise margins that have been annotated using the `set_noise_margin` command,

The curve is hyperbolic with three coefficients to fully specify the hyperbola: height and width of the input voltage bump at the threshold of logic failure, and an area value to specify the size of the curve. The three coefficients should be chosen to match the hyperbolic curve to the data points obtained by laboratory characterization.

Noise immunity characteristics can vary for different noise bump types, so there can be four different noise immunity curves associated with each input: below low, above low, below high, and above high. All coefficients are specified as positive numbers for all four types of noise bumps.

report_noise_calculation will show whether noise immunity information was taken from the library or annotated by this command.

EXAMPLES

This example specifies a noise height of 2, width of 3, area of 4 for above the ground rail noise for pin A of library cell IV in the `lsi_10k` library:

```
pt_shell> set_noise_immunity_curve -above -height 2 -width 3 \
-area 4 lsi_10k/IV/A
```

SEE ALSO

set_noise_margin (2), **report_noise_calculation** (2), **remove_noise_immunity_curve** (2).

set_noise_lib_pin

Sets an equivalent noise library pin for a driver or load.

SYNTAX

```
int set_noise_lib_pin  
pins  
lib_pin
```

```
list pins  
list lib_pin
```

ARGUMENTS

pins
Specifies collection of pins for which the noise library pin is set.

lib_pin
Specifies the equivalent library pin from which the noise information will be used.

DESCRIPTION

This command allows to redefine the noise information for a certain library pin. For output pins, setting an equivalent noise library pin means that the I/V curves of the equivalent noise library pin will be used. For input pins, noise immunity information of the equivalent noise library pin is used.

EXAMPLES

This example specifies an equivalence for noise information on two inputs pins of the design to be the same as an input library pin:

```
pt_shell> set_noise_lib_pin [get_pins {cell/pin1 cell/pin2}] lsi_10k/IV/A\
```

SEE ALSO

update_noise (2), **report_noise** (2), **report_noise_calculation** (2).

set_noise_margin

Sets noise margin for a library pin, port, or pin.

SYNTAX

```
int set_noise_margin  
[-above]  
[-below]  
[-low]  
[-high]  
margin_value  
object_list
```

```
float margin_value  
list object_list
```

ARGUMENTS

-above

Specifies the noise margin for above ground or power rail noise analysis region.

-below

Specifies the noise margin for below ground or power rail noise analysis region.

-low

Specifies the noise margin for ground rail noise.

-high

Specifies the noise margin for power rail noise.

margin_value

Specifies a margin value. The value is the input height in units of voltage. 1.0.

object_list

Specifies a list of lib-pins or ports.

DESCRIPTION

Each library cell input can tolerate a certain amount of noise without causing a failure at the cell output. This characteristic is called noise immunity. This command specifies noise immunity at library pins to determine whether noise failures occur as well as the amount of noise slack.

Noise immunity in the library can be specified as noise immunity curves, polynomials, or tables or in terms of noise margins.

This command specifies a noise margin for a library pin, design port, or pin. It can be used in the absence of library-specified noise immunity characteristics, or to

override the library-specified characteristics by replacing them with noise margins. Noise immunity curves that have been annotated using the `set_noise_immunity_curve` command will override noise margins set by this command.

Noise margins consider only the bump heights at the cell inputs. Using height-only noise margins is simpler, faster, and more conservative than the other methods.

For high, narrow noise bumps, using height-only noise margins is pessimistic because it treats some bumps as noise failures that would otherwise pass with the immunity curve model. However, for wide noise bumps, using noise margins gives the same results as using noise immunity curves.

Noise immunity characteristics can vary for different noise bump types, so there can be four different noise margins associated with each input: below low, above low, below high, and above high. All values are specified as positive numbers for all four types of noise bumps.

In the absence of command-specified or library-specified noise immunity data, PrimeTime SI calculates the maximum allowable noise bump heights based on DC noise margins of the driver and receiver, as defined in the `.lib` technology library by the input/output logic-level parameters.

report_noise_calculation will show whether noise margin information was taken from the library or annotated by this command.

EXAMPLES

This example specifies a margin of 4 for above the ground rail noise for pin A of library cell IV in the `lsi_10k` library:

```
pt_shell> set_noise_margin -above -low 4 lsi_10k/IV/A
```

SEE ALSO

set_noise_immunity_curve (2), **report_noise_calculation** (2), **remove_noise_margin** (2).

set_noise_parameters

Defines the noise analysis parameters for the current design.

SYNTAX

```
int set_noise_parameters  
[-ignore_arrival]  
[-include_beyond_rails]  
[-analysis_effort low | high]  
[-enable_propagation]  
[-analysis_mode report_at_source | report_at_endpoint]
```

ARGUMENTS

-ignore_arrival

Setting this option causes the arrival window information of the aggressors to be ignored during the noise analysis, and therefore the aggressors are assumed to be always overlapping to maximize the effect of coupled noise bump.

-include_beyond_rails

By default, the analysis of noise above the high rail and below the low rail is disabled. Using **-include_beyond_rails**, enables the analysis of noise beyond the high and low regions.

-analysis_effort low | high

Specifies the the effort level mode of the noise analysis. The valid values for this option are **low**, and **high**, and the default value is **high**. If the **low** effort mode is selected, a fast estimation noise calculation is performed for the entire design. If the **high** effort mode is selected, then the noise bumps with bump height greater than a certain threshold, are recalculated using detailed noise calculation engine.

-enable_propagation

Specifies whether to allow noise propagation or not. Propagated noise on a victim net is caused by noise at an input of the cell that is driving the victim net. PrimeTime SI can calculate propagated noise at a cell output, given the propagation characteristics of the cell, the noise bump at the cell input, and the load on the cell output.

-analysis_mode report_at_source | report_at_endpoint

Specifies the analysis mode. In report_at_source mode, violations are reported at the source of violations. In report_at_endpoint mode, violations are propagated through fanout and reported at endpoints. The default value is report_at_source.

DESCRIPTION

This command specifies the parameters that are considered during the noise analysis. If this command is not performed or reset command **reset_noise_parameters** is performed, the default settings are assumed for the noise analysis: beyond the rail

analysis is ignored, arrival times of aggressors are considered, analysis effort level is high, and propagation of noise is disabled.

This command has no effect if no option is specified.

Report command **report_noise_parameters** can be used to report the status of the noise analysis parameters for the current design.

Reset command **reset_noise_parameters** can be used to reset the noise analysis parameters to the default settings.

EXAMPLES

The following example has no effect.

```
pt_shell> set_noise_parameters
```

In the following example, the propagation is enabled, while other parameters are not affected.

```
pt_shell> set_noise_parameters -enable_propagation
```

SEE ALSO

update_noise (2), **report_noise** (2), **report_noise_parameters** (2),
reset_noise_parameters (2), **report_noise_violation_sources** (2),
get_noise_violation_sources (2).

set_operating_conditions

Defines the operating conditions (or environmental characteristics) for the *current design*.

SYNTAX

```
int set_operating_conditions  
[-analysis_type single | bc_wc | on_chip_variation]  
[-library lib]  
[condition]  
[-min min_condition]  
[-max max_condition]  
[-min_library min_lib]  
[-max_library max_lib]  
[-object_list objects]
```

```
string lib  
string condition  
string min_condition  
string max_condition  
string min_lib  
string max_lib  
list objects
```

ARGUMENTS

-analysis_type *single | bc_wc | on_chip_variation*

Indicates how the specified operating conditions are to be used. This option has the following mutually-exclusive valid values: **single**, **bc_wc**, and **on_chip_variation**. The **single** value specifies that only one operating condition is to be used. Specifying either **bc_wc** or **on_chip_variation** switches the design to min_max mode. The **bc_wc** value specifies that the min and max operating conditions are two extreme operating conditions. In the **bc_wc** analysis, setup violations are checked only for the maximum operating condition, and the hold violations are checked only for the minimum operating condition. The **on_chip_variation** value specifies that the minimum and maximum operating conditions represent, respectively, the lower and upper bounds of the maximum variation of operating conditions on the chip. All maximum path delays use the maximum operating condition, and all minimum path delays use the minimum operating condition.

-library *lib*

Specifies the library that contains definitions of the environmental characteristics to be used. This is either a library name or a collection object.

condition

Specifies the name of the single operating condition.

-min *min_condition*

Specifies the minimum operating condition used for checking minimum delays

set_operating_conditions

and hold violations (see the **report_timing** command with the **-delay min** option). If you use this option you must also use the **-max** option.

-max *max_condition*

Specifies the maximum operating condition used for checking maximum delays and setup violations (see the **report_timing** command with the **-delay max** option). If you use this option you must also use the **-min** option.

-min_library *min_lib*

Specifies the library that contains the *min_condition*. This is either a library name or a collection object. If you use this option you must also use the **-min** option.

-max_library *max_lib*

Specifies the library that contains the *max_condition*. This is either a library name or a collection object. If you use this option you must also use the **-max** option.

-object_list *objects*

Specifies the cells or ports on which to set operating conditions. If you do not use this option, operating conditions are set on the design. This option accepts both leaf cells and hierarchical blocks. You cannot use this option with the **-analysis_type** option. This option is only supported in legacy flows. Command **set_voltage** should be used instead.

DESCRIPTION

Defines the operating conditions (or environmental characteristics) under which the *current design* is timed. You are in *min_max* mode if you use the **-min** and **-max** options. In that case, the default analysis type is **bc_wc**.

The specified operating conditions must be defined in one of the following: in *min_lib* (for the *min_condition* only), in *max_lib* (for the *max_condition* only), or in one of the libraries in the *link_path*. The order for a library search is as follows:

1. *lib*
2. *min_lib* or *max_lib*
3. *link_path*

If no operating conditions are specified for a design, the tool uses the default operating condition of the library to which the cell is linked. If the library does not have default operating conditions, no operating conditions are used.

Operating conditions set by using the **-object_list** option override the operating conditions set on design or higher levels of hierarchy.

To view the operating conditions defined for the *current design* and to determine the libraries to which the *current design* is linked, use the **report_design** command. To view the operating conditions defined in the specified library, use the **report_lib** command. To view the operating conditions set on the cells or blocks library, use the **report_cell** command.

To remove operating conditions from the *current design*, use the **remove_operating_conditions** or the **reset_design** command.

When process factor, operating temperature, and operating voltage deviate from their nominal values, the tool uses delay scaling among multiple libraries.

PrimeTime derives cell instance voltages as follows. Please refer to PrimeTime multi-voltage manual for complete list precedence rules for deriving voltages on PAD cells and deriving cell temperature.

- Default supply voltage in library cell definition (voltage_map in Liberty syntax)
- set_operating_conditions applied at the design level
- set_voltage on a supply net
- set_voltage on a PG pin

Since operating condition defines only single voltage value caution should be exercised when setting operating conditions on designs containing multi-rail cells such as level shifters and power management cells. PrimeTime applies the operating condition voltage value only to the first library power rail (i.e., the first non-ground voltage_map line in .lib). Thus order of voltage_map entries in .lib is important in this partial multi-voltage flow.

The correct way to completely avoid this issue on multi-rail cells is to use proper multi-voltage flow such as UPF that defines all design rails using fset_voltage command. To verify voltage values of all rails of a multi-rail cell use command **report_power_pin_info**.

An alternative is not to use any set_operating_conditions command and thus all power rails will default to the library nominal voltage defined by voltage_map. Or use the design-wide set_operating_conditions but add set_voltage on specific PG pins of all multi-rail cells. Also, Liberty libraries should be created with the first voltage_map to match the nominal voltage nom_voltage.

EXAMPLES

The following example shows an operating condition definition, as it appears in the source text of a library.

```
operating_conditions("BCCOM") { process : 0.6 ; temperature : 20 ; voltage : 5.25 ;  
tree_type : "best_case_tree" ; }
```

The name of this set of operating conditions is *BCCOM*. The parameters are defined as follows:

process – A floating-point number that represents the characteristics of a semiconductor manufacturing process.

temp – A floating-point number that represents the temperature of the defined environment.

voltage – A floating-point number that defines the upper boundary of the voltage range in which the defined environment operates. The lower boundary is always 0.0.

tree-type – The interconnect model for the environment. The tool uses the interconnect model to select a formula for calculating interconnect delays. Three models are available: **best_case_tree** that uses a lumped RC model, **worst_case_tree** in

which all loads assume full wire resistance, and **balanced_tree** in which all loads share the wire resistance evenly.

The following example uses operating condition *WCIND* found in the *my_lib.db* library to set the operating conditions to *WCIND* if the *link_path* is *my_lib.db*.

```
pt_shell> set_operating_conditions WCIND
```

The following example uses operating condition *WCIND* found in the *other_lib.db* library to set the operating conditions to *WCIND* if the *link_path* is *other_lib.db*.

```
pt_shell> set_operating_conditions WCIND -library other_lib.db
```

In the following example, the *BCCOM* values are used for minimum operating conditions and the *WCCOM* values are used for maximum operating conditions. When reporting maximum delays, the delays are computed for the maximum operating conditions. When reporting minimum delays, the delays are computed for the minimum operating condition.

```
pt_shell> set_operating_conditions -min BCCOM -max WCCOM \  
-analysis_type bc_wc
```

SEE ALSO

create_operating_conditions (2), **remove_operating_conditions** (2), **report_cell** (2), **report_design** (2), **report_lib** (2), **report_timing** (2), **report_power_pin_info** (2), **reset_design** (2), **set_voltage** (2), **set_temperature** (2), **define_scaling_lib_group** (2), **default_oc_per_lib** (3).

set_opposite

Sets two ports to be logically opposite.

SYNTAX

```
string set_opposite port1 port2
stringport1
stringport2
```

ARGUMENTS

port1
Specifies the first input port.

port2
Specifies the second input port.

DESCRIPTION

Defines two input ports in the current design as logically opposite. This information is used during synthesis to eliminate redundant ports to improve optimization quality.

Logical inconsistencies are checked for in relationships between ports. Specifying an inconsistent logical relationship between ports is not allowed.

Use the **reset_design** command to remove this property from a port.

The logical relationship is characterized by the **characterize_context** and **create_timing_context** commands and is exported to synthesis using the **write_context** command.

EXAMPLES

The following example sets two input ports "A" and "B" to be logically opposite.

```
pt_shell> set_opposite A [get_ports B]
1
```

The following example sets up a contradictory relationship between two ports and generates an error message.

```
pt_shell> set_opposite A B
1
pt_shell> set_equal A B
Error: Can't set opposite ports equal in design 'top': 'A' 'B'. (DDB-23)
0
```

SEE ALSO

`set_equal` (2), `creating_timing_context` (2), `characterize_context` (2), `write_context` (2).

set_output_delay

Sets output path delay values for the current design.

SYNTAX

```
string set_output_delay [-clock clock_name]  
[-reference_pin pin_port_name]  
[-clock_fall]  
[-level_sensitive]  
[-rise]  
[-fall]  
[-max]  
[-min]  
[-add_delay]  
[-network_latency_included]  
[-source_latency_included]  
[-group_path group_name]  
delay_value  
port_pin_list  
  
list clock_name  
list pin_port_name  
string group_name  
float delay_value  
list port_pin_list
```

ARGUMENTS

-clock *clock_name*

Specifies the name of a clock to which the specified delay is to be related.
If you specify **-clock_fall**, you must also specify **-clock**.

-reference_pin *pin_port_name*

Specifies the clock pin or port to which the specified delay is related. If you use this option, and if propagated clocking is being used, the delay value is related to the arrival time at the specified reference pin, which is clock source latency plus its network latency from the clock source to this reference pin. The options **-network_latency_included** and **-source_latency_included** cannot be used at the same time as the **-reference_pin** option. For ideal clock network, only source latency is applied.

The pin specified with the **-reference_pin** option should be a leaf pin or port in a clock network, in the direct or transitive fanout of a clock source specified with the **-clock** option. If multiple clocks reach the port or pin where you are setting the input delay, and if the **-clock** option is not used, the command considers all of the clocks.

-clock_fall

Indicates that the delay is relative to the falling edge of the clock. If you specify **-clock_fall** with **-reference_pin**, the delay is relative to the falling transition of the reference pin. If you specify **-clock**, the default is the rising edge or the rising transition of the reference pin. If you specify **-clock_fall**, you must also specify **-clock**.

-level_sensitive

Indicates that the destination of the delay is a level-sensitive latch. This allows the tool to derive a setup and hold relationship for paths to this port as if it were a level-sensitive latch. If you do not use **-level_sensitive**, the output delay is treated as if it were a path to a flip-flop.

-rise

Indicates that *delay_value* refers to a rising transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-fall

Indicates that *delay_value* refers to a falling transition on specified ports of the current design. If you do not specify **-rise** or **-fall**, rising and falling delays are assumed to be equal.

-max

Indicates that *delay_value* refers to the longest path. If you do not specify **-max** or **-min**, maximum and minimum output delays are assumed to be equal.

-min

Indicates that *delay_value* refers to the shortest path. If you do not specify **-max** or **-min**, maximum and minimum output delays are assumed to be equal.

-add_delay

Indicates that delay information is to be added to the existing output delay, instead of overwriting the output delay. Using **-add_delay**, you can capture information about multiple paths leading from an output port that are relative to different clocks or clock edges. For example, **set_output_delay 5.0 -max -rise -clock phil {OUT1}** removes all other maximum rise output delays from **OUT1**, because **-add_delay** is not specified. Other output delays with a different clock or with **-clock fall** are removed.

In another example, **-add_delay** is specified: **set_output_delay 5.0 -max -rise -clock phil -add_delay {Z}**. If there is an output maximum rise delay for **Z** relative to the clock **phil** rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

-network_latency_included

Specifies whether the clock network latency should not be added to the output delay value. If this option is not specified, the clock network latency of the related clock will be added to the output delay value. It has no effect if the clock is propagated or the output delay is not specified with respect to any clock.

-source_latency_included

Specifies whether the clock source latency should not be added to the output delay value. If this option is not specified, the clock source latency of the related clock will be added to the output delay value. It has no effect if the output delay is not specified with respect to any clock.

-group_path *group_name*

Specifies the name of a group into which paths ending at the specified ports or pins are to be added. If the group does not already exist, one is created. This is equivalent to specifying the separate command **group_path -name** *group_name -to port_pin_list* in addition to **set_output_delay**. If you do not specify **-group_path**, the existing path grouping does not change.

delay_value

Specifies the path delay in units consistent with the technology library used during analysis. The *delay_value* represents the amount of time before a clock edge that the signal is required. For maximum output delay, this represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time.

port_pin_list

Specifies a list of output port or internal pin names in the current design to which *delay_value* is assigned. If more than one object is specified, the objects are enclosed in braces ({}).

DESCRIPTION

This command sets output path delay values for the current design. The input and output delays characterize the operating environment of the current design when used with **set_load** and **set_driving_cell**.

The **set_output_delay** command sets output path delays on output ports relative to a clock edge. Output ports have no output delay unless specified. For inout (bidirectional) ports, you can specify the path delays for both input and output modes.

To describe a path delay to a level-sensitive latch, use the **-level_sensitive** option. If the latch is positive-enabled, set the output delay relative to the rising clock edge; if it is negative-enabled, set the output delay relative to the falling clock edge. If time is borrowed at that latch, subtract that time from the path delay to the latch when determining output delay.

The **characterize_context** command automatically sets input and output delay, drive, and load values based on the environment of a cell instance.

PrimeTime adds input delay to path delay for paths starting at primary inputs and to output delay for paths ending at primary outputs.

The **-group_path** option modifies the path grouping. Path grouping affects the maximum delay cost. The worst violator within each group adds to the cost.

If a reference pin is not reachable by any given clock, zero arrival time is assumed. If no clock is specified with a reference pin and this reference pin is not reachable by any clock, an unconstrained path is reported. This behavior is changed after X0506. The new behavior is these invalid constraints will be ignored and path will be constrained by whatever it should be as if no such constraints are defined at all. The **check_timing** command generates a warning if the reference pin is not reachable by any active clock, or if multiple clocks reach the reference pin and no clock is specified in the command.

set_output_delay

To get a report on the latency calculation using a reference pin, use **report_timing -path_type full_clock** or **report_timing -path_type full_clock_expanded**.

To list output delays associated with ports, use **report_port**. To list output delays of internal pins, use **report_timing**. To list the path groups that are defined, use **report_path_group**.

To remove output delay values, use **remove_output_delay** or **reset_design**. To modify paths grouped with the **-group_path** option, use the **group_path** command to place the paths in another group or the default group.

EXAMPLES

The following example sets an output delay of 1.7 relative to the rising edge of CLK1 for all output ports in the design.

```
pt_shell> set_output_delay 1.7 -clock CLK1 [ all_outputs ]
```

The following example sets the input and output delays for the bidirectional port INOUT1. The input signal arrives at INOUT1 2.5 units after the falling edge of CLK1. The output signal is required at INOUT1 at 1.4 units before the rising edge of CLK2.

```
pt_shell> set_input_delay 2.5 -clock CLK1 -clock_fall { INOUT1 }
pt_shell> set_output_delay 1.4 -clock CLK2 { INOUT1 }
```

The following example models the situation where there are three paths from output port OUT1. One of the paths is relative to the rising edge of CLK1. Another path is relative to the falling edge of CLK1. The third path is relative to the falling edge of CLK2. The **-add_delay** option indicates that new output delay information does not cause the removal of old information.

```
pt_shell> set_output_delay 2.2 -max clock CLK1 -add_delay { OUT1 }
pt_shell> set_output_delay 1.7 -max clock CLK1 -clock_fall -add_delay { OUT1 }
pt_shell> set_output_delay 4.3 -max clock CLK2 -clock_fall -add_delay { OUT1 }
```

In the following example, two different maximum delays and two minimum delays for port Z are specified using **-add_delay**. Because the information is relative to the same clock and clock edge, only the largest of the maximum values and the smallest of the minimum values are maintained (in this case, 5.0 and 1.1). If **-add_delay** is not used, the new information overwrites the old information.

```
pt_shell> set_output_delay 3.4 -max -clock CLK1 -add_delay { Z }
pt_shell> set_output_delay 5.0 -max -clock CLK1 -add_delay { Z }
pt_shell> set_output_delay 1.1 -min -clock CLK1 -add_delay { Z }
pt_shell> set_output_delay 1.3 -min -clock CLK1 -add_delay { Z }
```

The following example shows how to use the **-group_path** option to add ports into a named group. Note that without this option, paths to these ports are included in the CLK group.

```
pt_shell> set_output_delay 4.5 -max -clock CLK -group_path busA {busA[*]}
```

SEE ALSO

all_outputs (2), **characterize_context** (2), **create_clock** (2), **current_design** (2),
group_path (2), **remove_output_delay** (2), **report_design** (2), **report_path_group** (2),
report_port (2), **reset_design** (2), **report_timing** (2), **check_timing** (2),
set_driving_cell (2), **set_load** (2), **set_max_delay** (2), **set_output_delay** (2).

set_parasitic_corner

Sets a parasitic corner for the timing analysis in the presence of variation-aware parasitics.

SYNTAX

Boolean **set_parasitic_corner** -name *corner_name*
file_name

string *file_name*
string *corner_name*

ARGUMENTS

-name *corner_name*
Specifies the name of the parasitic corner to be used from the corner file. A given corner file can contain definitions of a number of parasitic corners and this option specifies which corner definition is to be used to be set as the corner for analysis.

file_name
Specifies the name of the parasitic corner or corner file. The file is supposed to contain the percentage variation values for all the parasitic parameters in the variation-aware parasitics.

DESCRIPTION

The **set_parasitic_corner** command reads parasitic corner information from a disk file and sets it as the corner for analysis in the presence of variation-aware parasitics. The command succeeds only if variation-aware parasitics were previously annotated.

The syntax of the corner parasitic file is very simple. It simply defines various parasitic corners by specifying the VARIATION_RATIOS for all the parameters.

A VARIATION_RATIO signifies the actual amount of variation from the mean (or nominal) for a given parameter. A VARIATION_RATIO is expressed in terms of the percentage variation of a given parameter with respect to its variation coefficient. For example, a given parameter has a variation coefficient of 0.09. The actual amount of variation allowed for this parameter is between -0.27 and 0.27. The VARIATION_RATIO is the multiplier to the variation coefficient that gets us to the actual amount of variation for the parameter. So, by definition, the value of VARIATION_RATIO has to be between -3.0 and +3.0.

Technically, the syntax of a corner file can be expressed as:

```
(CORNER_NAME: <name of the corner> (PARAM_NAME PARAM_VARIATION_RATIO)*)+
```

The corner file should contain definition for at least one corner. A corner can contain VARIATION_RATIO values for ZERO or more number of parameters. If a parameter is not specified in the corner definition, the VARIATION_RATIO is assumed to be

ZERO.

Comments can be writtern in the file following TCL style - so, anything after the characters " #" (space and pound) on a line is assumed to be a comment till the end of the line. Only line comments are supported.

An example of the corner file is as follows:

```
CORNER_NAME: CMAX M1_W 3.0 M2_T 3.0 ILD_c_T -0.04 CORNER_NAME: RCMAX ## my
definition of RCMAX M1_W 3.0 M2_T 2.0 ILD_c_T -0.22 CORNER_NAME: RCMIN M1_W -3.0
M2_T -3.0 ILD_c_T 0.4
```

This indicates that the CMAX corner parasitics must be inferred where the M1_W parameter is changed by positive (3.0 * variation_coeff of M1_W) from the nominal and so on.

You can issue this command multiple times to change the corner for analysis. Only the last command takes effect. You can remove the parasitic corner setting via **remove_parasitic_corner** command.

Note that this command has no effect on **write_parasitics** command. This command sets the corner only for analysis and does not change the variation-aware parasitics.

The parasitic attributes like **rc_network** and **total_coupling_capacitance** are all modified to reflect the corner values.

EXAMPLES

The following example reads the parasitics corner file "./corner_file" from disk and uses the settings of the RCMAX corner to set the default parasitic corner on the design with variation-aware parasitics.

```
pt_shell> set_parasitic_corner -name RCMAX ./corner_file
```

SEE ALSO

remove_parasitic_corner (2), **read_parasitics** (2), **report_annotated_parasitics** (2),

set_port_fanout_number

Sets number of external fanout points on ports.

SYNTAX

```
string set_port_fanout_number [-min] [-max] fanout_number port_list  
int    fanout_number  
list   port_list
```

ARGUMENTS

-min

Specifies the value for minimum condition.

-max

Specifies the value for maximum condition.

fanout_number

Number of external fanout points (Range: 0 to 100000).

port_list

Specifies a list of ports. Each element in the list is either a collection of ports or a pattern that matches ports on the current design.

DESCRIPTION

Sets the number of external fanout pins for ports in the current design. The number of pins is used (along with wire load models) to calculate capacitance and resistance of nets.

To remove port fanout number values, use **remove_port_fanout_number**. To show port fanout number information, use **report_port -wire_load**.

EXAMPLES

This example sets the external wire load model for ports OUT1* to 70x70 and specifies an external fanout number of 2.

```
pt_shell> set_wire_load_model 70x70 [get_ports OUT1*]  
pt_shell> set_port_fanout_number 2 [get_ports OUT1*]
```

SEE ALSO

collections (2), **remove_port_fanout_number** (2), **report_port** (2), **set_wire_load_model** (2).

set_power_analysis_options

Sets the options for power analysis.

SYNTAX

```
int set_power_analysis_options
[-static_leakage_only]
[-variation_quantile quantile]
[-waveform_interval sampling_interval]
[-cycle_accurate_cycle_count cycles]
[-cycle_accurate_clock clock]
[-waveform_format fsdb | out | none]
[-waveform_output file_prefix]
[-include_top | all_without_leaf | all_with_leaf]
[-include_groups group_list]
[-cells cell_list]

float    quantile
float    sampling_interval
int      cycles
string   clock
string   file_prefix
list     group_list
list     cell_list
```

ARGUMENTS

- static_leakage_only**
Specifies that averaged power analysis will only calculate static leakage power and skip dynamic power calculation. The option only applies to averaged power analysis.
- variation_quantile *quantile***
Specifies that the leakage variation report should report the specified leakage quantile in the quantile column of the report. The value is between 0.0 and 1.0. Default value is 0.99. This option only applies to statistical leakage power variation analysis.
- waveform_interval *sampling_interval***
Specifies the sampling interval that is used for power waveform. The default value is the timescale from VCD file. This option only applies to time_based power analysis. The option will be ignored if VCD comes from RTL or zero delay simulation.
- cycle_accurate_cycle_count *cycles***
Specifies the number of clock cycle over which the power consumed by the design is considered constant when VCD is RTL or zero delay. Default value is 1. This option only applies to time_based power analysis, specifically for RTL or zero delay VCD or its equivalents (VPD etc.).
- cycle_accurate_clock *clock***
Specifies the reference clock for time_based power analysis, when VCD is RTL

or zero delay. The power consumed by the design is considered constant over one or more cycles of the specified clock. The default is the fastest clock in the design. This option only applies to `time_based` power analysis, specifically for RTL or zero delay VCD or its equivalents (VPD etc.).

`-waveform_format fsdb | out | none`

Specify the format of the file in which the waveform data is to be written. The value for this option can be: `fsdb`, `out` or `none`. Default is `fsdb`. If it's set to `none`, waveform data is not to be dumped, but peak power is still calculated. This option only applies to `time_based` power analysis.

`-waveform_output file_prefix`

Specify the prefix of the file in which the waveform data is to be written. The default is `primetime_px`. This option only applies to `time_based` power analysis.

`-include top | all_without_leaf | all_with_leaf`

Specifies the objects in the design hierarchy to be monitored for waveform generation. The tool will only be able to show the power waveforms limited by this option. There are three allowed values for the option. If the value is `top`, only top level design is monitored for power waveform generation. If the value is `all_without_leaf`, all design hierarchies except leaf cells are monitored. If the value is `all_with_leaf`, all design hierarchies including leaf cells are monitored. The default value is `all_without_leaf`. This option only applies to `time_based` power analysis.

`-include_groups group_list`

Specifies the power groups to be monitored for power waveform generation. The tool will be able to show the power waveforms for the specified power groups. By default, no power group is monitored. This option only applies to `time_based` power analysis.

`-cells cell_list`

Specifies the cell names or collections of cells for which power needs to be calculated. By default, PrimeTime PX calculates power for the whole design.

DESCRIPTION

There are several power analysis mode, such as `averaged`, `time_based`, `leakage_variation`, in PrimeTime PX. The mode is specified by the **power_analysis_mode** variable before the major power analysis commands. The working command for all power analysis mode is **update_power**. **update_power** itself does not take any option. The **set_power_analysis_options** command is used between **power_analysis_mode** variable and **update_power** command to specify various options for different modes of power analysis in PrimeTime PX. The options from this command will control how power analysis of the specified mode behaves.

Each option may be used for one or more modes of power analysis. If an option is not supposed to be used in a power analysis mode, the command will stop and an error message will be issued. **set_power_analysis_options** with no option will reset all the power analysis options to default values. New issue of **set_power_analysis_options** will overwrite the previous one and reset the rest of power analysis options to default values. Option `-cycle_accurate_clock` and `-cycle_accurate_cycle_count` are exclusive with option `-waveform_interval` in this command.

The following is the table of options supported in each power analysis mode.

				time_based	
		averaged	time_based	(RTL or zero delay vcd)	1
leakage_variation					
-----+-----+-----+-----+-----+-----					
-					
static_leakage_only		Yes	No	No	
No					
-----+-----+-----+-----+-----+-----					
-					
variation_quantile		No	No	No	
Yes					
-----+-----+-----+-----+-----+-----					
-					
waveform_interval		No	Yes	No	
No					
-----+-----+-----+-----+-----+-----					
-					
cycle_accurate_cycle_count		No	No	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
cycle_accurate_clock		No	No	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
waveform_format		No	Yes	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
waveform_output		No	Yes	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
include		No	Yes	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
include_groups		No	Yes	Yes	
No					
-----+-----+-----+-----+-----+-----					
-					
cells		Yes	Yes	Yes	

Yes

EXAMPLES

The following example will cause PrimeTime PX to do time_based power analysis. The time interval is 10 ns. The waveform output file will be my_design.fsdb.

```
pt_shell> set power_analysis_mode time_based
pt_shell> read_vcd -strip_path this_strip_path my_design.vcd
pt_shell> set power_analysis_options -waveform_interval 10 -
waveform_output my_design
pt_shell> update_power
```

The following example will cause PrimeTime PX to do averaged power analysis. Only leakage power is calculated for fast turn-around time.

```
pt_shell> set power_analysis_mode averaged
pt_shell> set power_analysis_options -static_leakage_only
pt_shell> update_power
```

The following example will cause PrimeTime PX to do time_based power analysis for an RTL based design. The reference clock is *clk*.

```
pt_shell> set power_analysis_mode time_based
pt_shell> source name_mapping_info.tcl
pt_shell> read_vcd -rtl -strip_path this_strip_path my_rtl_design.vcd
pt_shell> set power_analysis_options -cycle_accurate_clock clk
pt_shell> update_power
```

The following example will cause PrimeTime PX to do leakage power variation analysis. The variation quantile is 0.9.

```
pt_shell> set power_analysis_mode leakage_variation
pt_shell> set power_analysis_options -variation_quantile quantile
pt_shell> update_power
```

SEE ALSO

```
power_analysis_mode(3),  
report_power_analysis_options(2),  
update_power(2),  
report_power(2).
```

set_power_derate

Sets power derating factors for either the current design or a specified list of instances (cells, library cells, or power groups).

SYNTAX

```
int set_power_derate
    [-switching]
    [-internal]
    [-leakage]
    [-groups group_name_list]
    derate_value
    object_list
```

Data Type

```
list    group_name_list
float   derate_value
list    object_list
```

ARGUMENTS

-switching
Indicates that the *derate_value* specified should be applied to switching power.

-internal
Indicates that the *derate_value* specified should be applied to internal power.

-leakage
Indicates that the *derate_value* specified should be applied to leakage power only. For CCS power library, it should be applied to both gate leakage and subthreshold leakage power.

-groups
Specifies the power group(s) to which the *derate_value* will be applied. The *derate_value* will apply directly to each cell object in the power group.

derate_value
Specifies the power derating factor that will be applied to the specified power components as a scalar multiplicative factor.

object_list
Specifies current design or a list of cells or library cells to which the specified power derating factor will be applied.

DESCRIPTION

Sets power derating factors on different power components for either the current design, a list of cells, or library cells in the current design. If no object or

power group is specified, the power derating factor will be set on the current design. The power derating factors will be used as a scalar multiplicative factor in power calculation.

Power derating factors affect power analysis results in power reports and power attributes. Power analysis results are multiplied by the derating factors. If power derating factors are not specified, the value of 1.0 is assumed.

If the options `-internal`, `-switching`, `-leakage` is specified in the **set_power_derate** command, the specified power derating factor will only be applied to internal, switching or leakage power accordingly. If none of the `-internal`, `-switching` or `-leakage` option is specified, the power derating factor applies to all the power components, which includes internal, switching and leakage power. User can first set a generic power derating factor that applies to all power components, and then refine it by setting specific power derating factors for particular power components on particular design objects.

The `object_list` option can be used to set power specific derating factors on instances (cells or library cells) in the design. On each instance the `-switching`, `-internal`, and `-leakage` options can be used in the same way as outlined above to specify exactly how the derating factors should be applied to each instance in the object list.

When applying power derating factors the following priority is used (in decreasing order of precedence):

- 1) Leaf Cell or Power Group
- 2) Hierarchical Cell
- 3) Library Cell
- 4) Design

Therefore, when power derating factors are being applied for a cell, PrimeTime PX first checks if they have been set for the cell itself (leaf cell). It then checks if it has been set for any of the cells hierarchical parents. If no derate factors are found, it checks for the library cell and after that it uses the factors set globally on the design.

If power derating factors are set multiple times on the same design object for the same power component, the later value will override the previous value.

The `-groups` option can be used to set specific derating factors on all the cell objects in particular power groups. Power group can be user generated or default, such as `clock_network`, `memory`, `register`. They are considered as a collection of leaf and hierarchical cells. When applying power derating factors, the factors are set directly on the cell objects in the collection. There is one exception. Currently in PrimeTime-PX, clock network power includes the clock pin power of registers by default. The clock network power derating factors will not be applied to registers, but only applied to the clock pin power of the registers when it was included in the clock network power. The power derating factors set on clock network power group will be reset with `reset_power_derate` command on current design or without any option.

In addition, if the `-switching` option is specified and the `object_list` contains any hierarchical cells, all cells within that hierarchical cell and also all lower hierarchical cells will have that derating factor set on them.

The `-groups` option can be used to set specific derating factors on all the cell objects in particular power groups. Power group can be user generated or default, such as `clock_network`, `memory`, `register`, etc.. If a derate factor is set on a power group, all cell objects in the power group will be applied to the specified derating factors.

Power derating factors affect power values shown in power reports. The intended usage mode of this command is that the user first sets global or default derates on the design and then may use the `object_list` option to set specific derate values for cells or library cells in the design. To set derates globally on a design simply issue the command with no object list specified.

For example, to set an power derating factor of value 0.95 on all cells in the design use:

- **`set_power_derate 0.95`**

To set derating values back to the default (derating factor of 1.0 for every instance in the design), use the **`reset_power_derate`** command.

EXAMPLES

The following example sets switching power derate and internal power derate to 0.9.

```
pt_shell> set_power_derate 0.9 -switching -internal
```

The following example sets the power derate to 0.5 for all power components on the default clock network power group.

```
pt_shell> set_power_derate 0.5 -groups clock_network
```

The following example sets a leakage power derating factor of 1.1 on all instances of library cell IV in the library MY_LIB:

```
pt_shell> set_power_derate -leakage 1.1 [get_lib_cells MY_LIB/IV]
```

Assuming that cell, "inv" is a particular instantiation of MY_LIB/IV in the design the following command overwrites the power derating factor for the instance "inv" only:

```
pt_shell> set_power_derate -leakage 1.05 [get_cells inv]
```

The following example illustrates how to set a power derating factor except leakage power on all cells (including sub-hierarchies) within the hierarchy top/H1:

```
pt_shell> set_power_derate -switching -internal 1.05 [get_cells top/H1]
```

SEE ALSO

```
report_power(2)  
report_power_calculation(2)  
report_power_derate(2)
```

```
reset_power_derate(2)
```


set_program_options

Defines some runtime options.

SYNTAX

```
int set_program_options  
[-enable_high_capacity]  
[-disable_high_capacity]  
[-enable_eco]  
[-disable_eco]  
[-enable_fast_analysis]
```

-enable_high_capacity

Indicates that you should run the tool in high capacity mode, which means reduced peak memory footprint. The actual capacity improvement can be controlled by the **sh_high_capacity_effort** variable. Using this improved capacity mode does not change the results of timing analysis. You can only enable high capacity mode when there are no designs and libraries loaded in the program's memory.

By default, high capacity mode is enabled in default effort and you can use the **sh_high_capacity_effort** variable and this command to change the default behavior. You can query whether the high capacity mode is currently enabled through the **sh_high_capacity_enabled** read-only variable.

-disable_high_capacity

This option disables the high capacity mode of the program and returns to the default normal analysis mode. Usually it means larger peak memory footprint. You can only disable high capacity mode when there are no designs and libraries loaded in memory.

-enable_eco

This option enables the ECO mode of the program. ECO mode is disabled by default.

-disable_eco

This option disables the ECO mode of the program. Some commands, such as the `connect_net`, `create_cell`, `create_net`, `disconnect_net`, `get_alternative_lib_cells`, `insert_buffer`, `read_parasitics -eco`, `remove_buffer`, `remove_cell`, `remove_net`, `rename_cell`, `rename_design`, `rename_net`, `report_alternative_lib_cells`, `size_cell`, `swap_cell`, and `write_changes` commands, should not be used in the disabled ECO mode. Performance and capacity might be improved.

-enable_fast_analysis

This option enables fast analysis mode of the program. Performance and capacity might be improved. Fast analysis mode is disabled by default.

DESCRIPTION

This command provides you with options to control runtime behavior. You can provide guidelines on the trade-off between capacity and performance of the timing analysis without impacting the quality of results.

When high capacity mode is enabled, the tool requires some dedicated disk space to temporarily store some design data while analyzing others. The storage space used is

defined by the existing **pt_tmp_dir** variable, which by default is the local disk partition `/tmp` of the machine the tool is running on. The size of the disk storage space required by high capacity mode is estimated to be at least half of the expected peak memory footprint when the same analysis is done in the default non-high capacity mode. The potential capacity improvement might not be fully realizable in case of insufficient disk space. Upon exit the session, the disks space is automatically released.

It is also worth noting that high capacity mode is not preserved by the **save_session** command; therefore, the **restore_session** command does not override the current session mode at time of restore. This provides the flexibility for the **restore_session** command to resume the session to a different high capacity state independent from when it was saved. For example, you can analyze in high capacity mode, save the session, and then restore into a normal (non-high capacity) mode for further reporting. Or you can analyze in normal mode on a large and more capable machine, save the session, and then restore it in high capacity mode using a less demanding machines for further analysis.

Fast analysis mode provides high performance and accuracy for early stage timing. It overrides some variables and options to configure the tool for higher performance for early analysis runs. The **sh_fast_analysis_mode_enabled** read-only variable keeps track of whether the fast analysis mode is enabled. Fast analysis mode is preserved by the **save_session** command; therefore, the **restore_session** command always restores the status of fast analysis mode that is saved.

EXAMPLES

The following example shows how to enable high capacity mode with the default settings,

```
pt_shell> set sh_high_capacity_effort high
Information: The setting will be effective after next 'set_program_options' command
. (PTHC-001)
high
pt_shell> set program_options -enable_high_capacity
Information: enabling high capacity analysis in 'high' effort..... (PTHC-001)
pt_shell> printvar sh_high_capacity_enabled
sh_high_capacity_enabled = "true"
```

The following example shows how to enable fast analysis mode,

```
pt_shell> set program_options -enable_fast_analysis
Information: Fast analysis mode is enabled. (PTANA-002)
pt_shell> printvar sh_fast_analysis_mode_enabled
sh_fast_analysis_mode_enabled = "true"
```

SEE ALSO

sh_high_capacity_effort(3)
sh_high_capacity_enabled(3)
sh_eco_enabled(3)
sh_fast_analysis_mode_enabled(3)
pt_tmp_dir(3)

set_propagated_clock

Specifies propagated clock latency.

SYNTAX

```
string set_propagated_clock object_list  
list object_list
```

ARGUMENTS

object_list
Specifies a list of clocks, ports, or pins.

DESCRIPTION

Specifies that delays be propagated through the clock network to determine latency at register clock pins. If not specified, ideal clocking is assumed. Ideal clocking means clock networks have a specified latency (designated by the **set_clock_latency** command), or zero latency, by default. Propagated clock latency is used for post-layout, after final clock tree generation. Ideal clock latency provides an estimate of the clock tree for pre-layout.

If the **set_propagated_clock** command is applied to pins or ports, it affects all register clock pins in the transitive fanout of the pins or ports. If it is applied to an object (clock, port, or pin) on which network latency is already defined, then a warning is issued.

Virtual clocks do not have any source, and they cannot affect any register in the design. Thus, virtual clocks cannot be made propagated, and an error is issued if **set_propagated_clock** is applied on a virtual clock.

To undo **set_propagated_clock**, use the **remove_propagated_clock** command or use the **set_clock_latency** command to provide an ideal latency.

To see propagated clock attributes on clocks, use the **report_clock** command.

Limitations: CRPR does not support propagated clocks set on pins or ports unless they are source pins or ports of common clock for a path. If the variable **timing_remove_clock_reconvergence_pessimism** is set to TRUE then propagated clocks should be defined on clock objects, not pins or ports.

EXAMPLES

The following example specifies to use propagated clock latency for all clocks in the design.

```
pt_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

`timing_remove_clock_reconvergence_pessimism` (3), `remove_propagated_clock` (2),
`report_clock` (2), `set_clock_latency` (2).

set_pulse_clock_max_transition

Sets maximum transition for pulse generator input and pulse clock network with respect to the main library trip-points.

SYNTAX

```
string set_pulse_clock_max_transition transition_value  
[-rise][-fall]  
[-transitive_fanout]  
object_list
```

```
float transition_value  
list object_list
```

ARGUMENTS

-rise

Specifies rising transition for all selected pins.

-fall

Specifies falling transition for all selected pins.

-transitive_fanout

Specifies the constraint is set at the transitive fanout of pulse generator. If this option is not set only the input of the pulse generators are constrained.

transition_value

Sets the transition limit (Value >= 0). This is the maximum transition time in main library time units.

object_list

Provides a list of pulse generator cell, pulse generator lib cell, clock and design. .

DESCRIPTION

Specifies maximum transition in the pulse clock network at the forward of pulse generator cell, pulse generator lib cell, clock and design if **-transitive_fanout** option is set. By default, maximum transition constraint is set at the input of the pulse generator of the specified object list. If the cell or lib cell is not a pulse generator or clock or design does not have any pulse generators, then this maximum transition limit has no effect. The pulse clock network or the input of pulse generator is expected to have transition time less than the specified *transition_value*.

Maximum transition constraint set on design by **set_max_transition** command applies to the pulse clock network and input of pulse generator as also maximum transition constraint set on the pins in the library. Transition constraints are interpreted in main library trip-points and derate. During slack computation, the constraints are scaled to the pin trip-points and derate. The case of conflicting constraints on a

pin, the tightest constraint applies.

The maximum transition constraint can be optionally restricted to rising transitions only or falling transitions only by using **-rise** or **-fall** options respectively. If **-rise** or **-fall** options are not specified, both rise and falling transitions are constrained.

The **report_constraint -pulse_clock_max_transition** and **report_pulse_clock_max_transition** commands show maximum transition constraint evaluations. The limit with appropriate scaling used for slack computation is reported.

To remove maximum pulse clock transition limits, use **remove_pulse_clock_max_transition**.

EXAMPLES

The following example sets a maximum transition limit of 0.4 units on input of all the cells corresponding to lib cell pulse_rise_high in library celllib.

```
pt_shell> set_pulse_clock_max_transition 0.4 {"celllib/pulse_rise_high"}
```

The following example sets a maximum transition limit of 0.4 units on all the pulse clock networks in the clock network clk.

```
pt_shell> set_pulse_clock_max_transition -transitive_fanout 0.4 [get_clocks  
clk]
```

SEE ALSO

current_design (2), **remove_pulse_clock_max_transition (2)**,
report_pulse_clock_max_transition (2), **report_constraint (2)**.

set_pulse_clock_max_width

Sets maximum pulse width constraint for pulse generator network.

SYNTAX

```
string set_pulse_clock_max_width pulse_width_value  
[-transitive_fanout]  
object_list
```

```
float pulse_width_value  
list object_list
```

ARGUMENTS

-transitive_fanout

The constraints will be applied to the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

pulse_width_value

Sets the pulse width limit (Value >= 0).

object_list

Provides a list of pulse generator cell, pulse generator lib cell, clock and design. .

DESCRIPTION

Specifies maximum pulse width limit for the the pulse clock network at the forward of pulse generator cell by pulse generator instance name or pulse generator library cell. Setting the pulse width constraint by clock applies the constraint to the forward of all pulse generators driven by the clock. Setting by pulse width constraint by design applies the constraint to all the pulse clock networks in the design. In presence of conflicting constraint the most restrictive constraint is used. If the cell or lib cell is not a pulse generator or clock or design does not have any pulse generators, then this maximum width limit has no effect. The pulse clock network is expected to have pulse width time less than the specified *pulse_width_value*.

The high or low pulse width constraint is inferred based on sense propagation. For example, after an inverter, a high pulse width constraint is converted to a low pulse width constraint.

The **report_constraint -pulse_clock_max_width** and **report_pulse_clock_max_width** commands show maximum pulse width constraint evaluations.

To remove maximum pulse width limits, use **remove_pulse_clock_max_width**.

EXAMPLES

The following example sets a maximum pulse width limit of 0.4 units on all the pulse

```
set_pulse_clock_max_width  
1088
```


clock networks in the design.

```
pt_shell> set_pulse_clock_max_width 0.4 [current_design]
```

SEE ALSO

```
current_design (2), remove_pulse_clock_max_width (2), report_pulse_clock_max_width  
(2), report_constraint (2).
```

set_pulse_clock_min_transition

Sets minimum transition at the input of pulse generator with respect to the main library trip-points.

SYNTAX

```
string set_pulse_clock_min_transition transition_value  
[-rise][-fall]  
object_list
```

```
float transition_value  
list object_list
```

ARGUMENTS

-rise

Specifies rising transition for all selected pins.

-fall

Specifies falling transition for all selected pins.

transition_value

Sets the transition limit (Value >= 0). This is the minimum transition time in main library time units.

object_list

Provides a list of pulse generator cell, pulse generator lib cell, clock and design. .

DESCRIPTION

Specifies minimum transition at the input of pulse generator cell and pulse generator lib cell. If the constraint is specified on the clock and design, all the inputs of pulse generators in the clock network and design respectively are constrained. If the cell or lib cell is not a pulse generator or clock or design does not have any pulse generators, then minimum pulse clock transition limit has no effect. The input of pulse generator is expected to have transition time greater than the specified *transition_value*.

Transition constraints are interpreted in main library trip-points and derate. During slack computation, the constraints are scaled to the pin trip-points and derate. The case of conflicting constraints on a pin, the tightest constraint applies.

The minimum transition constraint can be optionally restricted to rising transitions only or falling transitions only by using **-rise** or **-fall** options respectively. If **-rise** or **-fall** options are not specified, both rise and falling transitions are constrained.

The **report_constraint -pulse_clock_min_transition** and **report_pulse_clock_min_transition** commands show minimum transition constraint

evaluations. The constraint with appropriate scaling used for slack computation is reported.

To remove minimum pulse clock transition limits, use **`remove_pulse_clock_min_transition`**.

EXAMPLES

The following example sets a minimum transition limit of 0.4 units on input of all the cells corresponding to lib cell pulse_rise_high in library celllib.

```
pt_shell> set_pulse_clock_min_transition 0.4 {"celllib/pulse_rise_high"}
```

SEE ALSO

```
current_design (2), remove_pulse_clock_min_transition (2),  
report_pulse_clock_min_transition (2), report_constraint (2).
```

set_pulse_clock_min_width

Sets minimum pulse width constraint for pulse generator network.

SYNTAX

```
string set_pulse_clock_min_width pulse_width_value  
[-transitive_fanout]  
object_list
```

```
float pulse_width_value  
list object_list
```

ARGUMENTS

-transitive_fanout

The constraints will be applied to the transitive fanout of pulse generators. In this release, specifying or not specifying this option has the same behavior.

pulse_width_value

Sets the pulse width limit (Value >= 0).

object_list

Provides a list of pulse generator cell, pulse generator lib cell, clock and design. .

DESCRIPTION

Specifies minimum pulse width limit for the the pulse clock network at the forward of pulse generator cell by pulse generator instance name or pulse generator library cell. Setting the pulse width constraint by clock applies the constraint to the forward of all pulse generators driven by the clock. Setting by pulse width constraint by design applies the constraint to all the pulse clock networks in the design. Minimum pulse width constraint from the library also applies to the pulse clock network. In presence of conflicting constraint the most restrictive constraint is used. If the cell or lib cell is not a pulse generator or clock or design does not have any pulse generators, then this minimum width limit has no effect. The pulse clock network at the forward of pulse generator is expected to have width more than the specified *pulse_width_value*.

The high or low pulse width constraint is inferred based on sense propagation. For example, after an inverter, a high pulse width constraint is a low pulse width constraint.

The **report_constraint -pulse_clock_min_width** and **report_pulse_clock_min_width** commands show minimum pulse width constraint evaluations.

To remove minimum pulse width limits, use **remove_pulse_clock_min_width**.

EXAMPLES

The following example sets a minimum pulse width limit of 0.4 units on all the pulse clock networks in the clock network clk

```
pt_shell> set_pulse_clock_min_width 0.4 [get_clocks clk]
```

SEE ALSO

`current_design (2)`, `remove_pulse_clock_min_width (2)`, `report_pulse_clock_min_width (2)`, `report_constraint (2)`.

set_qtm_attribute

Sets an attribute to the specified value on QTM object(s).

SYNTAX

```
string set_qtm_attribute  
-class class_name  
attr_name  
value  
[object_names]
```

```
string class_name  
string attr_name  
string value  
list object_names
```

ARGUMENTS

-class *class_name*
Specifies the class of the QTM objects to set attribute on. Allowed values are lib, lib_cell, or lib_pin.

attr_name
Specifies the name of the attribute.

value
Specifies the value of the attribute.

object_names
Specifies the names of the objects on which to set the attribute. Each element in the list is a name to identify an object in the specified class. The object names should not be specified when the object class is either lib or lib_cell. It must be specified when the class is lib_pin, in which case the object names should be the names of ports already defined for the QTM.

DESCRIPTION

The **set_qtm_attribute** command sets attributes on objects. These attributes are either application attributes that are reserved by PrimeTime or defined with **define_qtm_attribute** as user attributes. Please note the order of the command arguments. Also note that PrimeTime commands **list_attribute**, **report_attribute** and **get_attribute** do not work on attributes defined and/or set on the QTM objects.

EXAMPLES

The following example defines a boolean attribute 'is_XYZ' for QTM cell, then sets the value on the QTM cell under construction.

```
pt_shell> define_qtm_attribute -type boolean -class lib_cell "is_XYZ"  
pt_shell> set_qtm_attribute -class lib_cell "is_XYZ" "true"
```

The following example set the attribute 'max_transition' for 2 QTM ports, Because 'max_transition' is an application reserved attribute for the object class 'lib_pin', it should be set directly.

```
pt_shell> set_qtm_attribute -class lib_pin "max_transition" 1.0 {IN, CLK}
```

SEE ALSO

define_qtm_attribute (2), remove_qtm_attribute (2), define_user_attribute (2).

set_qtm_global_parameter

Sets a global parameter for QTM.

SYNTAX

```
string set_qtm_global_parameter  
[-param parameter]  
[-lib_cell lib_cell]  
[-pin pin_name]  
[-clock pin_name]  
[-value parameter_value]
```

```
string parameter  
string lib_cell  
string pin_name  
string pin_name  
float parameter_value
```

ARGUMENTS

- param *parameter***
Specifies the global parameter you want to set. The global parameter can be setup, hold, or clk_to_output.
- lib_cell *lib_cell***
Specifies the lib_cell you want to use to mimic the global parameter.
- pin *pin_name***
Specifies the related pin for the global parameter if you are using lib_cell to mimic the behavior. If the parameter is setup/hold, this pin must be an input pin. If the parameter is clk_to_output, this pin has to be an output pin. If you do not use this option, QTM uses the first encountered input pin that is constrained, in case of setup/hold; and the first output pin which has an edge delay arc coming into it, in case of clk_to_output parameter.
- clock *pin_name***
Specifies the constraining pin in case of setup or hold; and the launch pin in case of clk_to_output parameter.
- value *parameter_value***
The value of the global parameter in time units.

DESCRIPTION

This command sets the value of a desired QTM global parameter. The global parameter can be either setup, hold, or clk_to_output. These parameters specify the global parameter which is added to the arcs. To the QTM setup arc, the time is added to arrive at the final delay. To the QTM hold arc, the global hold time is added to arrive at the final delay. To the QTM edge triggered delay arc the global clk_to_output time is added to arrive at the final delay.

You can specify the global parameters of the sequential element by specifying a cell in the library (**-lib_cell** option), or by giving the actual value, by using **-value** option. If the **-lib_cell** option is specified, you can specify the clock pin and the input pin name. If you do not specify one, the first setup, hold, or clk_to_output arc encountered is chosen (based on which parameter is being set). If only **-clock** option is specified, the first setup, hold, or clk_to_output arc (depending on the global parameter) from that clock pin is chosen. If only the **-pin** option is specified, the first setup, hold, or clk_to_output arc (depending on the type of global parameter) coming into the pin is chosen.

To show the information about the current QTM model, use **report_qtm_model** command.

For a basic description about QTM, please refer to **create_qtm_model** man page. For a more detailed description about QTM, please refer to Chapter 12 of the PrimeTime User Guide.

EXAMPLES

In the following examples where ever **-lib_cell** is used, you must have loaded the technology library and set the qtm technology set. This is done through the following sequence of commands:

```
read_db my_technology_library.db
```

```
set_qtm_technology -library my_technology_library
```

The following example sets the global setup time equivalent to the setup time of DFF1.

```
pt_shell> set_qtm_global_parameter -param setup -lib_cell DFF1
```

The following example sets the global hold time equivalent to the setup time of pin D of DFF1.

```
pt_shell> set_qtm_global_parameter -param setup -lib_cell DFF1 -pin D
```

The following example sets the global clk_to_out time to be 1.5 time units.

```
pt_shell> set_qtm_global_parameter -param clk_to_output -value 1.5
```

SEE ALSO

```
create_qtm_constraint_arc(2), create_qtm_delay_arc(2), create_qtm_drive_type(2),  
create_qtm_load_type(2), create_qtm_model(2), create_qtm_path_type(2),  
report_qtm_model(2), save_qtm_model(2).
```

set_qtm_port_drive

Sets the drive on the QTM port.

SYNTAX

```
string set_qtm_port_drive [-type drive_type] [-value drive_value] [-  
input_transition_rise rtrans] [-input_transition_fall ftrans] [-  
subtract_max_delay_from_total] port_list  
string drive_type  
float drive_value  
float rtrans  
float ftrans  
list port_list
```

ARGUMENTS

-type *drive_type*

Specifies the the global *drive_type* parameter with this option.

-value *drive_value*

Specifies the drive value in terms drive resistance units with this option.

port_list

Specifies a list of input or inout ports on which to set the attribute. Each element in the list is either a collection of QTM ports or a pattern which matches QTM ports.

-input_transition_rise *rtran*

Specifies the input rising transition time associated with the **-input_pin** of the specified drive type to compute the delay for the drive arc for this port. If you do not include this option, the delay table for rising input of the drive arc will be loaded from library as is.

Use the **-input_transition_rise** and **-input_transition_fall** options to capture the transition time associated with the *input_pin* defined for the drive type referred. This can obtain more accurate information on the transition time and delay time for the drive arc to be defined for this port.

-input_transition_fall *ftran*

Specifies the input falling transition time associated with the *input_pin* of the specified drive type to compute the delay for the drive arc for this port. If you do not include this option, the delay table for falling input of the drive arc will be loaded from library as is.

-subtract_max_delay_from_total

Indicates that the drive arc delay computed with max load is to be subtracted from the total delay budgeted to this output port. This affects all types of delay arcs reach this output port. By default, the drive arc delay is added on top of the QTM delay arcs created to the output port and therefore increase the actual delay.

DESCRIPTION

This command sets the drive of the output port of the QTM model. There are two methods for setting the drive:

1. Use one the global *drive_type*, if you have drive types defined.
2. Define the drive in terms of the drive resistance units.

When drive is defined for an output port, by default, the actual delay to the output port will be the delay arc defined by **create_qtm_delay_arc** for the port plus the delay computed for this drive arc. If in your QTM creation, the delay defined by **create_qtm_delay_arc** for the port is actually the maximum delay budgeted for the output and you do not want the drive arc to add extra delay on top of that budget, you can optionally use **-subtract_max_delay_from_total** to keep the delay with maximum load at port unchanged. But with smaller load at the output port, the delay may actually be smaller than the budgeted maximum delay.

Define drive types by using the command **create_qtm_drive_type**.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to **create_qtm_model** man page. For a more detailed description about QTM, please refer to the *PrimeTime User Guide*.

EXAMPLES

The following command sets the drive of output port OUT1 to be of drive_type drive1.

```
pt_shell> set_qtm_port_drive -type drive 1 OUT1
```

The following command sets the drive of the output port OUT2 to be 5 drive units.

```
pt_shell> set_qtm_port_drive -value 5.0 OUT2
```

SEE ALSO

create_qtm_load_type(2), **create_qtm_model(2)**, **report_qtm_model(2)**,
save_qtm_model(2), **get_qtm_ports(2)**, **set_qtm_port_load(2)**.

set_qtm_port_load

Sets load on Quick Timing Model (QTM) ports.

SYNTAX

```
string set_qtm_port_load [-type load_type] [-factor multiplication_factor] [-value  
load_value] port_list  
string load_type  
float multiplication_factor  
float load_value  
list port_list
```

ARGUMENTS

-type *load_type*
Specifies the global *load_type* parameter.

-factor *multiplication_factor*
Specifies the multiplication factor for the load type to set the load on the QTM port.

-value *load_value*
Specifies the load value in the capacitance units.

port_list
Specifies a list of input or inout ports on which to set the attribute. Each element in the list is either a collection of QTM ports or a pattern that matches QTM ports.

DESCRIPTION

This command sets the load of the input port of the QTM model. There are two methods for setting the load:

1. Use the global *load_type* along with a multiplication factor if you have load types defined.
2. Define the load in terms of the capacitance units.

Define load types using the command **create_qtm_load_type**.

To show the information about the current QTM model, use the **report_qtm_model** command.

For a basic description about QTM, please refer to **create_qtm_model** man page. For a more detailed description about QTM, please refer to the *PrimeTime User Guide*.

EXAMPLES

The following command sets the load on IN1 to be 2 times the global *load_type* load1.

```
pt_shell> set_qtm_port_load -type load1 -factor 2 IN1
```

```
set_qtm_port_load  
1100
```

The following command sets the load on IN2 to be 4.5 capacitance units.

```
pt_shell> set_qtm_port_load -value 4.5 IN2
```

SEE ALSO

```
create_qtm_load_type(2), create_qtm_model(2), report_qtm_model(2),  
save_qtm_model(2), get_qtm_ports(2), set_qtm_port_drive(2).
```

set_qtm_technology

Sets the QTM technology variables.

SYNTAX

```
string set_qtm_technology [-library name]  
[-max_transition trans_value]  
[-min_transition trans_value]  
[-max_capacitance cap_value]  
[-min_capacitance trans_value]  
[-wire_load_model wlm_name]
```

```
string name  
float trans_value  
float cap_value  
string wlm_name
```

ARGUMENTS

- library *name*
Specifies the library name for using library elements to define global parameters. Substitute the string value you want for *name*.
- max_transition *trans_value*
Specifies the value of maximum transition to be used. Substitute the float value you want for *trans_value*.
- min_transition *trans_value*
Specifies the value of minimum transition to be used. Substitute the float value you want for *trans_value*.
- max_capacitance *cap_value*
Specifies the value of max_capacitance to be used. Substitute the float value you want for *cap_value*.
- min_capacitance *cap_value*
Specifies the value of minimum capacitance to be used. Substitute the float value you want for *cap_value*.
- wire_load_model *wlm_name*
Specifies the wire load model used while calculating delays. Substitute the string value you want for *wlm_name*.

DESCRIPTION

This command sets the various QTM technology parameters. The **-library** option is used if you use library cells to define *path_types*, *drive_types*, *load_types*, or if you set the other global parameters in terms of library cells. Before setting the library, the user must be certain that the library is loaded.

If the **wire_load_model** is set, the *path_type* delays are calculated using that

wire_load_model.

To show the information about the current QTM model, use **report_qtm_model** command.

For a basic description about QTM, see the **create_qtm_model** man page. For a more detailed description about QTM, see Chapter 12 of the *PrimeTime User Guide*.

EXAMPLES

The following example sets the library to **my_lib**.

```
pt_shell> read_db my_lib.db \  
#my_lib.db contain the library my_lib
```

```
pt_shell> set_qtm_technology \  
-library my_lib
```

The following command sets the wire_load_model to **wlm1**.

```
pt_shell> set_qtm_technology \  
-wire_load wlm1
```

SEE ALSO

create_qtm_model (2), **report_qtm_model** (2), **save_qtm_model** (2).

set_rail_voltage

Sets power rail voltage on cells.

SYNTAX

```
int set_rail_voltage
[-rail_value rvalue | -rail_list rname_value_list]
[-dynamic_rail_value dynamic_component | -dynamic_rail_list dynamic_component_list]
[-min]
[-max]
cell_list

float rvalue
list rname_value_list
list cell_list
```

ARGUMENTS

- rail_value** *rvalue*
Sets voltage on single-rail cells. If you use this option, you cannot use the **-rail_list** option. These options are mutually exclusive. Replace *rvalue* with a decimal or an integer value.
- rail_list** *rname_value_list*
Sets voltages on individual rails of multi-rail cells. Replace *rname_value_list* with a list, which must have even number of elements. (Odd elements are names of rails; even elements are voltage values.) The rail names must match the definitions of rails in the library power_supply section and in the library operating conditions. If you use this option, you cannot use the **-rail_value** option: These options are mutually exclusive.
- dynamic_rail_value** *dynamic_component*
Specifies what portion of *rvalue* is dynamic. You can only specify this option if the **-rail_value** option is specified. If you attempt to specify this option with the **-rail_list** option, an error message is issued.
- dynamic_rail_list** *dynamic_component_name_value_list*
Specifies the dynamic portion of each rail voltage listed with the **-rail_list** option. The list of dynamic component vlaues must match the entries in the **rname_value_list**.
- min**
Sets rail voltages for the minimum operating condition. If you do not specify either the **-min** or the **-max** option, the tool assumes you are using both the **-min** and **-max** options.
- max**
Sets rail voltages for the maximum operating condition. If you do not specify either the **-min** or the **-max** option, the tool assumes you are using both the **-min** and **-max** options.

`cell_list`

Specifies a list of cells on which to set rail voltages. Replace *cell_list* with the collection of cells you want.

DESCRIPTION

Sets power rail voltage on cells. This command overrides voltages specified in operating conditions. It can be applied to both leaf cells and hierarchical blocks. If used on hierarchical cells, all cells in the block must come from the same library, unless another **set_rail_voltage** or **set_operating_conditions** command has been set on them.

It should be noted that since **-min** and **-max** refer to operating conditions, the **-max** voltage is typically lower than the **-min** voltage.

The voltages are passed to delay equations in the library. Ground rail voltages cannot be changed; the tool assumes zero voltage.

In a typical multi-voltage flow you can set operating conditions on hierarchical blocks corresponding to voltage islands. You can then use a power network analysis tool to calculate the IR drop in power networks; subtract power and ground voltages on cells; and, by using the **set_rail_voltage** command, annotate them on important leaf cells or blocks.

The **-dynamic_rail_value** or **-dynamic_rail_list** can be used to specify the dynamic component of IR drop. In this case, the **-rail_value** or **-rail_list** options can be used to specify the total voltage on a cell and the dynamic options can specify what portion of that voltage is dynamic.

Care should be taken when specifying the dynamic component of minimum rail voltage. In this case, the dynamic component should be negative. This ensures that the static component of rail voltage is not less than the total minimum rail voltage.

If Clock Reconvergence Pessimism Removal (CRPR) is enabled and dynamic rail voltages are specified, the Dynamic CRPR mode is turned on. In this mode only CRPR is computed using clock totals considering the dynamic component of rail voltage if the path is zero-cycle. A zero-cycle path is one in which the same clock edge both launches and captures.

Note that the operating condition (or rail voltage) that is set on the lower levels of the hierarchy (or leaf cell) overrides the operating condition (or rail voltage) on the higher levels of hierarchy.

EXAMPLES

The following example sets the operating conditions named *OC1* and *OC2* on the block named *h1* and then overrides the voltage on the cell named *u3* in *h1*.

```
pt_shell> set_operating_conditions -min OC1 -max OC2 \  
-object_list [get_cells {h1}]  
1
```

```
pt_shell> set_rail_voltage -min -rail_value 1.4 [get_cells {h1/u3}]
```

1

```
pt_shell> set_rail_voltage -max -rail_value 1.1 [get_cells {h1/u3}]
```

1

To specify a dynamic component of 0.2 for the example above, you should use the following commands (Note the negative value for the minimum rail voltage):

```
pt_shell> set_rail_voltage -min -rail_value 1.4 -  
dynamic_rail_value 0.2 [get_cells {h1/u3}]
```

1

```
pt_shell> set_rail_voltage -max -rail_value 1.1 -dynamic_rail_value -  
0.2 [get_cells {h1/u3}]
```

1

SEE ALSO

`remove_rail_voltage` (2), `set_operating_conditions` (2).

set_related_supply_net

Associates an external supply net to the port of the design.

SYNTAX

```
status set_related_supply_net
[power_net]
[-object_list ports]
[-reset]
[-ground ground_net]
[-power power_net]
```

Data Types

<i>ports</i>	list
<i>power_net</i>	upf_supply_net
<i>ground_net</i>	upf_supply_net

ARGUMENTS

-object_list *objects*
Specifies the list of ports to be associated with power/ground nets or reset.
If this option is not specified, all the ports are considered.

-ground *ground_net*
Specifies the ground net.

-power *power_net*
Specifies the power net.

-reset
Resets the related power and ground nets of the ports specified in *object_list*. This can't be used in conjunction with **-power** or **-ground** options

power_net
Specifies the power net. This is deprecated and supported for backward compatibility only. New scripts should specify the power net using **-power** option.

DESCRIPTION

This command is used to associate supply nets with design ports. The tool uses this information for constraining and checking design. The level shifter insertion tool also uses supply nets to determine if level shifter is required between a port and the logic connected to the port.

If this command is not specified, the tool assumes that ports are externally connected to the primary supply net of the domain of the top design.

EXAMPLES

The following example sets the *VDD_EXT* power net and *VSS_EXT* ground net with *INPUT* port.

```
prompt> set_related_supply_net -object_list\  
[get_ports INPUT] -power VDD_EXT -ground VSS_EXT
```

This example resets the power net and ground net on all ports.

```
prompt> set_related_supply_net -reset
```

SEE ALSO

```
create_supply_net(2),  
get_supply_nets(2),  
get_ports(2)
```

set_resistance

Sets the **ba_net_resistance** attribute with a resistance value on specified nets.

SYNTAX

```
int set_resistance [-min] [-max] resistance_value object_list  
float resistance_value  
list object_list
```

ARGUMENTS

-min

Applies only for designs in min-max mode (min and max operating conditions).
Indicates that the *resistance_value* is the minimum resistance.

-max

Applies only for designs in min-max mode (min and max operating conditions).
Indicates that the *resistance_value* is the maximum resistance.

resistance_value

Specifies the resistance value for nets in *object_list*, in units consistent with those used by the technology library.

object_list

A list of nets on which the *resistance_value* is to be set.

DESCRIPTION

Sets the **ba_net_resistance** attribute, which specifies the back-annotation of resistance values on nets in the current design. If the current design is hierarchical, you must link it using the **link** command. The specified *resistance_value* overrides the internally estimated net resistance value and will also override any net resistance set by **read_parasitics**.

You can use the **set_resistance** command also for nets at lower levels of the design hierarchy. You can specify these nets as "BLOCK1/BLOCK2/NET_NAME".

To view resistance values, use the **report_net** command.

To remove a resistance value, use the **remove_resistance** command. To reset all back-annotated resistance values in a design, use the **reset_design** command.

EXAMPLES

The following example sets a resistance of 200 units to nets "a" and "b."

```
pt_shell> set_resistance 200 {a,b}
```

In the following example, a resistance of 300 units is set on net "U1/U2/NET3."

```
pt_shell> set_resistance 300 U1/U2/NET3
```

In the following example, the back-annotated resistance on net "U1/U2/NET3" is removed.

```
pt_shell> remove_resistance {get_nets U1/U2/NET3}
```

SEE ALSO

```
find (2), link (2), remove_resistance (2), report_net (2), report_timing (2),  
reset_design (2), set_drive (2).
```

set_retention

Defines the UPF retention strategy for the power domains in the design.

SYNTAX

```
int set_retention  
retention_strategy  
-domain power_domain  
[-retention_power_net retention_power_net]  
[-retention_ground_net retention_ground_net]  
[-no_retention]  
[-elements objects]
```

Data Types

<i>retention_strategy</i>	string
<i>power_domain</i>	string
<i>retention_power_net</i>	string
<i>retention_ground_net</i>	string
<i>objects</i>	list

ARGUMENTS

retention_strategy
Specifies the UPF retention strategy name. The retention strategy name should be unique within the specified power domain.

-domain *power_domain*
Specifies the power domain name that this UPF retention strategy will be applied to.

-retention_power_net *retention_power_net*
Specifies the retention power net for the retention cells that will be under this UPF retention strategy.

-retention_ground_net *retention_ground_net*
Specifies the retention ground net for the retention cells that will be under this UPF retention strategy.

-no_retention
Specifies that the specified objects in the power domain will not have retention.

-elements *objects*
Specifies the objects that this UPF retention strategy will be applied to. The objects can be hierarchical cells, leaf cells, hdl blocks, nets.

DESCRIPTION

This command defines the UPF retention strategy on the specified power domain under which to map the unmapped sequential cells to retention cells.

PrimeTime uses the `set_retention` command to build virtual power ground connectivity. The command creates explicit connection for retention(backup) power and ground nets to power and ground pins of retention cells. The **-no_retention** on specific elements takes precedence over domain-wide strategy.

If **-elements** option is not specified, then the retention strategy will be applied to all the unmapped sequential cells under the power domain. If **-elements** is specified, the UPF retention strategy will be applied to all the unmapped sequential cells that are under the objects from **-elements**.

EXAMPLES

The following example shows how to define a UPF retention strategy in the specified power domain PD1. Power domain PD1 is defined on instance shutdown_inst.

```
prompt> set_retention retention_1 -domain PD1 \  
-retention_power_net PN1 \  
-retention_ground_net GN1 \
```

In the following example, it shows how to define a UPF retention strategy in the objects under the specified power domain.

```
prompt> set_retention retention_2 -domain PD1 \  
-retention_power_net PN1 \  
-retention_ground_net GN1 \  
-elements shutdown_inst/mid_inst \
```

SEE ALSO

help UPF,
report_power_pin_info(2).

set_retention_control

Defines the UPF retention control signals for the defined UPF retention strategy.

SYNTAX

```
int set_retention_control  
  retention_strategy  
  -domain power_domain  
  -save_signal {save_signal save_sense}  
  -restore_signal {restore_signal restore_sense}
```

Data Types

```
retention_strategy    string  
power_domain          string  
save_signal save_sense  string high|low  
restore_signal restore_sense  string high|low
```

ARGUMENTS

```
retention_strategy  
    Specifies the UPF retention strategy name. The retention strategy should have  
    already been defined using set_retention command  
  
-domain power_domain  
    Specifies the power domain name that this UPF retention strategy that is  
    applied to.  
  
-save_signal save_signal_net_sense  
    Specifies the save signal net and the save signal sense for the retention  
    cells under this retention strategy.  
  
-restore_signal_net retention_ground_net  
    Specifies the restore signal net and the restore sense for the retention cells  
    under this retention strategy.
```

DESCRIPTION

This command defines the UPF retention control signals and control signal sense for this retention strategy of this power domain. The specified control signals are applied to the retention cells under the associated retention strategy.

PrimeTime uses additional information about the retention strategy provided by this command to properly match retention strategies to individual leaf cells. Retention strategy is used for deriving PG connectivity from UPF description. When multiple retention strategies match a retention cell then the last one entered is used.

EXAMPLES

The following example shows how to define the retention control signals on the

defined UPF retention strategy *retention_1* of the power domain *PD1*.

```
prompt> set_retention_control retention_1 \  
-domain PD1 \  
-save_signal {save_net high} \  
-restore_signal {restore_net low} \  

```

The following example shows how to find which retention strategy was used on which retention register to derive backup PG connectivity:

```
foreach_in_collection c [sort_collection [get_cells -hier * -  
filter "upf_retention_strategy != {}"] full_name] {  
    echo [format { Cell %-  
8s retention %s} [get_attribute $c full_name] [get_attribute $c upf_retention_stra  
tegy]]  
}
```

SEE ALSO

set_retention.2

set_rtl_to_gate_name

Sets the mapping between RTL and gate-level objects. This mapping is used if the user is reading RTL backward SAIF file or RTL VCD file for power estimation.

SYNTAX

```
int set_rtl_to_gate_name
    -rtl rtl_name
    -gate gate_name
    -substitute {from_string to_string}
    -inverted
```

```
string rtl_name
string gate_name
```

ARGUMENTS

-rtl *rtl_name*

This option provides the name of a RTL object, whose name have changed after synthesis. These RTL object appears in the backward RTL SAIF file or RTL VCD file.

-gate *gate_name*

This option provides the name of gate-level object, for the corresponding RTL object. RTL object name in backward RTL SAIF file or RTL VCD file may change after synthesis. The *gate_name* is the new gate-level name of the RTL object after synthesis.

-inverted

Specifies that the signal in the RTL activity file should be inverted when mapped to the gate-level design. This only applies when the object specified with the **-fi-gate** option is a net or a pin.

-substitute {*from_string to_string*}

This option asks the tool to substitute all *from_string* occurrences in RTL VCD or RTL SAIF file with *to_string*.

DESCRIPTION

If a user has done only the RTL simulation for generating backward SAIF file or RTL VCD, this command should be used for setting the mapping between RTL and gate-level object names. The names of the RTL objects, may change after synthesis. Therefore, **read_saif** or **read_vcd** command will not be able to map the names present in the RTL SAIF/VCD file to the gate-level objects. This may result in a lot of nets/ports/pins not having user asserted switching activity values, which will result in inaccurate power numbers. In order to avoid this situation **set_rtl_to_gate_command** allows the user to set mapping between RTL and gate-level names. The user can set the RTL and gate-level names by using **-rtl** and **-gate** options, respectively. The **-rtl *rtl_name*** option provides the RTL name present in backward RTL SAIF file or RTL VCD file, and **-gate *gate_name*** option provides the new gate-level name after synthesis. During **read_saif** and **read_vcd** command, if tool is not able to find the gate-level object

from the RTL name, it will look for the corresponding gate-level name, if it is provided by this command, and try to find the gate-level object. Note that `gate_name` should be the full name of the object with respect to the current instance. Similarly, `rtl_name` should be the full name as it appears in the backward RTL SAIF file or RTL VCD file.

If the `-substitute` option is used, any signal name in VCD or SAIF is only scanned once. The substitutions are not applied recursively.

EXAMPLES

The following example provides the mapping between RTL and gate-level objects.

```
pt_shell> set_rtl_to_gate_name -rtl reg_i[1] -gate reg_i_1
```

The following example provides the mapping for an RTL level signal to a gate-level pin, and instructs the tool to invert the signal found in the RTL activity file.

```
pt_shell> set_rtl_to_gate_name -rtl test_signal
                        -gate test_signal_reg/Q -inverted
```

SEE ALSO

```
unset_rtl_to_gate_name(2),
report_switching_activity (2),
read_saif(2),
reset_switching_activity(2),
set_switching_activity(2),
get_switching_activity(2),
report_power(2).
```

set_scope

Specify the current UPF scope. Return the current UPF scope prior to the execution of this command as a full path string relative to the current design top if successful and null string if it fails.

SYNTAX

```
string set_scope  
[instance]
```

Data Types

instance string

ARGUMENTS

instance
Specifies the working instance in pt_shell. If *instance* is not specified, the focus returns to the top level of the current design. If *instance* is ".", pt_shell returns to the working instance. If *instance* is "..", the context is moved up one level in the instance hierarchy. If *instance* begins with "/", pt_shell returns to the working instance of the design whose name is after the "/". More complex examples of *instance* arguments are described below in EXAMPLES.

DESCRIPTION

Sets the UPF scope in pt_shell. Functionally, this command is the subset of current_instance, but with different return value.

- If no *instance* argument is specified, the UPF scope of pt_shell is returned to the top level of the hierarchy.
- If *instance* is ".", the UPF scope is returned and no change is made.
- If *instance* is "..", the UPF scope is moved up one level in the design hierarchy.
- If *instance* is a valid cell at the current level of hierarchy, the UPF scope is moved down to that level of the design hierarchy.
- Multiple levels of hierarchy can be traversed in a single call to **set_scope** by separating multiple cell names with slashes.

For example, **set_scope U1/U2** sets the UPF scope down two levels of hierarchy if both cells exist at the current levels in the design hierarchy.

- The ".." directive can also be nested in complex *instance* arguments.

For example, the command **set_scope** "../..MY_INST" attempts to move the context up two levels of hierarchy, then down one level to the "MY_INST" cell.

NOTE: The **set_scope** command does not work on leaf cells. If you attempt to run **set_scope** on a leaf cell, an error will occur.

SEE ALSO

current_instance (2),
current_design (2).

set_setup_hold_pessimism_reduction

Set the optimization constraints for setup-hold pessimism reduction.

SYNTAX

set_setup_hold_pessimism_reduction

```
-mode SHPR mode  
[-setup_cutoff setup_cutoff_slack]  
[-hold_cutoff hold_cutoff_slack]  
float setup_cutoff_slack  
float hold_cutoff_slack
```

ARGUMENTS

- mode *mode_name***
Specifies the mode of setup_hold_pessimism_reduction. The valid names are: *total*, *setup* or *hold*.
- setup_cutoff *setup_cutoff_slack***
Specifies the cutoff slack of MAX path for SHPR optimization. Must be a float value. The default value is negative INFINITY.
- hold_cutoff *hold_cutoff_slack***
Specifies the cutoff slack of MIN path for SHPR optimization. Must be a float value. The default value is negative INFINITY.

DESCRIPTION

This command sets the optimization constraints for setup-hold pessimism reduction (SHPR). The valid modes are: *total*, *setup* and *hold*. The different modes control how setup slack can be traded off for hold slack or vice versa. In *setup* mode (also known as setup-preferred mode), the goal is to prefer positive setup slack at the expense of hold slack. This mode may be useful for performance-critical designs, knowing that the hold slacks can be improved later through strategic buffer insertion. In *hold* mode (also known as hold-preferred mode), the goal is to prefer positive hold slack at the expense of setup slack. This mode may be useful in the late stages of timing closure, when it is desirable to see if small hold violations can be resolved without any design changes. The *total* mode trades off setup and hold slack in a way which minimizes the sum of the negative rise/fall setup/hold slack. This mode results in the smallest overall set of setup and hold timing violations. This is the default mode of SHPR.

Of course, it may not be desirable to introduce a large violation in the sacrificial slack type to gain only a small improvement in the preferred slack type. To keep the tradeoff reasonable, the *-setup_cutoff* and *-hold_cutoff* options have been provided. In setup-preferred mode, hold violations will be made no worse than the *-hold_cutoff* value to improve setup. In hold-preferred mode, setup violations will be made no worse than the *-setup_cutoff* value to improve hold. The default value for both cutoff options is negative infinity, which allows any amount of slack worsening in the sacrificial slack to improve the preferred slack.

In both modes, the preferred slack type will be improved only as far as it takes to reach zero slack, and no further.

The slacks being analyzed above are the slacks of timing paths ending at sequential cell data pins which have SHPR constraint information.

EXAMPLES

The following example sets the optimization mode to *total* while setting the cutoff_slack of setup_cutoff and hold_cutoff to 0 and -0.03ns, respectively.

```
pt_shell> set_setup_hold_pessimism_reduction -mode total \  
                                         -setup_cutoff 0 -hold_cutoff -0.03  
1
```

The following example sets the optimization mode to *setup* while setting the cutoff_slack of hold_cutoff to 0.03ns, respectively.

```
pt_shell> set_setup_hold_pessimism_reduction -mode setup \  
                                         -hold_cutoff 0.03  
1
```

The following example sets the optimization mode to *hold* while setting the cutoff_slack of setup_cutoff to -0.05ns, respectively.

```
pt_shell> set_setup_hold_pessimism_reduction -mode hold \  
                                         -setup_cutoff 0.03  
1
```

SEE ALSO

remove_setup_hold_pessimism_reduction (2),

set_si_aggressor_exclusion

Sets the given nets to be exclusive while switching in the given direction, when they are aggressors to the same victim net.

SYNTAX

```
int set_si_aggressor_exclusion  
[-number_of_active_aggressors n]  
[-rise]  
[-fall]  
anets
```

```
list    anets
```

ARGUMENTS

-number_of_active_aggressors *n*

Specifies the maximum number of aggressors among the given list of exclusive aggressors that can be active at a given instant. This option can be used to set *n*-hot or *n*-cold behavior, where only *n* nets can be active(hot) or quiet(cold) at a given instant. If this option is not specified, the default value of *n*=1 would be used. i.e only one of the given exclusive aggressors would be considered to be active at a given time.

-rise

Specifies the aggressor nets to be exclusive while switching in the rise direction. If neither the *-rise* nor the *-fall* options are specified, the aggressor nets *anets* are considered to be exclusive in both *-rise* and *-fall* directions.

-fall

Specifies the aggressor nets to be exclusive while switching in the fall direction. If neither the *-rise* nor the *-fall* options are specified, the aggressor nets *anets* are considered to be exclusive in both *-rise* and *-fall* directions.

anets

Specifies the list of nets which are exclusive when they are aggressors to the same victim net.

DESCRIPTION

The **set_si_aggressor_exclusion** sets the aggressor nets to be exclusive while switching in the given direction at the same time. Among all combinations of active aggressors allowed by the *-number_of_active_aggressors*, only the combination which produces the worst alignment would be considered for crosstalk analysis.

Only the aggressor nets specified in the list *anets* are considered to be exclusive. If a victim net is specified in the list of *anets*, it is not considered to be exclusive with any of its aggressor nets.

To view which aggressor nets were considered to be active and which aggressor nets were set to be quiet, use the **report_delay_calculation** or **report_noise_calculation** commands on the victim net. The quiet aggressor nets are reported to be screened due to aggressor exclusion.

These exclusive commands are independent of parasitics, so they can be applied even before reading in parasitics.

Note that application of exclusive aggressors cannot be done incrementally, next `update_timing` and `update_noise` would be a full update.

The command **report_si_aggressor_exclusion** can be used to check which groups have been set by command **set_si_aggressor_exclusion**.

EXAMPLES

The following example shows how the nets *SCAN_LOGIC** can be set to be exclusive while switching in the rise direction.

```
pt_shell> set_si_aggressor_exclusion [get_nets SCAN_LOGIC*] -rise
1
```

The following example shows how the nets *LCO_LOGIC** can be set to be exclusive while switching in the fall direction.

```
pt_shell> set_si_aggressor_exclusion [get_nets LCO_LOGIC*] -fall
1
```

The following example shows how only 3 of the nets *LOGIC_BUS** can be considered to be active while switching in the rise direction.

```
pt_shell> set_si_aggressor_exclusion [get_nets LOGIC_BUS*] -rise
-number_of_active_aggressors 3
1
```

SEE ALSO

```
remove_si_aggressor_exclusion (2), report_si_aggressor_exclusion (2),
report_delay_calculation (2), report_noise_calculation (2),
si_analysis_logical_correlation_mode (3), set_si_delay_analysis (2),
set_si_noise_analysis (2),
```

set_si_delay_analysis

Sets coupling information on nets for crosstalk analysis.

SYNTAX

```
int set_si_delay_analysis  
[-reselect rnets]  
[-ignore_arrival inets]  
[-exclude]  
[-victims vnets]  
[-aggressors anets]  
[-rise]  
[-fall]  
[-min]  
[-max]
```

```
list rnets  
list inets  
list vnets  
list anets
```

ARGUMENTS

-reselect *rnets*

Specifies a list of nets to be reselected in each iteration, independent of reselection criteria. A net cannot be reselected if it is filtered out; if this is attempted, the XTALK-106 message comes up during the update_timing. You cannot use this option with the **-ignore_arrival**, **-exclude**, **-victims**, or **-aggressors** options. If it applied on a noncoupled net, it is ignored.

-ignore_arrival *inets*

Specifies a list of nets to be analyzed as infinite window. You cannot use this option with the **-reselect**, **-exclude**, **-victims**, or **-aggressors** options.

-exclude

Indicates that nets specified as *vnets* or *anets*) are to be excluded from the crosstalk analysis as victim nets or aggressor nets, respectively. You cannot use this option with the **-reselect** or **-ignore_arrival** option. When both **-victims vnets** and **-aggressors anets** are applied, all cross capacitances between **vnets** and **anets** are excluded, when **vnets** are victims and **anets** are aggressors.

-victims *vnets*

Specifies the list of nets on which **-exclude** information is applied as a victim. You cannot use this option with the **-reselect** or **-ignore_arrival** option. If you use the **-victims** option, you must use the **-exclude** option. When used with the **-aggressors** option, **-victims** excludes the cross capacitances between the victim nets (*vnets*) and the aggressor nets (*anets*).

-aggressors *anets*

The list of nets on which **-exclude** option information is applied as an aggressor. You cannot use this option with the **-reselect** or **-ignore_arrival**

option. If you use the **-aggressors** option, you must use the **-exclude** option. When used with the **-victims** option, **-aggressors** excludes the cross capacitances between the victim nets (*vnets*) and the aggressor nets (*anets*).

-rise

Excludes a list of nets for victim rising. If you use the **-rise** option, you must use the **-exclude** option.

-fall

Excludes a list of nets for victim falling. If you use the **-fall** option, you must use the **-exclude** option.

-min

Excludes a list of nets for min path analysis. If you use the **-min** option, you must use the **-exclude** option.

-max

Excludes a list of nets for max path analysis. If you use the **-max** option, you must use the **-exclude** option.

DESCRIPTION

Sets coupling information on nets for crosstalk analysis.

The **set_si_delay_analysis** command allows you to override default reselection criteria net by net basis. It also allows you to set some net as infinite window as both aggressor and victim. This command has no default. To remove the result of this command, use the **remove_si_delay_analysis** command.

The **set_si_delay_analysis** command **-exclude** option allows you to select nets to be excluded from crosstalk analysis. By default, all nets with coupling capacitances (non-filtered) are included in crosstalk analysis as both victim and aggressor. This command allows the nets to be excluded as victim (by using the **-victims** option) and/or aggressor (by using the **-aggressors** option). When you use both the **-victims** and **-aggressors** options, the command excludes the relationship between the specified victim nets (*vnets*) and aggressor nets (*anets*); this is referred to as "pair-wise exclusion."

Using the **set_si_delay_analysis** command with the **-reselect** option forces the *rnets* to be reselected for all the crosstalk iterations independent of the reselection criteria. During the crosstalk iterations the command prints in a tabular format the count of all nets reselected due to using this command. When this is applied to a net, if the net is not completely filtered (and screened) out in the first iteration, that net is reselected for all the following iterations. It is important to note that it overrides only reselection criteria, not the filtering and screening process. If, due to filtering (or screening), the net cannot be reselected, you receive the informational **XTALK-106** message regarding this situation.

By using the **set_si_delay_analysis** command with the **-ignore_arrival** option, you can set the *inets* as infinite window. When *inets* is analyzed as victim, the arrival time of all the nets in this coupling cluster is ignored. When *inets* is analyzed as aggressor the arrival time of *inets* is ignored. The infinite window analysis could make the crosstalk calculation more pessimistic.

Since users can set and remove any options individually, the priorities of options are decided during `update_timing`. For example if you reselect a net and later on, also exclude the net as a victim, during `update_timing`, the net is excluded as a victim and no calculation is done. Information message **XTALK-106** would be issued during `update_timing`. If you set `ignore_arrival` on a net and then reselect it, both options are respected. The net will be reselected and in all the crosstalk iterations, the arrival windows for this net would be ignored. You can use the command **report_si_delay_analysis** at any time to check the various options that were set on the nets or in the design.

The **set_si_delay_analysis** command returns a **1** if successful and a **0** if unsuccessful.

To view the results of this command, use the **report_si_delay_analysis** command.

EXAMPLES

The following example shows that all nets described by `CLK_NET_*` are excluded from crosstalk analysis as victim nets.

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets CLK_NET*]  
1
```

The following example shows that all critical nets described by `SCAN_LOGIC*` are reselected in all cross talk iterations.

```
pt_shell> set_si_delay_analysis -reselect [get_nets SCAN_LOGIC*]  
1
```

The following example shows how all nets in a design to be analyzed as infinite window.

```
pt_shell> set_si_delay_analysis -ignore_arrival [get_nets -hier *]  
1
```

The following example shows how to ensure that the scan clock described by `SCN_CLK_*` is not effecting the main clock network described by `CLK_NET_*`, and vice versa.

```
pt_shell> set_si_delay_analysis -exclude -victims \  
[get_nets SCN_CLK_*] \  
-aggressors [get_nets CLK_NET*]  
pt_shell> set_si_delay_analysis -exclude -aggressors \  
[get_nets SCN_CLK_*] -victims [get_nets CLK_NET*]  
1
```

SEE ALSO

```
read_parasitics (2), remove_si_delay_analysis (2); report_si_delay_analysis (2);  
si_enable_analysis (3), si_xtalk_reselect_delta_delay (3),  
si_xtalk_reselect_delta_delay_ratio (3), si_xtalk_reselect_max_mode_slack (3),  
si_xtalk_reselect_min_mode_slack (3).
```

set_si_delay_disable_statistical

Disables composite aggressor statistical analysis on nets for crosstalk analysis.

SYNTAX

```
int set_si_delay_disable_statistical  
dnets
```

```
list dnets
```

ARGUMENTS

dnets

A list of nets in the current design for which the composite aggressor statistical analysis is disabled.

DESCRIPTION

If selected in composite aggressor group for crosstalk analysis, *dnets* specified by this command are not treated statistically. If they are not selected into composite aggressor group, this command has no effect.

To remove the result of this command, use the **remove_si_delay_disable_statistical** command.

EXAMPLES

The following example shows how to disable net from statistical analysis when considered as a composite aggressor.

```
pt_shell> set_si_delay_disable_statistical [get_nets LOGIC1]  
1
```

SEE ALSO

remove_si_delay_disable_statistical (2) **report_si_delay_analysis** (2)

set_si_noise_analysis

Sets coupling information on nets for noise analysis.

SYNTAX

```
int set_si_noise_analysis  
[-ignore_arrival inets]  
[-exclude]  
[-victims vnets]  
[-aggressors anets]  
[-above]  
[-below]  
[-low]  
[-high]
```

```
list inets  
list vnets  
list anets
```

ARGUMENTS

-ignore_arrival *inets*

Specifies a list of nets to be set as infinite window. When *inets* are analyzed as victims, all of their aggressors are set with infinite window. When *inets* are analyzed as aggressors, they are set as aggressors with arrival time ignored. You cannot use this option with the **-exclude**, **-victims**, **-aggressors** or **-high**, **-low**, **-above**, **-below** options.

-exclude

Indicates that nets specified as *vnets* or *anets*) are to be excluded from the noise analysis as victim nets or aggressor nets, respectively. You cannot use this option with the **-ignore_arrival** option. When both **-victims vnets** and **-aggressors anets** are applied, the noise between **vnets** and **anets** is not analyzed, when **vnets** are victims and **anets** are aggressors.

-victims *vnets*

Specifies the list of nets on which **-exclude** information is applied as a victim. You cannot use this option with the **-ignore_arrival** option. If you use the **-victims** option, you must use the **-exclude** option. When used with the **-aggressors** option, **-victims** excludes the noise analysis between the victim nets (*vnets*) and the aggressor nets (*anets*).

-aggressors *anets*

The list of nets on which **-exclude** option information is applied as an aggressor. You cannot use this option with the **-ignore_arrival** option. If you use the **-aggressors** option, you must use the **-exclude** option. When used with the **-victims** option, **-aggressors** excludes the noise analysis between the victim nets (*vnets*) and the aggressor nets (*anets*).

-above

Excludes a list of nets for victim above rail. If you use the **-above** option, you must use the **-exclude** option.

-below

Excludes a list of nets for victim below rail. If you use the **-below** option, you must use the **-exclude** option.

-low

Excludes a list of nets for low rail noise analysis. If you use the **-low** option, you must use the **-exclude** option.

-high

Excludes a list of nets for high rail noise analysis. If you use the **-high** option, you must use the **-exclude** option.

DESCRIPTION

Sets coupling information on nets for noise analysis.

The **set_si_noise_analysis** command allows you to exclude some net from noise analysis or set some net as infinite window as aggressor. This command has no default. To remove the result of this command, use the **remove_si_noise_analysis** command.

The **set_si_noise_analysis** command **-exclude** option allows you to select nets to be excluded from noise analysis. By default, all nets with coupling capacitances (non-filtered) are included in noise analysis as both victim and aggressor. This command allows the nets to be excluded as victim (by using the **-victim** option) and/or aggressor (by using the **-aggressors** option). When you use both the **-victims** and **-aggressors** options, the command excludes the noise analysis between the specified victim nets (*vnets*) and aggressor nets (*anets*); this is referred to as "pair-wise exclusion."

By using the **set_si_noise_analysis** command with the **-ignore_arrival** option, you can set the *inets* as infinite window as aggressor if they are analyzed as aggressors. When *inets* are analyzed as noise victims, all of their aggressors will be infinite window.

The **set_si_noise_analysis** command returns a **1** if successful and a **0** if unsuccessful.

To view the results of this command, use the **report_si_noise_analysis** command.

EXAMPLES

The following example shows that all nets described by *CLK_NET_** are excluded from noise analysis as victim nets.

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets CLK_NET*]  
1
```

The following example shows that when all nets described by *SOME_LOGIC** are analyzed as aggressors, they are set as infinite window as aggressors in noise analysis.

```
pt_shell> set_si_noise_analysis -ignore_arrival [get_nets SOME_LOGIC*]  
1
```

The following example shows how to ensure that the scan clock described by *SCN_CLK_**

is not effecting the main clock network described by *CLK_NET_**, and vice versa.

```
pt_shell> set_si_noise_analysis -exclude -victims \  
[get_nets SCN_CLK_*] \  
-aggressors [get_nets CLK_NET*]  
pt_shell> set_si_noise_analysis -exclude -aggressors \  
[get_nets SCN_CLK_*] -victims [get_nets CLK_NET*]  
1
```

SEE ALSO

```
read_parasitics (2), remove_si_noise_analysis (2); report_si_noise_analysis (2);  
si_enable_analysis (3).
```

set_si_noise_disable_statistical

Disables composite aggressor statistical analysis on nets for noise analysis.

SYNTAX

```
int set_si_noise_disable_statistical  
dnets
```

```
list dnets
```

ARGUMENTS

dnets

A list of nets in the current design for which the composite aggressor statistical analysis is disabled.

DESCRIPTION

If selected in composite aggressor group for noise analysis, *dnets* specified by this command are not treated statistically. If they are not selected into composite aggressor group, this command has no effect.

To remove the result of this command, use the **remove_si_noise_disable_statistical** command.

EXAMPLES

The following example shows how to disable net from statistical analysis when considered as a composite aggressor.

```
pt_shell> set_si_noise_disable_statistical [get_nets LOGIC1]  
1
```

SEE ALSO

remove_si_noise_disable_statistical (2) **report_si_noise_analysis** (2)

set_steady_state_resistance

Sets steady-state resistance for a library pin or port.

SYNTAX

```
int set_steady_state_resistance  
[-above]  
[-below]  
[-low]  
[-high]  
res_value  
object_list
```

```
float res_value  
list object_list
```

ARGUMENTS

-above
Specifies steady state resistance for above ground or power rail noise analysis region.

-below
Specifies steady state resistance for below ground or power rail noise analysis region.

-low
Specifies steady state resistance for ground rail noise.

-high
Specifies steady state resistance for power rail noise.

res_value
Specifies the steady state resistance value in units of resistance. 1.0.

object_list
Specifies a list of lib-pins or ports.

DESCRIPTION

A steady-state driver of a net affects the size of crosstalk bumps on the net due to its loading effects on the net. PrimeTime SI needs the steady-state I-V characteristics of the driver output in order to accurately calculate the characteristics of crosstalk noise bumps.

There exist four operating regions where noise bumps can occur: below low, above low, below high, and above high. Each region may have its own steady state resistance value.

This command can be used in the absence of library-specified I-V characteristics, or to override the library-specified characteristics by replacing them with a steady-

state resistance value.

report_noise_calculation will show whether noise immunity information was taken from the library or annotated by this command.

EXAMPLES

This example specifies a steady state resistance of 2.0 for above the ground rail noise for pin Z of library cell AN4 in the lsi_10k library:

```
pt_shell> set_steady_state_resistance -above -low 2.0 lsi_10k/AN4/Z
```

SEE ALSO

report_noise_calculation (2), **remove_steady_state_resistance** (2).

set_supply_net_probability

Sets the static probability annotation on selected supply nets. This probability affects average power analysis.

SYNTAX

```
int set_supply_net_probability  
supply_nets  
[static_prob]  
[-remove]  
list supply_nets  
float static_prob
```

ARGUMENTS

supply_nets
Specifies the supply net or supply nets to apply the probability to. This argument can be a supply net name, a list of supply net names, a supply net (obtained using the **get_supply_nets** command) or a collection of supply nets (obtained using the **get_suppl_nets** command).

static_prob
The probability that the supply net is powered. The value should be between 0 and 1.
This argument is optional. If not supplied by the user, no annotation is performed and a description of the static probability on the supply nets is returned.

-remove
With this option, the annotation on the given supply nets is removed.

DESCRIPTION

Average power analysis uses the supply net probability to determine what fraction of the time the cells connected to this supply net are powered.

EXAMPLES

Here is an example of the command in use:

```
pt_shell> get_supply_nets *  
{ "VDDI", "VDDM", "VDDIS", "VDD", "VDDMS", "VDDGS", "VDDG", "VSS" }  
pt_shell> set_supply_net_probability VDDI 0.25  
{ "VDDI" 0.25 annotated }  
pt_shell> set_supply_net_probability [list VDDI VDD] 0.4  
{ "VDDI" 0.4 annotated } { "VDD" 0.4 annotated }  
pt_shell> set_supply_net_probability [get_supply_nets *] 0.4  
{ "VDDI" 0.4 annotated } { "VDD" 0.4 annotated } ...
```

```
pt_shell> set_supply_net_probability [get_supply_nets VDDI] 0.4  
{ "VDDI" 0.4 annotated }
```

SEE ALSO

```
create_supply_net(2)  
update_power(2)
```

set_switching_activity

Sets switching activity annotation on selected nets, pins, ports, and cells of the current design.

SYNTAX

```
int set_switching_activity
    [-static_probability value]
    [-toggle_count count]
    [-clock_derate value]
    [-glitch_count count]
    [-state_condition state]
    [-path_sources name_list]
    [-rise_ratio ratio_value]
    [-period period_value]
    [-base_clock clock]
    [-type object_type_list]
    [-no_hierarchy]
    [object_list]
    [-clocks clocks]
    [-quiet]
```

```
float value
float count
string state
list name_list
float ratio_value
float period_value
string clock
list object_type_list
list object_list
list clocks
```

ARGUMENTS

-static_probability *sp_value*
Specifies the value of the **static_probability** switching activity. *sp_value* represents the percentage of the time the signal is at the logic state 1. For example, a value of 0.25 indicates that the signal is in the logic state 1 for 25% of the time.

-toggle_count *count*
Specifies the value of the toggle rate switching activity. The count is a floating point number that represent the number of 0->1 and 1->0 glitch free transitions, that the signal makes during a period of time. You can specify the period with the **-period** option. Alternatively, you can annotate a related clock using the **-base_clock** argument, and the specified toggle rate is relative to the related clock period. If you specify the **-clocks** option, the related clock is chosen based on which clock domain the object belongs. If belonging to multiple clock domains, the faster clock is the related clock. If no clock domain is found for the object, choose the fastest clock in the design as the related clock.

-clock_derate *value*

An alternative to **-toggle_count** if **-base_clock** or **-clocks** is specified. The annotated toggle rate for a certain object is a factor of the toggle rate of the related clock of the object. The related clock is chosen based on which clock domain the object belongs. If belonging to multiple clock domains, the faster clock will be the related clock. If no clock domain is found for the object, choose the fastest clock in the design as the related clock.

-glitch_count *count*

Specifies the value of the glitch rate switching activity. The count is a floating point number that represent the number of 0->1 and 1->0 glitch transitions, that the signal makes during a period of time. You can specify the period with the **-period** option. Alternatively, you can annotate a related clock using the **-base_clock** argument, and the specified glitch rate is relative to the related clock period. You cannot use this option with the **-clocks** option.

Note that if only one of **-toggle_count** or **-glitch_count** options are specified, the other value is assumed to be zero.

-state_condition *state*

Specifies the state condition when annotating state-dependent toggle rate and glitch rate on pins or state-dependent static probabilities on cells. State dependent toggle rate and glitch rate can be annotated when the internal power of the library cell pin is characterized with state-dependent power tables. State-dependent static probabilities can be annotated when the cell leakage power is characterized with state-dependent power tables. The state condition specified with this argument must be logically equivalent to a state condition in the internal / leakage power characterization. The state condition should be enclosed between " ". Moreover, if the user wants to set switching activity information for the default state condition then the argument of **-state_condition** option should be "default". This option cannot be used with **-clocks**.

-path_sources *name_list*

Specifies the path sources when annotating path-dependent toggle rate and glitch rate on pins. This is used when the library cell pin has path-dependent internal power. The path source(s) specified with this argument must be the same as those in the internal power characterization. When listing more than one pin in path sources the pin names should be separated by a space and enclosed between " ". For example, if the pins in path sources are A and B, the argument to option **-path_sources** will look like "A B". Please note that if *name_list* is given using **-path_sources** argument, then *state* **should also be provided using -state_condition** condition. This option cannot be used with **-clocks**.

-rise_ratio *ratio_value*

Specifies the ratio of rise transitions to total transitions for the specified toggle rate and glitch rate when annotating pins that are characterized with both rise and fall internal power. The *ratio_value* argument is a floating point number between 0.0 (all transitions are falling) and 1.0 (all transitions are rising). You need to specify a toggle rate and glitch rate in order to use this option. The default value is 0.5.

-period *period_value*

Specifies the time period for which the number of transitions given in the

toggle_count and glitch_count occur; The time units for this value are those specified in the target technology library. When this argument is used, the values of toggle and glitch counts specified by the **-toggle_count** and by the **-glitch_count** options are divided by the given *period_value*; The resulting toggle rate and glitch rate are then annotated to the object(s) given to the set_switching_activity command. If the **-period** argument is not specified, a default *period_value* of 1.0 is assumed.

-base_clock *clock*

Specifies a clock to which toggle and glitch count values are related to. This clock is referred to as the object's related clock. When this argument is used, the values of toggle and glitch counts specified by the **-toggle_count** and by the **-glitch_count** options are divided by the clock period. The resulting toggle rate and glitch rate are then annotated to the objects given to the set_switching_activity command.

-type *object_type_list*

Specifies a list of object types that are used for implicitly selecting the objects that are annotated with the specified switching activity. The *object_type_list* argument can be a list of one or more of the following object types: **registers** (sequential cell outputs), **three_states** (tristate cell outputs), **inputs** (input design ports / hierarchical instance pins), **outputs** (output design ports / hierarchical instance pins), **inout** (inout design ports / hierarchical instance pins), **ports** (design port / hierarchical instance pins), **nets** (nets). **clock_gating_cells** (clock gating cell outputs) **black_boxes** (black box outputs) When the **-type** argument is used all the objects in the current instance (current design, if no current instance is set) that satisfy the selection criteria are annotated with the specified switching activity. If the **registers** selection type is used, both the noninverting (Q) and inverting (QN) sequential cell outputs are annotated. The annotated toggle rate and glitch rate on the inverting outputs is the same as that on the noninverting outputs. The static probability on the inverting outputs is 1.0 - *value* where *value* is the specified static probability.

If the type used is **registers**, sequential cell outputs are annotated. As of the 2007.12 release, clock gating cells are excluded from this group.

If the **-clocks** option is also specified, the selected objects are further constrained to fall into the specified clock domains. However, the source pins of the clock are not included.

-no_hierarchy

Used with the **-type** argument and specifies that top level objects in the current instance that satisfy the selection criteria is annotated. If not specified, the objects in all the hierarchies in the current instance that satisfy the selection criteria are considered.

object_list

Specifies a list of nets, pins, ports, or instances in the current design on which the **static_probability**, **toggle_count**, and **glitch_count** switching activity values are to be set. Used with the **-type** argument and specifies that all the objects in the given list of instances that satisfy the selection criteria are annotated. This option can also be used with or without the **-no_hierarchy** option.

-clocks *clocks*

Specifies that the specified or selected objects must belong to the given clock domains. The related clock for toggle rate are chosen based on which clock domain an object belongs to. If belonging to multiple clock domains, the faster clock is the related clock. If no clock domain is found for the object, choose the fastest clock in the design as the related clock.

When this option is used, nets connected to the source pins of the specified clocks will not have their activities set; only other objects in the clock domain can be set. By default, the activity of the source pin of the clock is related to the clock period. To override this default for the activity of the source pins of the clock, use `set_switching_activity` and specify the source pin without using the `-clocks` option.

-quiet

Specifies quiet mode and for instance suppresses warnings on design objects that could not be annotated on the design.

DESCRIPTION

You can use this command to annotate design nets, ports, pins, and cells with the different kinds of switching activity. These include simple toggle count, glitch count and static probability on nets, ports and pins; state and path dependent toggle count and glitch count on cell pins, and state dependent static probabilities on cells.

Toggle counts, glitch counts, and static probabilities are annotated using the **-toggle_count**, **-glitch_count** and **-static_probability** arguments respectively. The toggle count, glitch count, and static probability can be made state dependent by specifying the state condition with the **-state_condition** argument. The toggle count and glitch count can be made path dependent by specifying the path sources with the **-path_sources** argument. You can specify a related clock using the **-base_clock** argument. The annotated toggle count and glitch count are divided by the related clock period and the resultant toggle rate and glitch rate values are annotated on the design objects.

The design objects that are annotated with switching activity can be specified explicitly as a list of objects. The objects can also be specified implicitly by using the **-type**, and *object_list* arguments together.

If the objects is specified implicitly by the `-type`, the selection can be clock domain related. The selected objects must fall into the specified clock domains. The toggle rate will be calculated regarding to the related clocks of the objects. The related clock is chosen based on which clock domain the object belongs. If belonging to multiple clock domains, the faster clock will be the related clock. If no clock domain is found for the object, choose the fastest clock in the design as the related clock.

For statistics on the switching activity annotation on the current design use the **report_switching_activity** command.

EXAMPLES

The following **set_switching_activity** command annotates a simple toggle rate value of

`set_switching_activity`

33 / 1320 = 0.025, glitch rate value of 10 / 1320 = 0.0075 and a static probability value of 0.015 to all design input ports.

```
pt_shell> set_switching_activity -toggle_count 33 -glitch_count 10 -period
1320 -static_probability 0.015 [all_inputs]
```

The following example annotates a toggle rate value of 0.25, a glitch rate value of 0.05 and a static probability value of 0.015 to all design input ports. The clock CLK is specified as the related clock for all inputs.

```
pt_shell> create_clock CLK -period 10
pt_shell> set_switching_activity -toggle_count 0.25 -glitch_count 0.05 -
clock CLK -static_probability 0.015 -type inputs
```

The toggle count value of 0.25 and glitch count value of 0.05 are divided by clock period (10). The resulting toggle rate value of 0.025 and glitch rate value of 0.005 are annotated on the design objects.

The following example annotates state_condition dependent static probabilities on the cell or1:

```
pt_shell> set_switching_activity -static_probability 0.10 -state_condition
"A & B" [get_cell or1]
pt_shell> set_switching_activity -static_probability 0.25 -state_condition
"A & ! B" [get_cell or1]
pt_shell> set_switching_activity -static_probability 0.25 -state_condition
"! A & B" [get_cell or1]
pt_shell> set_switching_activity -static_probability 0.30 -state_condition
"! A & ! B" [get_cell or1]
pt_shell> set_switching_activity -static_probability 0.10 -state_condition
"default" [get_cell or1]
```

The following example annotates simple and path dependent toggle rates and glitch rates on the output pin Y of the cell xor1:

```
pt_shell> set_switching_activity -toggle_count 0.022 -glitch_count 0.001
[get_pin xor1/Y]
pt_shell> set_switching_activity -toggle_count 0.020 -glitch_count 0.001 -
path_sources "A" -state_condition "B" [get_pin xor1/Y]
pt_shell> set_switching_activity -toggle_count 0.002 -glitch_count 0.0001 -
path_sources "B" -state_condition "A" [get_pin xor1/Y]
pt_shell> set_switching_activity -toggle_count 0.002 -glitch_count 0.0001 -
path_sources "A B" -state_condition "default" [get_pin xor1/Y]
```

The following example annotates toggle rate to all clock gating cell outputs. The toggle rate value will be a factor (0.1 for this case) of the related clock of a certain clock gating cell.

```
pt_shell> set_switching_activity -clock_derate 0.1 -clocks [all_clocks] -
type clock_gating_cells
```

SEE ALSO

create_clock (2), **report_switching_activity** (2), **read_saif**(2),

```
reset_switching_activity(2), get_switching_activity(2), report_power (2);
```

set_temperature

Applies an operating temperature on a list of cell objects.

SYNTAX

```
int set_temperature  
max_case_temperature  
[-min min_case_temperature]  
-object_list list_of_cells
```

Data Types

<i>max_case_temperature</i>	float
<i>min_case_temperature</i>	float
<i>list_of_cells</i>	list

ARGUMENTS

max_case_temperature
Specifies the operating temperature for the maximum (worst) case.

-min *min_case_temperature*
Specifies the operating temperature for the minimum (best) case.

-object_list *list_of_cells*
Specifies the list of cells that will have the operating temperatures as specified in this command.

DESCRIPTION

Defines the operating temperature on the cells so that these cells of the design are timed and optimized at the specified temperature. If you do not specify any operating temperature for the cells, they will continue to be timed and optimized based on the available operating condition settings.

If *temperature_ranges* are specified for a cell, the operating temperatures of the cell must fall within one of the ranges specified in the *temperature_ranges*. Furthermore, *min_case_temperature* must fall within the same range as the *max_case_temperature*.

The operating temperature of a cell, once defined, can only be overridden with **set_temperature** on cell. It can also be cleared using **reset_design**.

EXAMPLES

The following example shows a typical context of use for the **set_temperature** command.

```
prompt> set_temperature 125 \  
-object_list [get_cells I1]
```

1

The above example sets a max_case_temperature of 125 on cell I1.

```
prompt> set_temperature 125 \  
-min 25 \  
-object_list [get_cells I2]  
1
```

The above example sets a max_case_temperature of 125 and min_case_temperature of 25 on cell I2.

SEE ALSO

set_voltage(2)

set_timing_derate

Sets delay derating factors for either the current design or a specified list of instances (cells, library cells, or nets).

SYNTAX

```
int set_timing_derate  
-early | -late  
[-rise] [-fall]  
[-clock] [-data]  
[-cell_delay] [-cell_check] [-net_delay]  
[-static] [-dynamic]  
[-scalar | -variation | -aocvm_guardband]  
derate_value  
object_list
```

Data Type

<i>derate_value</i>	float
<i>object_list</i>	list

ARGUMENTS

- early
Indicates that the *derate_value* specified should be applied to early delays (shortest paths). This option and the -late option are mutually exclusive.
- late
Indicates that the *derate_value* specified should be applied to late delays (longest paths). This option and the -early option are mutually exclusive.
- rise
Indicates that the *derate_value* specified should be applied to rise delays.
- fall
Indicates that the *derate_value* specified should be applied to fall delays.
- clock
Indicates that the specified derating factor should apply only to clock paths.
- data
Indicates that the specified derating factors should apply only to data paths.
- net_delay
Indicates that the specified derating factor should apply to net delays. This option cannot be specified with the -cell_check option.
- cell_delay
Indicates that the specified derating factor should apply to cell delays. This option cannot be specified with the -cell_check option.

-cell_check
 Indicates that the specified derating factor should apply only to cell timing checks. If this option is specified, the derate value set with the *-early* option is applied to hold and removal timing checks and the derate value set with the *-late* option is applied to setup and recovery timing checks. This option cannot be specified with any of the following options: *-clock*, *-data*, *-cell_delay*, *-net_delay*, and *-aocvm_guardband*.

-static
 Indicates that the specified derating factor should apply to the nondelta portion of net delays only. This option requires that the *-net_delay* option is specified.

-dynamic
 Indicates that the specified derating factor should apply to the dynamic component of delays only. This option requires that the *-net_delay* option is specified. This option cannot be specified with the *-aocvm_guardband* option.

-scalar
 Indicates that the specified derating factor should apply to deterministic delays only.

-variation
 Indicates that the specified derating factor should apply to statistical delays only.

-aocvm_guardband
 This option is only applicable in an advanced on-chip variation (OCV) context. In an advanced OCV context, the derate factor that is applied to an arc is a product of the guard-band derate factor and the advanced OCV derate factor. This option cannot be specified with any of the following options: *-scalar*, *-variation*, *-dynamic*, or *-cell_check*.

derate_value
 Specifies the timing derate value that are applied to the specified delays as a scalar multiplicative factor.

object_list
 Specifies current design or a list of cells, library cells, or nets to which the specified derate factor are applied.

DESCRIPTION

Sets derating factors for either the current design (if no object list is specified) or a set of cells, library cells, or nets in the current design.

Timing derating factors affect delay values shown in timing reports. Longest path delays (for example, launching paths for setup checks or capturing paths for hold checks) are multiplied by the derate value set using the *-late* option, and shortest paths (for example, capturing paths for setup checks or launching paths for hold checks) are multiplied by the derate values set using the *-early* option. If derating factors are not specified, the a value of 1.0 is assumed.

The intended usage mode of this command is that you first set global or default

set_timing_derate

derates on the design and then you can use the *object_list* option to set specific derate values for cells, library cells, or nets in the design. To set derates globally on a design simply issue the command with no object list specified. For example, to set an early derate factor of value 0.95 on all cell and net delays in the design, use:

set_timing_derate -early 0.95

If only the *-net_delay* option is specified then the derate factor supplied will be applied to all net delays in the design (if no *object_list* is specified). If an *object_list* is specified, the derate factor is applied to each net and/or to every net in each hierarchical cell in the *object_list*.

If only the *-cell_delay* option is specified, the derate factor supplied is applied to either all cell delays in the design (if no *object_list* is specified) or to each delay (apart from timing check delays) on each cell or library cell specified in the *object_list*.

If neither *-net_delay*, *-cell_delay*, or *object_list* is specified with the command, it is assumed that the derate factor is to be applied to all cell and net delays in the design.

If *-net_delay* is specified with the command and neither *-static* nor *-dynamic* has been specified, it is assumed that the derate factor is to be applied to both the dynamic and nondynamic portions of net delays in the design.

If only the *-clock* option is specified, the derate factor supplied is only applied to delays that are in clock paths. If only the *-data* option is specified, the derate factor supplied is only applied to delays that are in data paths. If neither of the above options is specified, it is assumed that the derate factor is to be applied to both clock and data paths.

If neither the *-scalar* nor *-variation* option is specified, it is assumed that the derate factor is to be applied to both deterministic and variation delays.

If you omit the *-net_delay* or *-cell_delay* option from the command, PrimeTime automatically determines the intended scope of the command from the types of objects in the object list. For example,

```
pt_shell> set_timing_derate -late 1.1 [get_nets n*]
```

```
Warning: Implicitly setting the 'net_delay' option for the set_timing_derate
command. (UITE-437)
```

If the collection contains nets, PrimeTime infers the *-net_delay* option. If the collection contains cells, PrimeTime infers the *-cell_delay* option. If the collection contains both nets and cells, PrimeTime infers both.

The *-cell_check* option applies the derate value to the setup and hold time requirements of cells: cell setup and cell recovery times for late derating or cell hold and cell removal times for early derating. These time values are timing constraints, not delays. The derate factor applies to the cell instances or library cells specified in the object list, or to all timing checks in the design if no object list is provided.

Note that the following commands set the derating factors for all cell and net

delays:

```
pt_shell> set_timing_derate -late 1.3
pt_shell> set_timing_derate -early 0.8
```

whereas the following commands set the derating factors for all cell setup time and cell hold time constraints:

```
pt_shell> set_timing_derate -late 1.3 -cell_check
pt_shell> set_timing_derate -early 0.8 -cell_check
```

If the **timing_use_constraint_derates_for_pulse_checks** variable is set to *true*, the *derate_value* value specified by the *-cell_check -late* option also applies to the min-pulse-width and min-period constraints. In this case, the *-fall* option applies to min-pulse-width checks for low pulses, the *-rise* option applies to min-pulse-width checks for high pulses. The greater the *derate_value* from the *-rise* or *-fall* option applies to min-period checks.

The *object_list* option can be used to set specific derate factors on instances (cells or library cells) in the design. On each instance the *-cell_delay*, *-cell_check*, *-clock*, and *-data* options can be used in the same way as outlined above to specify exactly how the derate factors should be applied to each instance in the object list.

If no options are specified and the object list does not contain hierarchical cells, it is assumed that *-cell_delay*, *-net_delay*, *-clock*, and *-data* are *true*.

If no options are specified and the object list contains hierarchical cells, *-cell_delay*, *-net_delay* or *-cell_check* must be specified.

When applying derate factors the following priority is used (in decreasing order of precedence):

- 1) Leaf Cell
- 2) Hierarchical Cell
- 3) Library Cell
- 4) Design

Therefore, when derate factors are being applied for a cell, PrimeTime first checks if they have been set for the cell itself (leaf cell). It then checks if it has been set for any of the cells hierarchical parents. If no derate factors are found, it checks for the library cell and after that it uses the factors set globally on the design.

In addition, if the *-net_delay* option is specified and the *object_list* contains any hierarchical cells, all nets within that hierarchical cell and also all nets of all lower hierarchical cells have that derate factor set on them.

The *object_list* option can also be used to set specific derate factors on particular nets in the design. A derate factor set on a specific net has precedence over net derate factors set globally. If a derate factor is set on a portion of a hierarchical net, all portions of that hierarchical net also have that derate factor set and a warning is issued indicating this fact. Any values previously set on this hierarchical net is overwritten.

PrimeTime only inherits derates onto global nets from the parent cell of the highest hierarchical level of the global net. This means that derates are not inherited onto global nets from parent cells of any lower hierarchical levels. This convention resolves inconsistent application of derates onto global nets.

To set derating values back to the default (derate factor of 1.0 for every instance in the design), use the **reset_timing_derate** command.

For delays an early factor less than 1 and late factor more than 1, add pessimism to the slack calculation. However, constraint derating is applied directly to the value of constraint. For example, a factor of less than 1 for hold means making hold time smaller (for positive hold time) thus giving less conservative slack. A factor greater than 1 for setup means making setup time larger (for positive setup time) thus more conservative slack.

The following equation is used to apply derates to both positive and negative delays:

$$\bullet \text{ derated\_delay} = \text{delay} + ((\text{derate\_factor} - 1.0) * \text{ABS}(\text{delay}))$$

This equation ensures that negative delays are correctly made more conservative by the application of derates.

If Signal Integrity is enabled, net delays might also include delta delay resulting from crosstalk interaction between nets. Because minimum delta delays are negative, net derate factors are applied separately to the net_delay (without any delta delay component) and delta delay. Both derated values are subsequently summed to give the total derate net delay.

In addition, if the **si_use_driving_cell_derate_for_delta_delay** variable is set to *true*, the delta delay specified in the previous paragraph is derated using the relevant derate factor from the cell driving that net.

If the operating condition analysis type was previously set to *single*, executing the **set_timing_derate** command automatically switches to the *on_chip_variation* analysis type. This is equivalent to executing **set_operating_conditions -analysis_type on_chip_variation** command. Otherwise, the analysis modes of *bc_wc* or *on_chip_variation* remain unchanged.

If only global derate factors are specified, their values can be shown using the **report_design** command. This also indicates what analysis mode PrimeTime is in.

To report all derate factors set for the design or for any cell, library cell, or net in the design, use the **report_timing_derate** command.

PrimeTime version V-2003.12 does not support using different timing derates for different operating conditions just like other tools, such as Design Compiler. Because of this, the tool does not accept per-operating condition derates read in from scripts or DB files.

EXAMPLES

The following example sets an early (shortest path) derate factor of 0.7 and a late

(longest path) derate factor of 1.0 globally on all cell and net delays the design:

```
pt_shell> set_timing_derate -early 0.70  
pt_shell> set_timing_derate -late 1.0
```

The following example sets an early derate factor of 0.8 for the cell delay of cell U1:

```
pt_shell> set_timing_derate -early 0.8 -cell_delay [get_cells U1]
```

The following example sets a late derate factor of 1.1 on all instances of library cell IV in the library MY_LIB:

```
pt_shell> set_timing_derate -late 1.1 [get_lib_cells MY_LIB/IV]
```

Assuming that cell, "inv" is a particular instantiation of MY_LIB/IV in the design the following command overwrites the late derate factor for the instance "inv" only:

```
pt_shell> set_timing_derate -late 1.05 [get_cells inv]
```

The following example illustrates how to set a derate factor on all cells and nets (including sub-hierarchies) within the hierarchy top/H1:

```
pt_shell> set_timing_derate -cell_delay -net_delay -late 1.05 [get_cells top/H1]
```

The following shows how to set an early derate factor of 0.7 on a specific net 'n1':

```
pt_shell> set_timing_derate -net_delay -early 0.7 [get_nets n1]
```

SEE ALSO

```
report_design(2)  
report_timing(2)  
report_timing_derate(2)  
reset_timing_derate(2)  
si_use_driving_cell_derate_for_delta_delay(3)  
set_operating_conditions(2)
```

set_units

Checks the specified units with the main library units. The command fails if the units specified do not match the main library units.

SYNTAX

```
int set_units  
[-time[optional float][optional scale_value]s]  
[-capacitance[optional float][optional scale_value]F]  
[-resistance[optional float][optional scale_value]Ohm]  
[-voltage[optional float][optional scale_value]V]  
[-current[optional float][optional scale_value]A]  
[-power[optional float][optional scale_value]W]
```

Data Types

scale_value [f/p/n/u/m/k/M] character

ARGUMENTS

- time[optional float][optional scale_value]s
Checks the time unit with the time unit of the main library.
- capacitance[optional float][optional scale_value]F
Checks the capacitance unit with the capacitance unit of the main library.
- resistance[optional float][optional scale_value]Ohm
Checks the resistance unit with the resistance unit of the main library.
- voltage[optional float][optional scale_value]V
Checks the voltage with the voltage unit of the main library.
- current[optional float][optional scale_value]A
Checks the current unit with the current unit of the main library.
- power[optional float][optional scale_value]W
Checks the power unit with the leakage power unit of the main library. The power units can only be checked if power analysis mode is enabled.

DESCRIPTION

The **set_units** command checks the units specified with the main library units. This command can only be used after the design has been linked. The main library units are the design units. The **set_units** command does not allow setting of design units, but is intended to prevent inadvertent use of the wrong units from different SDC files. The power units can only be checked if power analysis mode is enabled. This can be done by setting the **power_enable_analysis** variable to *true* and running power related commands, such as the **update_power**, **read_vcd**, **read_saif**, and **set_switching_activity** commands. You can check all units using the **report_units** command. The **read_sdc** command supports the **set_units** command. The **write_sdc** and **write_script** commands write the **set_units** command as the first command at their

output.

EXAMPLES

The following example checks the if the main library time unit is 1ns, capacitance unit is 1pF, current unit is 1mA and voltage unit is 1V.

```
pt_shell> set_units -time ns -capacitance pF -current mA -voltage V
```

The following example checks if the main library capacitance unit is 0.5pF.

```
pt_shell> set_units -capacitance 0.5pF
```

SEE ALSO

```
report_units(2)  
read_sdc(2)  
write_sdc(2)  
write_script(2)
```

set_user_attribute

Sets a user attribute to a specified value on an object.

SYNTAX

```
string set_user_attribute  
[-class class_name]  
[-quiet]  
object_spec  
attr_name  
value
```

```
string class_name  
list object_spec  
string attr_name  
string value
```

ARGUMENTS

-class *class_name*
If *object_spec* is a name, this is its class. Allowable values are design, port, cell, pin, net, lib, lib_cell, or lib_pin.

-quiet
Suppresses all report messages.

object_spec
Objects on which to set the attribute. Each element in the list is a collection or a pattern which is combined with the *class_name* to find the objects.

attr_name
Shows the name of the attribute.

value
Shows the value of the attribute.

DESCRIPTION

The **set_user_attribute** command sets attributes on objects. These attributes are defined with **define_user_attribute**.

You cannot set application attributes with this command. Each settable application attribute has a command dedicated to it. For example, the *fanout_load* attribute is set with the **set_fanout_load** command. However, you can still retrieve the values of any application or user-defined attribute using the **get_attribute** command.

EXAMPLES

This example defines an attribute 'X' for cells, then sets the value on all cells in this level of the hierarchy.

```
pt_shell> define_user_attribute -type int -class cell X
pt_shell> set_user_attribute [get_cells *] X 30
Set attribute 'X' on 'i1'
Set attribute 'X' on 'i2'
```

SEE ALSO

collections (2), **define_user_attribute** (2), **get_attribute** (2), **list_attributes** (2), **remove_user_attribute** (2), **report_attribute** (2).

set_user_sensitization

Specify user sensitization for a timing or library arc in PrimeTime.

SYNTAX

```
int set_user_sensitization  
-analysis_type [rise | fall | high | low]  
arcs_list  
-event_pins input_pins_list  
-event_states list_of_r_f_1_and_0s  
[-event_step separation_time_in_library_unit]  
[-initial_condition { node_name volt_in_library_unit }]  
[-tie_high logic_one_input_pin_list]  
[-tie_low logic_zero_input_pin_list]  
[-use_pwl_for_clock]
```

Data Types

```
list arcs_list  
list input_pins_list  
list list_of_r_f_1_and_0s  
float separation_time_in_library_unit  
string node_name  
float volt_in_library_unit  
list logic_one_input_pin_list  
list logic_zero_input_pin_list<plain
```

ARGUMENTS

-analysis_type [*rise* | *fall* | *high* | *low*]
Specifies the final state of the output pin of timing or library arc.

arcs_list
Specifies a list timing or library arcs to be sensitized. You can use **get_timing_arcs** or **get_lib_timing_arc** commands.

-event_pins *input_pins_list*
Specifies a list of input pins of the cell on which the sensitization sequence/vector is applied to achieve the desired state at output pin. It must include the input pin of the timing/library arc.

-event_states *list_of_r_f_1_and_0s*
Specifies the sensitization sequence to sensitize the arc. It is expressed as a list of **1**, **0**, **r**, and **f**. They stand for the states of logic one, logic zero, rising, and falling. The number of states must be a multiple of the number of pins in the **-event_pins** options.

-event_step *separation_time_in_library_unit*
Specifies the minimum time between the events. It is use to remove pulses of small width to avoid sensitization problems.

-initial_condition { node_name volt_in_library_unit }
 Specifies the initial condition of a node present in the cell containing the arc. Correct initialization of nodes may reduce the length of sensitization vectors.

-tie_high logic_one_input_pin_list
 Specifies a list of input pins to be connected to a voltage source that represents logic one of this cell in SPICE simulation.

-tie_low logic_zero_input_pin_list
 Specifies a list of input pins to be connected to a voltage source that represents logic zero of this cell in SPICE simulation.

-use_pwl_for_clock
 Specifies to force the usage of PWL statements instead of a pulse statements for clock input pins.

DESCRIPTION

The **set_user_sensitization** command specifies the user sensitization for timing arc or library arc collection. It allows you to override PrimeTime default sensitization used in the **write_spice_deck** command of a given arc. If the arc is a timing arc, the sensitization sequence given by this command is applied to the specified arc of the design instance. If the arc is a library arc, all arcs of this library cell are sensitized using this sequence.

The **set_user_sensitization** command is currently intended to only supplement the behavior of **write_spice_deck** command. This command does NOT change the behavior of any other part of PrimeTime. After a sensitization sequence is specified to an arc, it stays effective for the rest of the PrimeTime session. If you want to remove this sensitization vector, use **remove_user_sensitization** command. Specifying new sequence will replace the old sensitization sequence on the arc.

All the input pins of the cell that contains the arc must be specified exactly once in either **-tie_high**, **-tie_low**, or **-event_pins** option. At least one transition must be specified in the **-event_states** option for the analysis type rise or fall. The order of state-transitions in the **-event_states** option is important. They describe the sequential order of state changes on input pins to achieve the desired output transition.

Consecutive group of pin state specified by sequence must be consistent. All timing/library arcs in the arc collection must be compatible to the other conditions on the pins. For example, the final states of all input pins must satisfy the when condition of the arc. State changes for a given pin must satisfy the following rules:

0 and **f** must be followed by **0** and **r**
1 and **r** must be followed by **1** and **f**

In addition to the event sequence, you can also specify the initial conditions on some internal nodes and the output pins of the cell. The **-initial_condition** option causes **write_spice_deck** to write the initial transient condition for the cell instance of the timing arc or library-timing arc. For example, assume top/block1/ff1 is a flip-flop in the design and its corresponding library timing arc is sensitized

with the option

```
-initial_condition { .n1 0.5 }
```

The **write_spice_deck** command prints the following .IC statement in the SPICE deck for a path that uses top/block1/ff1 as the launching cell.

```
.IC V(top/block1/ff1.n1) = 0.5
```

EXAMPLES

Rising Q pin sensitization of JK Flip-flop's clock to Q arc.

```
set_user_sensitization -analysis_type rise \  
-tie_high {SDN CDB} \  
-event_pins {CP J KZ} \  
-event_states {r 0 0 \  
f r r \  
r 1 1} \  
[get_timing_arcs -from ujk1/CP -to ujk1/Q] \  

```

Sensitization of S to rising X arc of a MUX cell of instance umax0.

```
set_user_sensitization -analysis_type rise \  
-event_pin { D0 D1 S } \  
-event_states { r f 1 \  
1 0 f } \  
[ get_timing_arcs -from umux0/S -to umax0/X ] \  

```

Sensitization of D1 to rising X arc of a MUX cell of instance umax0.

```
set_user_sensitization -analysis_type rise \  
-event_pin { D0 D1 S } \  
-event_states { 0 r 1 } \  
[ get_timing_arcs -from umux0/D1 -to umax0/X ] \  

```

or

```
set_user_sensitization -analysis_type rise \  
-tie_high { S } -tie_low { D0 } \  
-event_pin { D1 } \  
-event_states { r } \  
[ get_timing_arcs -from umux0/D1 -to umax0/X ]
```

Sensitization of S to rising X arc in no-unate sense of all library cell SDN_MUX2_1 of the library db13ugfsdsc_ss.

```
set_user_sensitization -analysis_type rise \  
-tie_high { D0 } -tie_low { D1 } <HardSpace  
-initial_condition { X 0.0 } \  
-event_pin { S } \  
-event_states { f } \  
[get_lib_timing_arcs -from db13ugfsdsc_ss/SDN_MUX2_1/S \  
-to db13ugfsdsc_ss/SDN_MUX2_1/X ]
```

Sensitization of A to X arc of a three input XOR gate.

```
set lib_arc [get_lib_timing_arcs -from LV160C/XOR3HV15/A -to LV160C/XOR3HV15/Z]
set_user_sensitization -analysis_type rise -event_pin A \
    -event_states r -tie_low { B C } $lib_arc
set_user_sensitization -analysis_type fall -event_pin A \
    -event_states f -tie_low { B C } $lib_arc
```

Sensitization of data flip-flop with enable pin.

```
set larcs_0 [ get_lib_timing_arcs -from ITS_FF/XFFF0/CP -to ITS_FF/XFFF0/Q ]
set_user_sensitization -analysis_type rise \
    -event_pin { D E CP } \
    -event_states { 0 1 r \
r f 1 \
1 0 f \
1 r 0 \
1 1 r } $larcs_0
set_user_sensitization -analysis_type fall \
    -event_pin { D CP } \
    -tie_high { E } \
    -event_states { 1 r \
f f \
0 r } \
$larcs_0
```

SEE ALSO

```
write_spice_deck(2)
remove_user_sensitization(2)
report_user_sensitization(2)
```

set_variation

Sets one or more variations onto one or more timing objects.

SYNTAX

```
int set_variation  
variation_list  
[object_list]  
collection variation_list  
collection object_list
```

ARGUMENTS

variation_list

The list of variations to be set on the timing objects. Each of these variations must have a *parameter_name* definition.

object_list

If missing, it defaults to the timing objects of the whole design. Otherwise, it applies to instances of libcell arcs.

DESCRIPTION

Sets each of the variations on each of the timing objects. Each variation must have been created by **create_variation** with a *parameter_name*. Once a variation is associated with a timing object it will affect the timing behavior of the timing object.

EXAMPLES

The following example loads a variation library with parameters "len" and "vth", creates a variation of name "Lfab" and parameter_name "len", and finally sets the variation on the current design.

```
pt_shell> set_variation_library -parameter_names "len vth" -  
values "100 0.2394" maxvth.db  
pt_shell> create_variation -parameter_name len -name Lfab -type normal -  
values {0 1}]  
_sel23  
pt_shell> set_variation [get_variations Lfab]  
pt_shell> set arc7 [get_timing_arcs -of_object [get_cells u7]]  
pt_shell> set_variation [create_variation -parameter_name vth -name vth_u7 -  
type normal -values {0 0.007} ] $arc7
```

SEE ALSO

reset_variation(2), create_variation(2).

set_variation_correlation

Applies a correlation type to a variation(s) or to all instances of that variation(s). May also apply a cross-correlation to a vector of variations.

SYNTAX

```
int set_variation_correlation  
-name variation_correlation_name  
-correlation correlation_object  
[object_list]  
[-all]  
string variation_correlation_name  
string correlation_object  
collection object_list
```

ARGUMENTS

-name correlation_name
Defines a name for this variation_correlation.

-correlation correlation_object
Specifies the correlation to be applied to the variations.

object_list
This is a list of variations that are to be correlated. If the corresponding correlation is not of the cross-correlation type, exactly one of *object_list* and *-all* must be specified. If it is, *-all* must not be specified.

-all
When this option is present all variations that are created with a *parameter_name* are correlated. Exactly one of *object_list* and *-all* must be specified.

DESCRIPTION

Applies a correlation that was created by **create_correlation** to one or more variations.

There are two exclusive types of correlations: auto-correlation, which applies to instances of the same variation, or cross-correlation, which applies across variations. For illustrative purposes, assume we are interested in device variations. Auto-correlations would involve the same variation but applied to different devices, whereas cross-correlations would involve different variations, but applied to a same device.

If the correlation was created with option *-constant*, each variation in the *object_list* will have an auto-correlation described by that correlation.

If the correlation was created with option *-cross_correlations*, all the variations in the *object_list* will be jointly correlated, with correlation matrix deduced by

the values listed in the `create_correlation` statement.

In this release, one cannot set a cross-correlation via the `-all` option; the variations must be explicitly listed. The number of listed variations must correspond to the number of correlation values listed in option `-cross_correlation` of `create_correlation`.

EXAMPLES

```
pt_shell> set_variation_library -parameter_names "len vth" -  
values "100 0.2394" maxvth.db  
pt_shell> create_variation -parameter_name len -name Lfab1 -type normal -  
values {0 1}]  
pt_shell> create_variation -parameter_name vth -name Lfab2 -type normal -  
values {0 1}]  
pt_shell> create_correlation -name full_correlation -constant 1  
pt_shell> set_variation_correlation -name device_len -  
correlation full_correlation [get_variations Lfab*]  
pt_shell> set_variation_correlation -name device -  
correlation xcorr {len_die len_waf}
```

SEE ALSO

`create_correlation(2)`, `create_variation(2)`.

set_variation_library

Sets the library that defines a point in the variation space.

SYNTAX

```
int set_variation_library  
[-parameter_names variation_name_list]  
[-values variation_value_list]  
[-reference_value]  
[library_file_name]  
[-link_library link_library_list]
```

```
list variation_name_list  
list variation_value_list  
string library_file_name  
list link_library_list
```

ARGUMENTS

-parameter_names *variation_name_list*
Specifies a list of parameter names of the *m* variations that define a point in the *n* dimensional variation space where *n* is larger than or equal to *m*. When *reference_value* is used, *m* must equal *n*.

-values *variation_value_list*
Specifies a value for each variation name.

-reference_value
States that this point is the reference value point. This option requires the use of the values option. The first invocation of the command must use this option.

library_file_name
Specifies the file name of the technology library.

-link_library *library_linked*
Specifies the library(ies) the *library_file_name* must be linked to. If **-link_library** is not specified, **-parameter_names**, **-values**, and *library_file_name* are required options.

DESCRIPTION

Loads technology library files that are characterized in the variation space.

EXAMPLES

The following example loads the library file *typical.db* that is characterized with the channel length equal to 100nm and threshold voltage at 0.228V, which are the nominal points of variations on *len* and *vth* in the variation space.

```
pt_shell> set_variation_library -parameter_names "len vth" -values \
```

```
set_variation_library  
1160
```



```
"100 0.228" -reference_value typical.db
```

```
1
```

The following example loads the library files maxlen.db and minlen.db that are characterized with the channel length equal to 105nm and 95nm respectively, with the threshold voltage at the nominal point 0.228V.

```
pt_shell> set_variation_library -parameter_names "len vth" -values \  
          "105 0.228" maxlen.db
```

```
1
```

```
pt_shell> set_variation_library -parameter_names "len vth" -values \  
          "95 0.228" minlen.db
```

```
1
```

The following example loads the library files maxvth.db and minvth.db that are characterized with the threshold voltage equal to 0.2394V and 0.2166V respectively, with channel length at the nominal point 100nm.

```
pt_shell> set_variation_library -parameter_names "len vth" -values \  
          "100 0.2394" maxvth.db
```

```
1
```

```
pt_shell> set_variation_library -parameter_names "len vth" -values \  
          "100 0.2166" minvth.db
```

```
1
```

SEE ALSO

```
create_variation(2), remove_variation(2).
```

set_variation_quantile

Determines quantiles for analysis and reporting.

SYNTAX

```
int set_variation_quantile  
[-quantile_high high_value]  
[-quantile_low low_value]  
float high_value  
float low_value
```

ARGUMENTS

-quantile_high *high_value*
Specifies the quantile level closer to 1.0. Must be a value between 0.0 and 1.0, exclusive. The default is 0.99865.

-quantile_low *low_value*
Specifies the quantile level closer to 0.0. Must be a value between 0.0 and 1.0, exclusive. The default is 0.00135.

DESCRIPTION

Sets quantiles for analysis and reporting to reduce distributions to scalars.

The default high quantile is 0.99865 and the default low quantile is 0.00135.

EXAMPLES

The following example sets the high quantile to be 0.90 and the low quantile to be 0.10 for statistical analysis. In the resulting timing report, the "data arrival time 0.027899" corresponds to the 90% quantile value 0.027899ns of the arrival distribution at the end point. The "statistical adjustment 0.003720 0.971734" refers to the 10% quantile value 0.971734ns of the slack distribution of the timing path, with the adjustment of 0.003720ns compared to the corner slack.

```
pt_shell> set_variation_quantile \  
                                -quantile_high 0.90 -quantile_low 0.10
```

```
1
```

```
pt_shell> report_timing -significant_digits 6 $stat_path
```

```
*****
```

```
Report : timing
```

```
        collection of 1 path(s)
```

```
        -path full
```

```
Design : inv1
```

```
*****
```

```
Startpoint: in1 (input port clocked by CLK)
```

```
Endpoint: out1 (output port clocked by CLK)
```

```
Path Group: CLK
```

```
set_variation_quantile
```

```
1162
```

Path Type: max

Point	Incr	Path

clock CLK (rise edge)	0.000000	0.000000
clock network delay (ideal)	0.000000	0.000000
input external delay	0.000000	0.000000 f
in1 (in)	0.000000	0.000000 f
buf0/ZN (INVD1)	0.031986 +	0.031986 r
out1 (out)	0.000000 +	0.031986 r
statistical adjustment	-0.004087	0.027899
data arrival time		0.027899
clock CLK (rise edge)	1.000000	1.000000
clock network delay (ideal)	0.000000	1.000000
output external delay	0.000000	1.000000
statistical adjustment	0.000000	1.000000
data required time		1.000000

data required time		1.000000
data arrival time		-0.027899

corner slack		0.968014
statistical adjustment	0.003720	0.971734
slack (MET)		0.971734

1

In this example, the high and low quantile values are set to be 0.60 and 0.40. In the timing report, the data arrival time is reduced to 0.027042ns that is equal to the 60% quantile value of the arrival distribution at the end point. The statistical slack is increased to 0.972719ns according to the 40% quantile value of the slack distribution of the timing path.

```
pt_shell> set_variation_quantile \
               -quantile_high 0.60 -quantile_low 0.40
```

1

```
pt_shell> report_timing -significant_digits 6 $stat_path
*****
```

Report : timing
collection of 1 path(s)
-path full

Design : inv1

Startpoint: in1 (input port clocked by CLK)
Endpoint: out1 (output port clocked by CLK)
Path Group: CLK
Path Type: max

Point	Incr	Path

clock CLK (rise edge)	0.000000	0.000000
clock network delay (ideal)	0.000000	0.000000
input external delay	0.000000	0.000000 f
in1 (in)	0.000000	0.000000 f
buf0/ZN (INVD1)	0.031986 +	0.031986 r
out1 (out)	0.000000 +	0.031986 r
statistical adjustment	-0.004944	0.027042
data arrival time		0.027042
clock CLK (rise edge)	1.000000	1.000000
clock network delay (ideal)	0.000000	1.000000
output external delay	0.000000	1.000000
statistical adjustment	0.000000	1.000000
data required time		1.000000

data required time		1.000000
data arrival time		-0.027042

corner slack		0.968014
statistical adjustment	0.004705	0.972719
slack (MET)		0.972719

1

SEE ALSO

create_variation (2), **create_correlation** (2), **report_timing** (2).

set_voltage

Applies an operating voltage on a list of power nets or pg pins.

SYNTAX

```
int set_voltage  
max_case_voltage  
[-min min_case_value]  
[-dynamic dynamic_max_case_value]  
[-min_dynamic dynamic_min_case_value]  
[-object_list list_of_power_nets]  
[-cell cell]  
[-pg_pin_name pg_pin]
```

Data Types

<i>max_case_voltage</i>	float
<i>min_case_voltage</i>	float
<i>dynamic_max_case_voltage</i>	float
<i>dynamic_min_case_voltage</i>	float
<i>list_of_power_nets</i>	list
<i>cell</i>	list
<i>pg_pin</i>	string

ARGUMENTS

max_case_voltage
Specifies the operating voltage for the maximum delay (slowest) case. This is typically a smaller voltage value.

-min *min_case_voltage*
Specifies the operating voltage for the minimum delay (fastest) case. This is typically a larger voltage value.

-object_list *list_of_power_nets*
Specifies the list of power nets that will have the operating voltages as specified in this command.

-cell *cell*

-pg_pin_name *pg_pin*
The two options are used together to specify power or ground pin of cell instance for IR drop annotation.

-dynamic *dynamic_max_case_value*

-min_dynamic *dynamic_min_case_value*
Specifies a portion of the voltage changes that varies with frequency significantly higher than several clock periods. The dynamic portion is considered during capacitive coupling analysis and various pessimism removal optimizations.

DESCRIPTION

Defines the operating voltage on the power nets or pins so that the parts of the design powered by these power nets are timed and optimized at the specified voltage. If you do not specify any operating voltage for a power net, the part of the design connected by this power net will continue to be timed and optimized based on the available operating condition settings.

If *voltage_ranges* are specified for a power net, the operating voltages of the power net must fall within one of the ranges specified in the *voltage_ranges*. Furthermore, *min_case_voltage* must fall within the same range as the *max_case_voltage*.

To see the operating voltages of power nets, use **report_power_net_info**.

The operating voltage of a power net, once defined, can only be overridden with **set_voltage** on power pin. It can also be cleared using **reset_design**.

When setting voltage on UPF supply nets the voltage is set only on the segments connected to the segment in *-object_list* at the time of executing *set_voltage*. Therefore it is important to load complete UPF description of power network before issuing *set_voltage*. You can also check whether any power net segment has missing voltage by *check_timing -override supply_net_voltage*.

EXAMPLES

The following example shows a typical context of use for the **set_voltage** command.

```
prompt> create_power_domain TOP
1
prompt> create_supply_net VDD -domain TOP
1
prompt> create_supply_net VSS -domain TOP
1
prompt> set_domain_supply_net TOP -primary_power_net VDD \
                                -primary_ground_net VSS
1
prompt> set_voltage 0.88 -min 1.188 -cell I1 -pg_pin_name PWR
1
prompt> set_voltage 0.04 -min -0.05 -cell I1 -pg_pin_name GND
1
```

SEE ALSO

```
create_power_domain(2)
connect_power_domain(2)
create_supply_net(2)
report_power_pin_info(2)
report_power_domain(2)
check_timing(2)
```

set_wire_load_min_block_size

Sets the minimum block area for automatic wire load selection. Any blocks with an area below the minimum are promoted to the minimum.

SYNTAX

```
int set_wire_load_min_block_size block_size  
float block_size
```

ARGUMENTS

block_size

Minimum block size for auto wire load selection. This float value must be greater than or equal to 0.0.

DESCRIPTION

Specifies the minimum block area for automatic wire load selection in the current design. When automatic wire load selection is enabled, hierarchical cells are assigned wire load models based on their area. Any cells with an area below the minimum block size are assigned the minimum *block_size* before a wire load model is selected. If the minimum block size is not explicitly set, the default value of 0.0 is used.

EXAMPLES

This example specifies a minimum block size of 1000 area units.

```
pt_shell> set_wire_load_min_block_size 1000
```

SEE ALSO

report_wire_load (2), **set_wire_load_selection_group** (2), **auto_wire_load_selection** (3).

set_wire_load_mode

Sets wire load mode for the current design.

SYNTAX

```
string set_wire_load_mode mode_name  
stringmode_name
```

ARGUMENTS

mode_name
Name of mode: *top*, *enclosed*, or *segmented*.

DESCRIPTION

Specifies a wire load mode with the **set_wire_load_mode** command. If the mode for the top level design is *top*, the wire load model on hierarchical cells has no effect, and the top-level wire load model is used to compute wire capacitance for all nets within the design.

If the mode for the top-level design is either *enclosed* or *segmented*, wire load models on hierarchical cells are used to calculate wire capacitance, resistance, and area in nets inside these blocks. If the *enclosed* mode is set, net values are determined using the wire load model of the hierarchical cell which fully encloses the net. If the *segmented* mode is set, segments of the net in each level of hierarchy are calculated separately, and the net value is the total of all segments. Hierarchical boundary pins are included in the pin count for the segment.

If no **wire_load_model_mode** is specified for a design, a default **wire_load_model_mode** is searched for in the first library in the link path. If the library checked does not have a default **wire_load_model_mode**, *top* is assumed.

EXAMPLES

This example sets the wire load mode to *enclosed* on the current design.

```
pt_shell> set_wire_load_mode enclosed
```

SEE ALSO

```
report_wire_load(2), set_wire_load_model(2).
```


set_wire_load_model

Sets wire load model on designs, ports, or hierarchical cells.

SYNTAX

```
int set_wire_load_model -name model_name [-library lib_spec] [-min] [-max]
[object_list]
string model_name
string lib_spec
list object_list
```

ARGUMENTS

-name *model_name*
Specifies the name of the wire load model; must be a valid wire load model in a library specified in **link_path**.

-library *lib_spec*
Specifies the library in which to search for the *model_name*. If not specified, libraries in the *link_path* are searched.

-min
Indicates that the model is for minimum conditions.

-max
Indicates that the model is for maximum conditions.

object_list
Specifies a list of designs, ports, or hierarchical cells. The default is to set the wire model on the current instance, if set; or on the current design otherwise.

DESCRIPTION

Sets the wire load model on designs, ports, or hierarchical cells. The wire load model calculates net capacitance, resistance, and area for designs that have not been placed and routed.

Use automatic wire load selection for the design, as another way to specify wire load models. If this is enabled, hierarchical blocks have wire load models assigned automatically based on their cell area.

To remove the wire load model setting, use **remove_wire_load_model**.

EXAMPLES

The following example specifies a wire load model of "BIG" on the top level design, a model of "MEDIUM" on hierarchical cell "u1", and a model of "SMALL" on

hierarchical cell u1/u2.

```
pt_shell> current_design TOP
{"TOP"}
pt_shell> set_wire_load_model -name BIG
1
pt_shell> current_instance u1
u1
pt_shell> set_wire_load_model -name MEDIUM
1
pt_shell> current_instance u2
u1/u2
pt_shell> set_wire_load_model -name SMALL
1
pt_shell> current_instance
Current instance is the top-level of design 'TOP'.
pt_shell>
```

The following example specifies a wire load model of "BIG" on the top-level design, and a model of "SMALL" on all instances of design "adder."

```
pt_shell> current_design TOP
{"TOP"}
pt_shell> set_wire_load_model -name BIG
1
pt_shell> set_wire_load_model -name SMALL [all_instances -hierarchy adder]
1
```

The following example sets the external wire load model for ports OUT1* to 70x70, and specifies an external fanout number of 2.

```
pt_shell> set_wire_load_model 70x70 [get_ports OUT1*]
pt_shell> set_port_fanout_number 2 [get_ports OUT1*]
```

SEE ALSO

```
remove_wire_load_model(2), report_lib(2), report_wire_load(2),
set_port_fanout_number(2), set_wire_load_mode(2), set_wire_load_selection_group(2);
auto_wire_load_selection(3), link_path(3).
```

set_wire_load_selection_group

Sets the wire load selection group for current design.

SYNTAX

```
int set_wire_load_selection_group [-min] [-max] [-library lib_spec]  
selection_group_name [object_list]  
string lib_spec  
string selection_group_name  
list object_list
```

ARGUMENTS

-min

Specifies the selection group for minimum conditions.

-max

Specifies the selection group for maximum conditions.

-library *lib_spec*

Specifies the library in which to search for the *selection_group_name*. If not specified, libraries in the *link_path* are searched.

selection_group_name

Name of the selection group. This must be a valid selection group in a library specified in **link_path**.

object_list

Provides a list of hierarchical cells or designs. If you do not specify *object_list*, the wire load selection group is set on the current instance, or on the current design if current instance is not set.

DESCRIPTION

Specifies the name of the selection group used when determining the wire load model based on cell area of blocks, when **auto_wire_load_selection** is true. This option is required only when the main library has more than one selection group defined, and the default group defined in the library is not the desired group. Automatic wire load selection is supported only for the *enclosed* wire load mode, specified using the **set_wire_load_mode** command.

To remove the wire load selection group setting, use **remove_wire_load_selection_group**.

EXAMPLES

This example specifies that the selection group named "selgrp1" from library "tech_lib" be applied to the current design.

```
pt_shell> set_wire_load_selection_group selgrp1 -library tech_lib
```

SEE ALSO

```
remove_wire_load_selection_group(2), report_wire_load(2), report_lib(2),  
set_wire_load_min_block_size(2), set_wire_load_mode(2), set_wire_load_model(2),  
auto_wire_load_selection(3), link_path(3).
```

size_cell

Relinks leaf cells to a new library cells that have the required drive strength (or other properties).

SYNTAX

```
int size_cell [-current_library] [-libraries lib_spec] cell_list lib_cell
```

```
list lib_spec  
list cell_list  
string lib_cell
```

ARGUMENTS

-current_library

If this option is specified, then PrimeTime searches for library cells in the cells' current libraries only. You cannot specify this option with the **-libraries** option. You cannot specify this option if a full library cell name has been specified.

-libraries *lib_spec*

If this option is specified, then PrimeTime resolves library cells from the libraries contained in the *lib_spec* only. Libraries are searched in the order in which they appear in *lib_spec*. *lib_spec* can be a list of library names, or collections of libraries loaded into PrimeTime; the latter can be obtained using the **get_libs** command. You cannot specify this option with the **-current_library** option. You cannot specify this option if a full library cell name has been specified.

cell_list

Specifies a list of leaf cells to be relinked. Each cell must be in scope (at or below the current instance).

lib_cell

Specifies the name of the library cell to which the specified cells are to be linked. *lib_cell* can be a library cell object, or the name of a library cell. The former can be obtained using the **get_lib_cells** command. The latter can either be the full library cell name, i.e. '*lib_name/lcell_name*', or the just the base name of the library cell, i.e. '*lcell_name*'. You cannot specify either the **-current_library** or the **-libraries** options and explicitly specify the full library cell name. If the user invokes PrimeTime with the **-multi_scenario** option, then the library cell base name only must be used. For more information, see the section entitled "RESOLVING LIBRARY CELLS".

DESCRIPTION

The **size_cell** command changes the drive strength (or other properties) of a leaf cell by relinking it to a new library cell that has the required properties. Like all other netlist editing commands, for **size_cell** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **size_cell** returns a 1 if successful and a 0 if unsuccessful. Each cell in *cell_list* must be in

scope; that is, at or below the current instance.

You can get a list of library cells that are compatible with a given cell using the **get_alternative_lib_cells** command, or report on the effects of compatible library cells using the **report_alternative_lib_cells** command.

The **size_cell** command is for leaf cells only. To swap a hierarchical block for a different design, you must use the **swap_cell** command.

Executing **size_cell** typically triggers an incremental timing update. However, a full timing update is required if the resized cell is in the clock tree or connected to the clock tree.

EQUIVALENT LIBRARY CELLS

Two library cells are compatible for sizing if all of the following restrictions are satisfied. The exact same set of restrictions applies to the **size_cell**, **get_alternative_lib_cells** and **report_alternative_lib_cells** commands.

1) The library cells must be functionally equivalent. At least one of the following must be true, and none can be false:

- If defined, the **function_id** attribute must match. This attribute is created by Library Compiler and is available on **lib_cell** objects.

- If defined, the **mog_func_id** attribute must match. This attribute is created by Library Compiler for multiple output gate (MOG) cells and is available on **lib_cell** objects.

- If defined, the **user_function_class** attribute must match. This is a user-defined functional equivalency attribute for cells that are too complex for Library Compiler to compute functional information for. This attribute is available on **lib_cell** objects.

2) The number of pins, ordering of pins, pin directions, and internal pin names must match.

3) The number of enabled library cell arcs, arc types and sense must match, including any conditional or "when" expressions.

RESOLVING LIBRARY CELLS

If the *lib_cell* has been specified in base name only format, i.e. without a library from which to resolve it from, then PrimeTime resolves the library cell according to the following methodology.

If the **-current_library** option is specified, then PrimeTime searches for library cells in the cells' current libraries only. If the **-libraries** option is specified, then PrimeTime searches for library cells in the libraries contained within the *lib_spec* only.

Alternatively, if neither of the above options are specified, PrimeTime searches for the library cell in the following sources in this order:

- 1) The cell's current library
- 2) The **link_path_per_instance** variable
- 3) The **link_path** variable. If a library is found with a min library relationship (from **set_min_library**), the min library is automatically included.

The first library cell that is found is used.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'il/low' (low) as 'low_0'.
Uniquifying 'il' (inter) as 'inter_0'.
Sized 'il/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is *true* after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

In the following example, an attempt to size a cell fails because the target is not functionally equivalent. After finding a compatible library cell, the second attempt succeeds.

```
pt_shell> size_cell o_reg1 class/NR4P
Error: Could not size 'o_reg1' ('class/FD2') with 'class/NR4P':
Cells not functionally equivalent. (NED-005)
Error: No changes made. (NED-040)
0
```

```
pt_shell> get_alternative_lib_cells o_reg1
{"class/FD2P"}
```

```
pt_shell> size_cell o_reg1 class/FD2P
Information: Sized 'o_reg1' with 'class/FD2P'. (NED-045)
1
```

In the following example, cells are sized to alternative library cells using the cells' current libraries and the library cell base name only.

```
pt_shell> get_lib_cells -of_objects n1
{"class/ND2"}
```

```
pt_shell> get_alternative_lib_cells -current_library n1
{"class/ND2P"}
```

```
pt_shell> get_lib_cells -of_objects n2
{"libA/ND2"}
```

```
pt_shell> get_alternative_lib_cells -current_library n2
{"libA/ND2P"}
```

```
pt_shell> size_cell -current_library {n1 n2} ND2P
Information: Sized 'n1' with 'class/ND2P'. (NED-045)
Information: Sized 'n2' with 'libA/ND2P'. (NED-045)
1
```


In the following example, cells are sized to alternative library cells using user specified libraries and the library cell base name only. Assume that there are no 'ND2P' library cells in 'lib1'. The 'lib1' library is searched first as it appears in the *lib_spec* list first, then 'lib2' is searched.

```
pt_shell> size_cell -libraries {lib1 lib2} {n1 n2} ND2P
```

```
Information: Sized 'n1' with 'lib2/ND2P'. (NED-045)
```

```
Information: Sized 'n2' with 'lib2/ND2P'. (NED-045)
```

```
1
```

SEE ALSO

```
get_alternative_lib_cells (2), report_alternative_lib_cells (2), swap_cell (2),  
write_changes (2), link_path_per_instance (3), link_path (3).
```

sizeof_collection

Returns the number of objects in a collection.

SYNTAX

```
int sizeof_collection collection1
collection collection1
```

ARGUMENTS

collection1
Specifies the collection for which to get the number of objects. If the empty collection (empty string) is used for the *collection1* argument, the command returns 0.

DESCRIPTION

The **sizeof_collection** command is an efficient mechanism for determining the number of objects in a collection.

EXAMPLES

The following example shows a simple way to find out how many objects matched a particular pattern and filter in a **get_cells** command.

```
pt_shell> echo "Number of hierarchical cells: \  
? [sizeof_collection [get_cells * -hier \  
? -filter "is_hierarchical == true"]]"
```

Number of hierarchical cells is: 10

The following example shows what happens when the argument to **sizeof_collection** results in an empty collection.

```
pt_shell> set s1 [get_cells *]  
{"u1", "u2", "u3"}  
pt_shell> set ssize [filter_collection $s1 "area < 0"]  
pt_shell> echo "Cells with area < 0: [sizeof_collection $ssize]"  
Cells with area < 0: 0  
pt_shell> unset s1
```

SEE ALSO

collections (2), **filter_collection** (2).

sort_collection

Sorts a collection based on one or more attributes, resulting in a new, sorted collection. The sort is ascending by default.

SYNTAX

```
collection sort_collection [-descending] collection1 criteria  
collection collection1  
list criteria
```

ARGUMENTS

-descending

Indicates that the collection is to be sorted in reverse order. By default, the sort proceeds in ascending order.

collection1

Specifies the collection to be sorted.

criteria

Specifies a list of one or more application or user-defined attributes to use as sort keys.

DESCRIPTION

You can use **sort_collection** to order the objects in a collection based on one or more attributes. For example, to get a collection of leaf cells increasing alphabetically, followed by hierarchical cells increasing alphabetically, sort the collection of cells using the **is_hierarchical** and **full_name** attributes as *criteria*.

In an ascending sort, Boolean attributes are sorted with those objects first that have the attribute set to *false*, followed by the objects that have the attribute set to *true*. In the case of a sparse attribute, objects that have the attribute come first, followed by the objects that do not have the attribute.

Sorts are ascending by default. The **-descending** option reverses the order of the objects.

EXAMPLES

The following example sorts a collection of cells based on hierarchy, and adds a second key to list them alphabetically. In this example, cells i1 and i2 are hierarchical, and u1 and u2 are leaf cells. Because **is_hierarchical** is a Boolean attribute, those objects with the attribute set to *false* are listed first in the sorted collection.

```
pt_shell> set zcells [get_cells {o2 i2 o1 i1}]  
{"o1", "i2", "o1", "i1"}  
pt_shell> set zsort [sort_collection $zc {is_hierarchical full_name}]  
{"o1", "o2", "i1", "i2"}
```

SEE ALSO

`collections` (2).

start_gui

Starts the Primetime GUI.

SYNTAX

```
string start_gui

        [-file name_of_script_file]
        [-no_windows]
string name_of_script_file
```

ARGUMENTS

```
-file name_of_script_file
    The given script file is sourced before the GUI starts.

-no_windows
    The GUI starts without showing the default window.
```

DESCRIPTION

This command starts the Primetime GUI from the `pt_shell` prompt. It if the Primetime GUI has already been started.

EXAMPLES

The following example starts the Primetime GUI.

```
pt_shell> start_gui
```

SEE ALSO

stop_gui (2).

start_hosts

Start the hosts specified by the **set_host_options** command.

SYNTAX

```
int start_hosts  
[-timeout seconds]  
[-min_hosts num_hosts]
```

integer *seconds*
integer *number*

ARGUMENTS

-timeout *seconds*

Specifies, in seconds, how long the **start_hosts** command will wait for remote hosts to come online. While the command is waiting, no other commands can be executed at the master. If all the remote hosts come online before the timeout has expired, the command will return immediately on connection of the final host. If all the remote hosts do not come online before the timeout has expired, the command will return and accept the remainder of the hosts online in the background. The value specified must be greater than or equal to zero. If the option is not specified, the default value of 21600s (6 hours) is used.

-min_hosts *num_hosts*

Specifies the minimum number of hosts that the **start_hosts** command will wait for to come online. While the command is waiting, no other commands can be executed at the master. As soon as the minimum number of hosts has come online, the command will return immediately and allow commands to execute at the master. The -timeout option to the command overrides the -min_hosts option so if the minimum number of hosts does not come online before the timeout has expired, then the command will be governed by the behaviour of the -timeout option as described above. The value specified must be greater than or equal to zero. If the option is not specified, by default the command will wait for all hosts launched to come online.

DESCRIPTION

This command explicitly starts the hosts to be used as compute resources. Host options must first be setup using the **set_host_options** command. If the hosts are not explicitly started using this command, they are started during execution of the **update_timing** command.

When hosts are started they end up in one of the following states

LAUNCHED : The process is launched but not ready for use yet.
ONLINE : The process is online and ready for use.
SHUTDOWN : The process was shut down.
FATAled : The process abnormally terminated.
MISSALIGNED : The master a slave processes are different versions
of PrimeTime and the slave process is not usable.

EXAMPLES

In the following example, the master process is running on the host named ptopt018 using a 64-bit binary and specifies five different sets of host options. The first host options, named "my_opts1", specifies 1 process with the same architecture as the master on the same host as the master. The second host options, named "my_opts2", specifies 2 processes using the 32-bit binary on the same host as the master but explicitly mentions the name of the master's host. The third host options, named "my_opts3", specifies 4 processes (with the same architecture as the master) on the host named ptopt010 using rsh to connect to ptopt010. The fourth host options named "my_opts4", specifies 5 processes (with the same architecture as the master) on an LSF farm requesting 500MB of memory on the compute servers. The fifth host options, named "my_opts5", specifies 2 processes (with the same architecture as the master) on a Grid farm requiring the jobs to be launched using the project named bnormal.

Call the **set_host_options** command to specify the host options.

```
pt_shell> set_host_options -name my_opts1
1
pt_shell> set_host_options -name my_opts2 -32bit -num_processes 2 \
ptopt018
1
pt_shell> set_host_options -name my_opts3 -num_processes 4 \
-submit_command "/usr/bin/rsh -n" ptopt010
1
pt_shell> set_host_options -name my_opts4 -num_processes 5 \
-submit_command "/lsf/bin/bsub -R rusage\[mem=500\]"
1
pt_shell> set_host_options -name my_opts5 -num_processes 2 \
-submit_command "/grid/bin/qsub -b y -P bnormal"
1
```

Call the **start_hosts** command to start the slave processes. Notice that each instance launched is assigned a number e.g. 1] Launched ...

```
pt_shell> start_hosts
1] Launched : ...
...
Status : Forking successful
Stdout : Process group is (7199)
Stderr : **<<EMPTY>>**

2] Launched : ...
...
Status : Forking successful
Stdout : Process group is (7233)
Stderr : **<<EMPTY>>**

3] Launched : ...
```

```

        ...
        Status : Forking successful
        Stdout  : Process group is (7269)
        Stderr  : **<<EMPTY>>**

4] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Process group is (8481)
        Stderr  : **<<EMPTY>>**

5] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Process group is (8558)
        Stderr  : **<<EMPTY>>**

6] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Process group is (8629)
        Stderr  : **<<EMPTY>>**

7] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Process group is (8716)
        Stderr  : **<<EMPTY>>**

8] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Job <627124> is submitted to default queue <normal>.
        Stderr  : **<<EMPTY>>**

9] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Job <627125> is submitted to default queue <normal>.
        Stderr  : **<<EMPTY>>**

10] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Job <627126> is submitted to default queue <normal>.
        Stderr  : **<<EMPTY>>**

11] Launched : ...
        ...
        Status : Forking successful
        Stdout  : Job <627127> is submitted to default queue <normal>.
        Stderr  : **<<EMPTY>>**

12] Launched : ...
        ...

```



```

Status : Forking successful
Stdout  : Job <627128> is submitted to default queue <normal>.
Stderr  : **<<EMPTY>>**

```

```
13] Launched : ...
```

```

...
Status : Forking successful
Stdout  : Your job 8998610 ("pt_shell") has been submitted
Stderr  : **<<EMPTY>>**

```

```
14] Launched : ...
```

```

...
Status : Forking successful
Stdout  : Your job 8998611 ("pt_shell") has been submitted
Stderr  : **<<EMPTY>>**

```

```

-----
Distributed farm creation timeout : 21600 seconds
Waiting for 14 (of 14) distributed hosts (Wed Jul 29 09:07:07 2009)
Waiting for 9 (of 14) distributed hosts (Wed Jul 29 09:07:17 2009)
Waiting for 0 (of 14) distributed hosts (Wed Jul 29 09:07:27 2009)
-----

```

1

Call the **report_host_usage** command with the **-verbose** option after starting the hosts to report the host options created by the **set_host_options** command, as well as the real time data associated with the processes.

```

pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:07:42 2009
*****

```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	#	Host Name	Job ID	Process ID	Status
my_opts1	1	ptopt018	7199	7199	ONLINE
my_opts2	2	ptopt018	7233	7233	ONLINE

	3	ptopt018	7269	7269	ONLINE
my_opts3	4	ptopt010	8481	8481	ONLINE
	5	ptopt010	8558	8558	ONLINE
	6	ptopt010	8629	8629	ONLINE
	7	ptopt010	8716	8716	ONLINE
my_opts4	8	maddog108	627124	7718	ONLINE
	9	maddog108	627125	7715	ONLINE
	10	maddog108	627126	7717	ONLINE
	11	maddog108	627127	7716	ONLINE
	12	maddog108	627128	7714	ONLINE
my_opts5	13	peopf11	8998610	30080	ONLINE
	14	peopf1	8998611	11076	ONLINE

Options Name	Action	Command
my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

SEE ALSO

stop_hosts (2), **set_host_options** (2), **remove_host_options** (2), **report_host_usage** (2)

start_profile

Start profiling PrimeTime commands.

SYNTAX

Boolean **start_profile**

ARGUMENTS

DESCRIPTION

The **start_profile** command can be used to perform command level profiling in PrimeTime. You should use this command in conjunction with **stop_profile** and **write_profile** to write profiling information for PrimeTime scripts. Please send the profiling file to Synopsys for recommendations.

For every PrimeTime command or procedure invocation, the cpu time, elapsed time, delta memory and number of calls are registered. Two commands or procedure invocations at a given scope level are collapsed into one and their cpu times, elapsed times and delta memory are added and the number of calls is increased by one.

PrimeTime profile handles **source** command in a special manner. Since source can be used to arbitrarily call deep levels of scripts, source commands also show file names. Hence, for source commands, even if two source commands are executed at same scope level, they will usually have two different entries unless they source the exact same file.

This command succeeds all the time except when the profiling is already active which means there is a previous **start_profile** that has not been stopped yet using **stop_profile** command.

EXAMPLES

The following gives an example of profiling

```
pt_shell> start_profile
pt_shell> ....    # Usual PT script
pt_shell> stop_profile
pt_shell> write_profile prof.data
```

SEE ALSO

stop_profile (2), **write_profile** (2).

stop_gui

Stops the Primetime GUI.

SYNTAX

string **stop_gui**

ARGUMENTS

None.

DESCRIPTION

This command stops the Primetime GUI and returns to the pt_shell prompt. It is ignored within scripts, and at the pt_shell prompt if the Primetime GUI has not been started.

EXAMPLES

The following example stops the Primetime GUI and returns to the pt_shell prompt.

```
Primetime> stop_gui
```

SEE ALSO

start_gui (2).

stop_hosts

Stops all hosts that have been started.

SYNTAX

```
int stop_hosts
```

DESCRIPTION

The **stop_hosts** command shuts down all hosts that have been started. The hosts could have been explicitly launched using the **start_hosts** command or implicitly launched during the execution of the **update_timing** command.

EXAMPLES

In the following example, the master process is running on the host named ptopt018 using a 64-bit binary and specifies five different sets of host options. The first host options, named "my_opts1", specifies 1 process with the same architecture as the master on the same host as the master. The second host options, named "my_opts2", specifies 2 processes using the 32-bit binary on the same host as the master but explicitly mentions the name of the master's host. The third host options, named "my_opts3", specifies 4 processes (with the same architecture as the master) on the host named ptopt010 using rsh to connect to ptopt010. The fourth host options named "my_opts4", specifies 5 processes (with the same architecture as the master) on an LSF farm requesting 500MB of memory on the compute servers. The fifth host options, named "my_opts5", specifies 2 processes (with the same architecture as the master) on a Grid farm requiring the jobs to be launched using the project named bnormal.

Note: Before calling the **stop_hosts** command, all the processes report as ONLINE and after calling the **stop_hosts** command all the processes report as SHUTDOWN. The order in which the hosts shutdown is random.

Call the **set_host_options** command to specify the host options.

```
pt_shell> set_host_options -name my_opts1
1
pt_shell> set_host_options -name my_opts2 -32bit -num_processes 2 \
ptopt018
1
pt_shell> set_host_options -name my_opts3 -num_processes 4 \
-submit_command "/usr/bin/rsh -n" ptopt010
1
pt_shell> set_host_options -name my_opts4 -num_processes 5 \
-submit_command "/lsf/bin/bsub -R rusage\[mem=500\]"
1
pt_shell> set_host_options -name my_opts5 -num_processes 2 \
-submit_command "/grid/bin/qsub -b y -P bnormal"
1
```

Call the **start_hosts** command to start the slave processes.

```
pt_shell> start_hosts
1] Launched : ...
    ...
    Status : Forking successful
    Stdout  : Process group is (7199)
    Stderr  : **<<EMPTY>>**

2] Launched : ...
    ...
    Status : Forking successful
    Stdout  : Process group is (7233)
    Stderr  : **<<EMPTY>>**

...
...
```

```
-----
Distributed farm creation timeout : 21600 seconds
Waiting for 14 (of 14) distributed hosts (Wed Jul 29 09:07:07 2009)
Waiting for 9 (of 14) distributed hosts (Wed Jul 29 09:07:17 2009)
Waiting for 0 (of 14) distributed hosts (Wed Jul 29 09:07:27 2009)
-----
```

1

Call the **report_host_usage** command with the **-verbose** option after starting the hosts to report the host options created by the **set_host_options** command, as well as the real time data associated with the processes.

```
pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:07:42 2009
*****
```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	#	Host Name	Job ID	Process ID	Status

my_opts1	1	ptopt018	7199	7199	ONLINE
my_opts2	2	ptopt018	7233	7233	ONLINE
	3	ptopt018	7269	7269	ONLINE
my_opts3	4	ptopt010	8481	8481	ONLINE
	5	ptopt010	8558	8558	ONLINE
	6	ptopt010	8629	8629	ONLINE
	7	ptopt010	8716	8716	ONLINE
my_opts4	8	maddog108	627124	7718	ONLINE
	9	maddog108	627125	7715	ONLINE
	10	maddog108	627126	7717	ONLINE
	11	maddog108	627127	7716	ONLINE
	12	maddog108	627128	7714	ONLINE
my_opts5	13	peopf11	8998610	30080	ONLINE
	14	peopf1	8998611	11076	ONLINE

Options Name	Action	Command

my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

Call the **stop_hosts** command so shutdown the processes.

```
pt_shell> stop_hosts
Shutting down all slaves processes ...
Shutdown Process 3
Shutdown Process 1
Shutdown Process 2
Shutdown Process 9
Shutdown Process 5
Shutdown Process 4
Shutdown Process 6
Shutdown Process 8
Shutdown Process 7
Shutdown Process 10
Shutdown Process 12
Shutdown Process 11
Shutdown Process 14
Shutdown Process 13
```

1

Call the **report_host_usage** command with the **-verbose** option after starting the hosts to report the host options created by the **set_host_options** command, as well as the real time data associated with the processes. Notice that the job and process ID are now -1 and the status is SHUTDOWN.

```

pt_shell> report_host_usage -verbose
*****
Report : host_usage
        -verbose
Version: D-2009.12
Date   : Wed Jul 29 09:42:08 2009
*****

```

Options Name	Host	32Bit	Num Processes
my_opts1	**localhost**	N	1
my_opts2	**ptopt018**	Y	2
my_opts3	ptopt010	N	4
my_opts4	>>farm<<	N	5
my_opts5	>>farm<<	N	2

Options Name	#	Host Name	Job ID	Process ID	Status
my_opts1	1		-1	-1	SHUTDOWN
my_opts2	2		-1	-1	SHUTDOWN
	3		-1	-1	SHUTDOWN
my_opts3	4		-1	-1	SHUTDOWN
	5		-1	-1	SHUTDOWN
	6		-1	-1	SHUTDOWN
	7		-1	-1	SHUTDOWN
my_opts4	8		-1	-1	SHUTDOWN
	9		-1	-1	SHUTDOWN
	10		-1	-1	SHUTDOWN
	11		-1	-1	SHUTDOWN
	12		-1	-1	SHUTDOWN
my_opts5	13		-1	-1	SHUTDOWN
	14		-1	-1	SHUTDOWN

Options Name	Action	Command
my_opts1	SUBMIT	<execute directly on masters host>
my_opts2	SUBMIT	<execute directly on masters host>
my_opts3	SUBMIT	/usr/bin/rsh -n
my_opts4	SUBMIT	/lsf/bin/bsub -R rusage[mem=500]
my_opts5	SUBMIT	/grid/bin/qsub -b y -P bnormal

1

SEE ALSO

start_hosts (2), **set_host_options** (2), **remove_host_options** (2), **report_host_usage** (2)

stop_profile

Stop profiling PrimeTime commands.

SYNTAX

boolean **stop_profile**

ARGUMENTS

DESCRIPTION

The **stop_profile** command can be used to stop the command level profiling in PrimeTime that you started previously using **start_profile** command. You should use this command in conjunction with **start_profile** and **write_profile** to write out profiling information for PrimeTime scripts.

Stopping a profile simply stops collecting information about commands for profiling. It does not delete previously collected profiling information. That enables you to do **write_profile** after stopping profiling.

This command succeeds all the time except when the profiling is not currently active.

EXAMPLES

The following gives an example of profiling

```
pt_shell> start_profile
pt_shell> ....    # Usual PT script
pt_shell> stop_profile
pt_shell> write_profile prof.data
```

SEE ALSO

start_profile (2), **write_profile** (2).

sub_variation

Subtracts one variation from another. Returns a collection (that corresponds to this difference variation).

SYNTAX

```
collection sub_variation  
variation_list  
collection variation_list
```

ARGUMENTS

```
variation_list
```

List of the variations to be subtracted. The list must contain two variations.
The resulting operation is the first element in the list minus the second.

DESCRIPTION

Enables the subtraction of two variations. This difference is a variation. The command creates this subtraction variation and returns it as a collection.

EXAMPLES

The following example computes \$v1 - \$v2 (assuming variations \$v1 and \$v2 have already been created) and saves it into collection \$vsub, which is a new variation collection.

```
pt_shell> set col [add_to_collection $v1 $v2]  
_sel128  
pt_shell> set vsub [sub_variation $col]  
_sel129
```

SEE ALSO

create_distribution (2), **sub_variation** (2), **max_variation** (2), **min_variation** (2).

swap_cell

Swaps one or more cells with a new design or library cell.

SYNTAX

```
int swap_cell cell_list swap_in
```

```
[-dont_preserve_constraints]
```

```
[-file file_name]
```

```
[-format file_format]
```

```
list cell_list
```

```
string swap_in
```

```
string file_name
```

```
string file_format
```

ARGUMENTS

cell_list

Specifies a list of cells to be swapped out.

swap_in

Specifies the name of the design or library cell to be swapped in.

-dont_preserve_constraints

Indicates that **swap_cell** is not to reapply the current design constraints after the swap.

-file *file_name*

Specifies the name of a file that contains a design that is to be swapped in.

-format *file_format*

Specifies the format for *file_name*. Allowed values are *db* (the default), *Verilog*, *EDIF*, and *VHDL*.

DESCRIPTION

The **swap_cell** command replaces the cells in *cell_list* with *swap_in* (a new design or library cell), and incrementally relinks the design. The command checks to ensure that the pinouts of the swapped-in and swapped-out cells are the same. For example, every boundary pin of the cell being swapped out must have a counterpart with the same name in the cell being swapped in, and vice versa.

The **swap_cell** command is considered a netlist editing command. Like all other netlist editing commands, for **swap_cell** to succeed, all of its arguments must succeed. If any arguments fail, the netlist remains unchanged. **swap_cell** returns a 1 if successful and a 0 if unsuccessful.

If you want to swap a library cell for another library cell that is just a different size, use **size_cell** instead of **swap_cell**. **size_cell** is more efficient than **swap_cell**, and is optimized for incremental timing updates.

Library cells being swapped in must be in memory. Designs being swapped in can be in memory, or can be retrieved from a file using the **-file** and **-format** options. For details, see the section entitled "Interaction of Reading Netlists and swap_cell".

By default, design constraints are preserved by writing a script to the directory referenced by the **pt_tmp_dir** variable, then loading the script after the swap has completed. Use the **-dont_preserve_constraints** option only if you plan to perform several **swap_cell** commands; for example, in a loop. In this case, it is much more efficient to use **write_script**, then **swap_cell -dont_preserve_constraints**, then **source** your script.

If a swapped cell is in the parent chain of any objects that are elements of a saved collection, these collections may be deleted based on the **collection_deletion_effort** variable. For details, see the manual pages for **collections** and associated variables.

The **swap_cell** command does not create black boxes. If the design that is swapped in contains unresolved references, the result of the swap is a partially linked design.

Interaction of Reading Netlists and swap_cell

The best way to swap in a design that is not in memory is to use the **-file** and **-format** options. When you load the new design this way, **swap_cell** maintains the current design, and the swap is a one-step operation. Explicitly reading a new design prior to swapping cells changes the current design, so you must remember to reset the current design back to the previous current design before using **swap_cell**. Thus, it is preferable to use the **-file** and **-format** options. Both alternatives are illustrated in the EXAMPLES section.

Interaction of Linking and swap_cell

Do not perform an explicit link after executing **swap_cell**. The **swap_cell** command implicitly relinks the part of the design you have changed; therefore, performing an explicit link after executing **swap_cell** can undo some or all of the work done by **swap_cell**.

For example, consider U2, an instance of design X, which contains instance U0, an instance of design Y. If you swap out U2/U0 for a different design, PrimeTime has modified the *instance* U2/U0; it has not modified the design Y. Therefore, an initial link causes U2/U0 to revert to the original Y (the version of Y inferred by the **link_path**).

PrimeTime does not save information about cells that have been swapped. That information persists only until the next link.

Managing Changes

The changes caused by netlist editing commands are recorded in a *change list*, which is annotated on the design. The contents of the change list can be displayed or exported using the **write_changes** command. The script output formats for PrimeTime and Design Compiler are the most useful. For details, see the **write_changes** manual page.

Edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with **swap_cell**. However, edits to the top level of a design cannot be undone without removing the design, re-reading the original design, and relinking it.

Uniquification

Uniquification is the process of making each hierarchical block unique in a set of multiple instances of a design. For example, if the block is an instance of BLOCK, PrimeTime searches for a design name (BLOCK_0, BLOCK_1, and so on), until an unused name is found. That becomes the name of the unique design. Uniquification in PrimeTime is somewhat different from uniquification in other Synopsys applications, in that new designs are not created at the time that uniquification occurs. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by **link_design -remove_sub_designs**. Uniquified designs can be listed using **list_designs -all**. They become real designs only when that becomes necessary; for example, when the design is re-linked.

PrimeTime's netlist editing user model provides auto-uniquification, which uniquifies on demand only that which is necessary to uniquify: the block being edited and all blocks above it up to the top of the hierarchy. A block that is an instance of a design with multiple instances becomes unique when you edit it. Messages are generated when uniquification occurs.

The following example shows the **size_cell** command being used on a cell and causing uniquification, which is announced by the messages.

```
pt_shell> size_cell n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, and so on, up the chain to the design, which itself has been edited. This information is available through a Boolean attribute, **is_edited**, available on a design and on hierarchical cells. The following example shows that the value of the **is_edited** attribute on i1/low is **true** after the **size_cell** command was used in the previous example.

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

EXAMPLES

The following example shows the optimal method for swapping in a design from a file. The design "A" is in "A.db". The **swap_cell** command replaces cells u1, u2, and u3 in TOP with the design A (in A.db).

```
pt_shell> current_design
```

```

{"TOP"}
pt_shell> swap_cell {u1 u2 u3} A -file A.db
Loading db file '/usr/designs/A.db'
1
pt_shell> current_design
{"TOP"}

```

The following example accomplishes the same task as the previous example, but shows the method where the design file is read explicitly.

```

pt_shell> current_design
{"TOP"}
pt_shell> read_db A.db
Loading db file '/usr/designs/A.db'
1
pt_shell> current_design
{"A"}
pt_shell> current_design TOP
{"TOP"}
pt_shell> swap_cell {u1 u2 u3} A.db:A
1

```

The following example replaces cells u1 and u2 in TOP with the lib_cell misc_cmos/LC3.

```

pt_shell> swap_cell {u1 u2} [get_lib_cells misc_cmos/LC3]

```

SEE ALSO

collections (2), **current_design** (2), **link_design** (2), **size_cell** (2), **source** (2), **write_changes** (2), **write_script** (2); **collection_deletion_effort** (3), **link_path** (3), **pt_tmp_dir** (3).

transform_exceptions

Performs transformation on timing exceptions.

SYNTAX

Boolean **transform_exceptions** [-from *from_list* | -rise_from *rise_from_list* | -fall_from *fall_from_list*] [-through *through_list* | -rise_through *rise_through_list* | -fall_through *fall_through_list*] [-to *to_list*] | -rise_to *rise_to_list* | -fall_to *fall_to_list*] [-verbose] [-dry_run] [-remove_ignored *reason*] [-use_to_for_endpoints] [-flatten]

```
list  from_list
list  rise_from_list
list  fall_from_list
list  through_list
list  rise_through_list
list  fall_through_list
list  rise_to_list
list  fall_to_list
list  to_list
```

ARGUMENTS

-from *from_list*

Specifies a list of clocks, ports, cells, and pins in the current design. This list must specify valid path start points, such as primary inputs or register clock pins. The **-from**, **-rise_from**, and **-fall_from** options are mutually exclusive; you can use only one.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. The **-from**, **-rise_from**, and **-fall_from** options are mutually exclusive; you can use only one.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. The **-from**, **-rise_from**, and **-fall_from** options are mutually exclusive; you can use only one.

-through *through_list*

Specifies a list of pins or ports. The **-through**, **-rise_through**, and **-fall_through** options are mutually exclusive; you can use only one.

-rise_through *rise_through_list*

Specifies the same list as the **-through** option does, but applies only to paths that have a rising transition at the through points. The **-through**, -

rise_through, and **-fall_through** options are mutually exclusive; you can use only one.

-fall_through *fall_through_list*

Specifies the same list as the **-through** option does, but applies only to paths that have a falling transition at the through points. The **-through**, **-rise_through**, and **-fall_through** options are mutually exclusive; you can use only one.

-to *to_list*

Specifies a list of clocks, ports, cells, and pins in the current design. This list must specify valid path end points, such as primary outputs or register D-pins. The **-to**, **-rise_to**, and **-fall_to** options are mutually exclusive; you can use only one.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. The **-to**, **-rise_to**, and **-fall_to** options are mutually exclusive; you can use only one.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. The **-to**, **-rise_to**, and **-fall_to** options are mutually exclusive; you can use only one.

-verbose

Provides a detailed output of each exception that exists on the set of paths specified by using the path selectors or, if none are specified, by using the entire design. Identifies whether the exception is completely valid, partially ignored, or unignored in the context of the selected paths. Provides justification for ignoring exception paths from the following list:

- An exception path has an invalid start or end point.
- The multi-cycle path exception specifies single cycle values.
- There is a nonexistent path between exception specifiers.
- The exception is set on an unconstrained path.
- The exception specifies a path in the clock network.
- The exception is overridden by a higher precedence exception.
- The exception is overridden because of mode analysis.
- The exception is overridden due to exclusive clock groups.

-dry_run

Allows analysis and reporting, but precludes replacing the original exceptions with the transformed ones.

-remove_ignored *reason*

Enables exception compaction to reduce ignored exceptions or trim exception specifiers to remove paths that ignore exception. Valid values for *reason* are **invalid_specifiers**, **no_path**, **unconstrained_path**, **clock_path**, and **overridden**.

-use_to_for_endpoints

Causes path endpoints specified using "-through*" to use "-to*" instead. This option fixes the transformation context of the command and hence cannot be used in conjunction with *-remove_ignored* or *-flatten*.

-flatten

Converts user user-compressed exceptions into simple exceptions such that each exception specifier (*-rise_from*, *-fall_from*, *-from*, *-rise_through*, *-fall_through*, *-through*, *-rise_to*, *-fall_to*, *-to*) matches a single object. This option fixes the transformation context of the command and hence cannot be used in conjunction with *-remove_ignored* or *-use_to_for_endpoints*.

DESCRIPTION

Performs transformation on timing exceptions to determine the usage of each exception and interference between exceptions across the specified paths in the design. By default, the behavior of this command is to cleanup exceptions to reduce ignored exception specifiers within the context of specified design paths. The command outputs statistics about the processed set of exceptions. The *-verbose* option expands the output display usage and justification information about each exception processed.

The command can be run for reporting purposes only by using the *-dry_run* option. While all required analysis would still be performed as for the default case, the transformations are disregarded when the command completes.

To generate a file containing the new set of exceptions, first execute the command with the options you want, excluding the *-dry_run* option. Next, use the *write_script* or *write_sdc* commands to emit design constraints and context. Use the *-include* option with either *write_script* or *write_sdc* to emit exceptions only if you want them emitted.

EXAMPLES

The following example performs exceptions transformations on all timing exceptions set on the design.

```
pt_shell> transform_exceptions
```

```
*****
```

```
Report : transform_exceptions
```

```
Design : CORE
```

```
Version: T-2002.09-Alpha1
```

```
Date   : Mon Jun 24 11:23:03 2002
```

```
*****
```

```
Transformations Summary  false  multicycle  delay  Total
```

```
-----  
invalid specifiers      5      10          1      16  
non-existent paths      1       3          0       4  
clock network paths     0       1          0       1  
unconstrained paths     1       0          0       1  
overridden              0       2          1       3  
-----
```

Total transformations 25

Exceptions Summary	false	multicycle	delay	Total
Removed	0	2	1	3
Modified	5	9	0	14
Analyzed	7	16	1	24

The following example performs selective transformation on a specified set of paths without committing the final exceptions.

```
pt_shell> transform_exceptions -from {A B} -to D \
-remove_ignored { overridden no_path } -dry_run -verbose
```

Report : transform_exceptions

-verbose

-dry_run

Design : TEST

Version: T-2002.09-Alpha1

Date : Mon Jun 24 11:29:03 2002

From	To	Setup	Hold
{A B}	*	FALSE	FALSE
A	D	cycles=2	cycles=0
OVERRIDDEN by false_path			
-from A			
B	D	cycles=3	cycles=0
OVERRIDDEN by false_path			
-from B			

Transformations Summary	false	multicycle	delay	Total
overridden	0	2	0	2
Total transformations				2

Exceptions Summary	false	multicycle	delay	Total
Analyzed	1	2	0	3

SEE ALSO

report_exceptions (2), **set_clock_groups** (2), **set_false_path** (2), **set_max_delay** (2), **set_min_delay** (2), **set_mode** (2), **set_multicycle_path** (2), **write_script** (2), **write_sdc** (2).

translate_stamp_model

Translates a STAMP model to a LIB format.

SYNTAX

```
string translate_stamp_model -model_file model_file_name -data_file data_file_name -  
output output_file_name  
string model_file_name  
string data_file_name  
string output_file_name
```

ARGUMENTS

```
-model_file model_file_name  
    Specifies the name of the model file to read and translate.  
  
-data_file data_file_name  
    Specifies the name of the data file to read and translate.  
  
-output output_file_name  
    Specifies the root name to use for the generated LIB file. The file is  
    generated as <output_file_name>.lib.
```

DESCRIPTION

This command translates STAMP files directly to a LIB file. This model can be read into PrimeTime and Design Compiler to perform timing analysis. STAMP is a modeling language developed by Synopsys to describe the static timing models of complex blocks such as RAMs and microprocessor cores. STAMP model description consists of a single model file and multiple data files. The model file contains the definition of design ports and timing arcs. The data file contains the arc data (timing numbers) and the port data (capacitance, max_transition, and so on). Both types of data are dependent on the operating condition.

EXAMPLES

The following command translates the STAMP model specified in the sram.mod file and sram.data file and generates the LIB file sram_model.lib.

```
pt_shell> translate_stamp -model_file sram.mod -data_file sram.data -output  
sram_model
```

SEE ALSO

extract_model (2).

unset_rtl_to_gate_name

Specifies a name without RTL to gate name mapping.

SYNTAX

```
int unset_rtl_to_gate_name  
    name
```

ARGUMENTS

name

This is the name of a RTL object that is not changed to gate level by the name mapping process.

DESCRIPTION

If you have done only the RTL simulation for generating backward SAIF file or VCD file, PrimeTime PX tries to do the name mapping to find the gate level objects in the RTL SAIF or VCD file. This command is to forbid this name mapping process applied to the specified RTL objects.

EXAMPLES

The following example provides the unmapping between RTL and gate-level objects. PrimeTime PX just looks for reg_i[1] in the gate level design if it appears in RTL VCD or SAIF.

```
pt_shell> unset_rtl_to_gate_name reg_i[1]
```

SEE ALSO

set_rtl_to_gate_name(2)

update_noise

Performs static crosstalk noise analysis for the current design.

SYNTAX

```
int update_noise [-full]
```

ARGUMENTS

-full

By default, `update_noise` performs the noise analysis only if the design is not up to date for noise analysis. Using **-full**, forces the `update_noise` to perform the noise analysis regardless whether the design is out of date or not.

DESCRIPTION

Updates the crosstalk noise for the current design. The crosstalk noise is also automatically updated by commands or attributes that need the information, such as **report_timing**.

The command **set_noise_parameters** controls the settings that are used during the noise analysis update.

The noise analysis is performed at each stage of the design, by calculating the worst case noise bump that aggressors induce on the victim line when the victim line is not switching. Also, if the **-enable_propagation** option is used with the command **set_noise_parameters**, then during the noise analysis, calculated noise can also propagate to the next stage and combine with the bumps induced by the aggressors of the next stage.

The induced bumps from the aggressors are calculated based on the characteristics of the driver of the victim net. If the **set_steady_state_resistance** is specified on the library cell of the driver cell, that information would be used for the calculation of induced bumps from the aggressors. Otherwise, the steady state current-voltage (I-V) characteristics of the library cell of the driver is used. If the I-V information does not exist in the library, the steady state resistance from the library is used. If none of these are available, the steady state resistance of the victim driver is estimated from the delay and slew tables. The variables **si_noise_pmos_threshold_ratio** and **si_noise_nmos_threshold_ratio**, are during this estimation method.

By default, the analysis of beyond high and low rails are disabled. If **-include_beyond_rails** option is used with the command **set_noise_parameters**, then noise analysis for beyond high and low rail is also performed during the noise update.

Another parameter set by **set_noise_parameters** is the **-analysis_effort**, which by default is set to *high*, but can be set to *low* for a faster but less accurate noise update.

EXAMPLES

The following example updates the noise information for the current design.

```
pt_shell> update_noise
```

SEE ALSO

```
report_noise (2); set_noise_parameters (2); set_steady_state_resistance (2);  
si_noise_effort_threshold_within_rails (3). si_noise_effort_threshold_within_rails  
(3). si_noise_pmos_threshold_ratio (3). si_noise_nmos_threshold_ratio (3).
```

update_power

Updates power information on the current design.

SYNTAX

```
int update_power
```

DESCRIPTION

Updates power for the current design. Power is also automatically updated by command that retrieves power results, such as command **report_power** and most power attributes. Command **update_power** explicitly prepares the design for further analysis. Note that you must set variable **power_enable_analysis** to TRUE before any power command.

Starting from 2008.12 release, **update_power** can also perform power waveform generation and peak power calculation.

Variable **power_analysis_mode** can be used to specify the power analysis mode to perform. PrimeTime PX can perform different types of power analysis based on the mode setting. The default power analysis mode is "averaged". For more information, please check out the man page for **power_analysis_mode**.

Command **set_power_analysis_options** can be used to specify the options for power analysis. It must be set before **update_power** to take effects in power analysis. Issuing command **set_power_analysis_options** with no option can reset all the power analysis options to default. Use command **report_power_analysis_options** to get the current analysis option settings. please check out the man page for **set_power_analysis_options** for more information.

PrimeTime PX can consume either time-based (e.g. VCD file) or statistical switching activity information (e.g. SAIF file) for power calculation. For a particular analysis mode, appropriate activity information must be provided. For example, in time_based power analysis mode, a VCD file must be provided. In averaged power analysis mode, any form of switching activity information is accepted. You can specify a VCD file by using command **read_vcd**. The statistical switching activity can be specified by using command **read_saif** or **set_switching_activity**.

For power estimation using vector free power analysis, all primary inputs and black boxes' pins are assigned with default switching activities. You may use **set_case_analysis** or **set_switching_activity** to improve accuracy. For example, using **set_case_analysis** to set a scan enable pin to 0 gives more accurate average power for normal operating mode. Setting reasonable switching activity values for set/reset pins by using the **set_switching_activity** command is also preferred.

In averaged power analysis mode, only averaged power results will be calculated.

In time_based mode, power waveforms can be generated for the design, specific hierarchies or cells. Power waveforms are series of event-based power data calculated over time while processing VCD activity data. Meanwhile, peak power, which is the largest power value in the waveform, is also captured. The power waveforms are written into waveform files, which can later be displayed by waveform

viewing tools. The peak power is saved and can be reported by **report_power**. Besides generating power waveforms, the averaged power results are also calculated.

If a gate-level zero_delay VCD (indicated by **read_vcd -zero_delay**) or a RTL VCD (indicated by **read_vcd -rtl**) is used in time_based mode, cycle accurate peak power analysis will be performed. In cycle accurate peak power analysis, the default sampling interval is the clock cycle associated with the fastest clock in the design. Within the sampling interval the power waveform is a constant which represents the total power dissipated in that cycle. Cycle accurate peak power analysis provides cycle level accurate power behavior for the design. It can report maximum cycle power and the simulation time of the cycle in which the maximum power happens. The results are shown as the peak power reported by **report_power** command and peak power attributes. Options **-cycle_accurate_clock** and **-cycle_accurate_cycle_count** in command **set_power_analysis_options** can be used to specify the cycle for this analysis.

In leakage_variation power analysis mode, leakage variation analysis will be performed. For more information, please check out the man page for **power_enable_leakage_variation_analysis**.

PrimeTime PX supports UPF domain-based power report feature for multi-Vdd designs. You can use **set_current_power_domain** or **set_current_power_net** commands to set the domain(s) or supply net(s) of interest. Only the power dissipated on the specified domain(s) or supply net(s) are calculated and reported. By default (if not specified), all the domains are included in power analysis. Use **get_current_power_domain** or **get_current_power_net** to show the current settings.

EXAMPLES

This example performs time_based power analysis for time windows (0-20 and 50-end).

```
pt_shell> set power_analysis_mode time_based
pt_shell> read_vcd -time {0 20 50 -1} dump.vcd
pt_shell> update_power
```

This example performs averaged power analysis.

```
pt_shell> set power_analysis_mode averaged
pt_shell> read_saif -instance top -input file.saif
pt_shell> update_power
```

This example performs time_based power analysis for instance u1.u2 from 100 ns to 500 ns. It also dumps out a text file foo.out with time_based power results in .out format.

```
pt_shell> set power_analysis_mode time_based
pt_shell> read_vcd -time {100 500} dump.vcd
pt_shell> set_power_analysis_options -waveform_output foo -waveform_format out -
cells {u1.u2}
pt_shell> update_power
```

The following example performs cycle accurate peak power analysis to generate a cycle accurate waveform by using an RTL VCD file as input.


```
pt_shell> set power_analysis_mode time_based
pt_shell> read_vcd myRTL.vcd -rtl
pt_shell> set_power_analysis_options -waveform_output capp_waveform -
waveform_format out
pt_shell> update_power
```

SEE ALSO

power_enable_analysis(3), power_analysis_mode(3), set_power_analysis_options(2), report_power_analysis_options(2), report_power(2), read_vcd(2), read_saif(2), write_saif(2), set_switching_activity(2), get_switching_activity(2), report_switching_activity(2), set_case_analysis(2), power_enable_leakage_variation_analysis(3), current_power_domain(2), current_power_net(2).

update_scope_data

Updates the scope data captured and stored in the scope file.

SYNTAX

```
int update_scope_data
[-remove_block blk_names]
[-remove_scenario scenario_names]
[-merge_with other_scope_files]
[-keep_last]
file_name

list    blk_names
list    scenario_names
```

ARGUMENTS

-remove_blocks *blk_names*
Specifies the names of the blocks for which the scope data is to be removed from the given scope data file. If no scenario names are specified, by default, scope data for the block in all scenarios are removed.

-remove_scenarios *scenario_names*
Specifies the names of the scenarios for which the scope data is to be removed from the given scope data file. If no block names are specified, by default, scope data for the specified scenarios in all blocks are removed.

-merge_with *other_scope_files*
Specifies all the other scope files from which to load the scope data and merge into the scope_file being updated. The scope_file being updated should not be put in this list.

-keep_last
Specifies the precedence of the data when resolving duplicated scope data during merging. By default, when this option is not specified, the first scope data found takes precedence and is kept in the final data after merge is done. If you want the last loaded scope data to take precedence, specify this option to override the default behavior. Note that the scope_file being updated is always considered the first. All the other scope files specified in option **-merge_with** are treated in the order they are specified and loaded. Note this option should only be used together with **-merge_with**.

DESCRIPTION

This command is used to update the data stored in the binary scope file(s). It provides users with 2 types of mutually exclusive operations to manage the scope data created during block-level analysis: (1) to remove certain scope data that are no longer needed, (2) to merge the scope data stored in other scope files together into one file.

Note that only the file currently being updated is changed after a successful updating operation. No changes are made to the contents of any other files. Also note that the changes to scope file being updated is generally not reversible.

EXAMPLES

This example merge 2 scope files into file 'myBlock.scope'.

```
pt_shell> update_scope_data ./myBlock.scope -merge_with {./1.scope ./2.scope}
```

The following command removes the scope data captured for block 'Block_A' at scenarios "XYZ" and "ABC".

```
pt_shell> update_scope_data myBlock.scope -remove_block Block_A \  
? -remove_scenario {XYZ ABC}
```

SEE ALSO

```
create_ilm (2), extract_model (2), create_scenario (2), check_block_scope (2),  
report_block_scope (2), hier_scope_check_defaults (3).
```

update_timing

Updates timing information on the current design.

SYNTAX

string **update_timing** [-full]

ARGUMENTS

-full

Indicates that the entire timing analysis is to be performed from the beginning. The default is to perform an incremental analysis, which updates only out-of-date information and runs more quickly.

DESCRIPTION

Updates timing for the current design. Timing is also automatically updated by commands that need the information, such as **report_timing**. This command explicitly prepares the design for further analysis.

By default, **update_timing** uses an efficient timing analysis algorithm that requires minimal computation effort and updates existing timing analysis information only where needed. You can override this default behavior using the **-full** option, which causes the entire timing update to be performed from the beginning. To avoid unnecessarily long run times, use the **-full** option only when you need to override incremental timing analysis. If the design is not timed, **update_timing** has the same run time independent of the **-full** option.

The variable **timing_update_default_mode** changes the default behavior of the **update_timing** command. This variable is set to *incremental* by default. If set to *full*, the default behavior of **update_timing** is the same as **update_timing -full**.

If the variable **si_enable_analysis** is set to true, and a valid PrimeTime SI license is in place, **update_timing** also performs crosstalk-aware timing analysis. For more information, see the *PrimeTime SI User Guide*.

EXAMPLES

The following example updates timing information for the current design.

```
pt_shell> update_timing
```

```
pt_shell> update_timing -full
```

SEE ALSO

report_timing (2); **si_enable_analysis** (3), **timing_update_default_mode** (3).

upf_version

Specify the version for the UPF file/syntax.

SYNTAX

```
string upf_version  
[version_id]
```

Data Types

```
version_id      string
```

ARGUMENTS

```
version_id  
    Specifies the UPF version for which the file or following UPF commands are  
    intended.
```

DESCRIPTION

As the UPF standard matures, new updated versions of the UPF standard can occur. The command can be used to specify the version to restrict syntax checking and semantics of UPF commands.

If called with an argument, returns the argument. If called with no argument, returns the current version number.

Currently only version 1.0 is supported. A warning is issued if you specify any other version.

EXAMPLES

```
prompt> upf_version  
1.0  
prompt> help UPF  
connect_supply_net    # Connect supply net to supply ports and/or pins  
create_power_domain   # define power supply distribution network  
create_power_switch   # Define a switch in the power domain  
create_supply_net      # Create power or ground supply net object  
create_supply_port     # Create a port on power domain  
get_power_domains     # Create a collection of power domains  
....
```

The first example queries the current UPF version. The second example shows all UPF commands (i.e., commands that may be affected by a specific UPF standard).

SEE ALSO

```
create_power_domain(2)  
report_power_domain(2)
```

```
create_supply_net(2)
create_supply_port(2)
```

variation_correlation

Computes the correlation between two variations. These variations must have been obtained through attributes (for example, `get_attribute $stat_path variation_slack`) or through a statistical operation (for example, `add_variation`, `sub_variation`, `max_variation`, or `min_variation`) performed on variations obtained from a `get_attribute`. The function returns a float value (in string format). The value -2.0 is returned if an error is detected.

SYNTAX

```
string variation_correlation  
variation1  
variation2
```

Data Types

```
variation1    collection  
variation2    collection
```

ARGUMENTS

```
variation1  
    First variation object.
```

```
variation2  
    Second variation object.
```

DESCRIPTION

This command computes the correlation between two variations. A correlation of 1.0 or close to 1.0 is interpreted as a very strong correlation between the two variations. A correlation of 0.0 can be interpreted as follows: one variation or both has no variation, or the two variations are statistically independent.

A variation is a Tcl collection. Naturally, it is of size one.

EXAMPLES

The following example computes the correlation between two variations.

```
pt_shell> set points [get_attribute $stat_path points]  
_sel90  
  
pt_shell> set ptb [index_collection $points 3]  
_sel91  
  
pt_shell> set pte [index_collection $points 5]  
_sel92  
  
pt_shell> set arrb [get_attribute $ptb variation_arrival]
```

```
{ " " }
```

```
pt_shell> set arre [get_attribute $pte variation_arrival]
{ " " }
```

```
pt_shell> set del [sub_variation [add_to_collection $arre $arrb]]
{ " " }
```

```
pt_shell> set corr [variation_correlation $arrb $del]
0.12854
```

SEE ALSO

```
get_attribute(2)
add_variation(2)
sub_variation(2)
max_variation(2)
min_variation(2)
```


write_activity_waveforms

Create activity waveforms from VCD

SYNTAX

```
int write_activity_waveforms
[-output file_name]
-interval sampling_interval
[-hierarchical_levels level]
[-coverage]
[-exclude_signals list_of_signals]
[-exclude_cells list_of_modules]
-vcd file_name
[-time time_list]
[-peak_window window_size]
[-peak_type min | max]
[-strip_path path]
[-format format]
[-verbose]
```

```
string file_name
int level
float sampling_interval (in ns)
list time_list
string path
string format
```

ARGUMENTS

- output *file_name*
Specify the prefix of the file in which the waveform data is to be written.
- interval *sampling_interval*
Specify the sampling interval that is used for activity waveform. during each interval, activity is aggregated.
- hierarchical_levels *level*
Create power waveforms for hierarchical cells only to the specified level from the top. The default is to create waveforms for only the top level of the design.
- coverage
Define activity to be the percentage of nets in a block that toggle at least once during an interval. By default, the definition of activity is the number of toggles in the block for the interval, divided by number of nets and the interval period.
- exclude_signals *list_of_signals*
This option allows the user to exclude some signal names from consideration. The list of signals is a list of signal names. No path should be given with the signal names; all signals with the specified names will be ignored from

the VCD. This option is useful for excluding signals such as clock signals from analysis.

-exclude_cells *list_of_modules*

This option allows the user to exclude some module names from consideration. The list of modules is a list of module names. The names can either not give a path to the module (such as C instead of TOP/A/B/C), or the full path can be included. If no path is given, all modules with the specified names will be ignored from the VCD. The given module names may be globs, such as A*, which would exclude any module starting with 'A'. When a module is excluded, it will not be included in the waveforms or the text reports (see -verbose). Also, the signals in the excluded modules will not be considered in the hierarchy above the excluded module. As an exception to this, when a signal is defined in TOP/A/B and also defined in TOP/A/B/C, the signal will still be counted as being in B if module C is excluded. The -exclude_cells option is useful for excluding macro modules or unsynthesized modules. The given list of modules needs to be a list of module names in the vcd. It should not be a collection of cells from a design.

-vcd *filename*

Specifies the filename of the VCD to be analyzed. This option is required.

-time *time_list*

Specifies the time windows for the analysis. The user may want to only analyze and produce waveforms for a portion of the time in the vcd. The -time option accepts a list of numbers as its argument. The argument must have an even number of elements. Each consecutive pair of numbers is a time window; the first number is the start of the window, the second is the end of the window. Activity analysis is only performed on the given time windows. The given time values are in nanoseconds. The first number in the list can use the special value -1 to represent the beginning of the VCD. The last number in the list can use the special value -1 to represent the end of the VCD.

-peak_window *window_size*

Specify the window size to be used when searching for the peak activity period. By default, the peak_window is the same as the interval. PTPX searches for the peak activity period for each hierarchical block during activity file analysis. The width of the period is the peak_window. For example, the interval might be set to 10ns, while the peak_window could be 500ns. In this case, PTPX looks for activity in a 500ns window when deciding when the peak activity occurs. Due to implementation limitations, the 500ns window in the example will start and end on interval boundaries. Therefore, the value specified for peak_window must be a multiple of the interval value.

-peak_type {min | max}

Specifies whether the command is to search for an interval with maximum or minimum activity (or coverage, if -coverage is also specified). By default, the command searches for an interval with maximum activity (or coverage).

-strip_path *path*

Specifies the path to the module in the VCD of interest. All modules outside the hierarchy under the specified path are excluded. The names in the waveform file are shortened.

-format *format*
 Specifies the format for the output file. Valid formats are "fsdb" and "out".
 The default format is "fsdb"

-verbose
 After analysis, a report is generated, similar to the command
report_activity_waveforms.

DESCRIPTION

Plots the average toggle rate for modules in the VCD over time. The toggle rate during an interval for a module is the average toggle rate for signals in the module for the interval.

No design or library needs to be loaded in order to use this command, only a VCD is needed.

Activity waveforms can be useful for qualifying the activity vectors. The user can review the waveforms to determine if the testbench simulated as expected, and whether the vectors have covered enough of the design to be useful as inputs to power analysis.

The total VCD simulation time is partitioned into intervals, for which the average activity is calculated based on the formula below:

$$\text{Average toggle rate} = \frac{\text{\# of toggles on all the signals for the interval}}{\text{\# of signals} \times \text{(length of interval in nanoseconds)}}$$

Users can also select to view the activity coverage over time to get a sense of how many of the signals are being exercised. A signal is "covered" if it has at least one toggle within the interval, so the coverage per interval is calculated as follows

$$\text{Coverage} = \frac{\text{\# of signals with at least one toggle}}{\text{\# of signals}} \times 100\%$$

The output format for the waveforms is either .fsdb (by default) or the .out format. Either format can be viewed with a waveform viewer such as nWave.

EXAMPLES

This example outputs the text file foo.out with waveform information in the .out format.

```
pt_shell> write_activity_waveforms -output foo -vcd my_vcd.vcd -interval 10
```

This example only analyzes during the time window from 100 to 300 and then from 1000 to the end of the vcd:

```
pt_shell> write_activity_waveforms -output foo \  
-vcd my_vcd.vcd -interval 10 -time {100 300 1000 -1}
```

Use **-exclude_cells** to exclude some modules from the waveforms and to exclude nets in the modules from being included in the totals for other modules higher in the hierarchy:

```
pt_shell> write_activity_waveforms -output foo -vcd my_vcd.vcd \  
        -interval 10 -exclude_cells {assertion_module*  other_module}
```

SEE ALSO

```
read_vcd (2),  
report_activity_waveforms(2)
```

write_arrival_annotations

Writes arrival and slew annotations for ILMs or contexts of the given list of instances or for all top level instances as a script of commands.

SYNTAX

```
int write_arrival_annotations  
[-instances instance_list]  
[-context]  
[-design]  
  
list instance_list
```

ARGUMENTS

-instances *instance_list*
Indicates that PrimeTime is to write arrival annotations for the set of pins affecting the given list of instances. Primetime looks for appropriate files that contain the list of aggressor driver pins inside the directories meant for these instances, and creates corresponding scripts.

-context
Indicates that the annotations are meant for the context of the block.

-design
Indicates that the annotations are meant for the entire design.

DESCRIPTION

Writes arrival annotations to be used at block level or chip level hierarchical SI analysis. This command is affected by **pt_ilm_dir** environment variable. Primetime looks for directories meant for the list of instances (or all top level instances) at the location given by variable **pt_ilm_dir**. Inside the directory, it looks for wrapper.txt file or ilm.txt file depending on whether **-context** option is given or not. Then it creates a new file called wrapper.pt.gz or ilm.pt.gz that contains the script needed to annotate the arrivals and slews at block or chip level.

If the annotations are for the chip level and the ILM is being generated for the full design (happens if the blocks are shielded from each other and the SI context of the instance is not generated), then use the **-design** option. Then, Primetime looks for a directory with the name of the block design and looks for ilm.txt file inside the directory.

EXAMPLES

The following example writes the annotations for contexts of all top level blocks.

```
pt_shell> write_arrival_annotations -context
```

The following example write annotations for the ILM of instance I1.

```
pt_shell> write_arrival_annotations -instance {I1}
```

SEE ALSO

```
pt_ilm_dir(3), create_si_context(3), create_ilm(3).
```

write_astro_changes

Outputs netlist changes and coupling separations from this session in native Astro formats.

SYNTAX

```
int write_astro_changes -format type [-dont_merge_changes] file_name
string type
string file_name
```

ARGUMENTS

-format *type*

Output format. Available formats are *heco* (used for structural netlist changes such as buffer insertions), and *scheme* (which is used for coupling separation directives).

-dont_merge_changes

By default, **write_astro_changes** will attempt to merge together redundant structural operations to reduce and simplify the HECO file. If this option is specified, no simplification will be performed.

file_name

Specifies the name of a file to which the the output is to be written.

DESCRIPTION

The **write_astro_changes** command outputs design change information in native Astro formats. The type of information depends on the specified output format.

If the 'heco' format is specified, structural changes are written in Astro's hierarchical ECO (HECO) format. The following PrimeTime operations are considered to be structural changes:

size_cell, insert_buffer, remove_buffer, connect_net, disconnect_net, create_cell, create_net, remove_cell, remove_net.

By default, **write_astro_changes** will attempt to merge together redundant structural operations to reduce and simplify the HECO file. For example, if a buffer is inserted then removed, this will simplify to a null operation. If a cell is resized multiple times, this is simplified to a single resize operation. If the **-dont_merge_changes** option is specified, no simplification will be performed.

If the 'scheme' format is specified, the coupling separations in the current PrimeTime SI analysis will be written as scheme commands for Astro. These coupling separations are applied in a PrimeTime SI analysis using the **set_coupling_separation** and **remove_coupling_separation** commands.

The separations written depend on the capacitive couplings present in the current analysis. For a given victim net, only effective aggressors to that victim net will be processed. If SI analysis is not enabled (i.e. if si_enable_analysis is not set to true), no separations will be written out.

SEE ALSO

`connect_net (2)`, `create_cell (2)`, `create_net (2)`, `disconnect_net (2)`, `insert_buffer (2)`, `remove_buffer (2)`, `remove_cell (2)`, `remove_net (2)`, `size_cell (2)`, `remove_coupling_separation (2)`, `set_coupling_separation (2)`.

write_binary_aocvm

Creates a binary advanced on-chip variation (OCV) file from an advanced OCV file.

SYNTAX

```
int write_binary_aocvm  
[-compress]  
aocvm_file  
binary_file
```

Data Types

<i>aocvm_file</i>	string
<i>binary_file</i>	string

ARGUMENTS

-compress
Compresses the output binary advanced OCV file to reduce file size.

aocvm_file
Specifies the name of the input advanced OCV file.

binary_file
Specifies the name of the output binary advanced OCV file.

DESCRIPTION

The **write_binary_aocvm** command creates binary encoded advanced OCV files from ASCII advanced OCV files. This command is used to protect sensitive process related information.

The **read_aocvm** command can read binary and compressed binary advanced OCV files created using the **write_binary_aocvm** command. No additional arguments are required to read binary or compressed binary advanced OCV files.

An advanced OCV derate table imported from a binary or compressed binary file is not visible in the **report_aocvm** output.

EXAMPLES

The following example shows how to create a binary advanced OCV file named "binary_file" from an ASCII advanced OCV file named "aocvm_file".

```
pt_shell> write_binary_aocvm aocvm_file binary_file
```

The following example shows how to create a compressed binary advanced OCV file

named "small_binary_file" from an ASCII advanced OCV file named "aocvm_file".

```
pt_shell> write_binary_aocvm -compress aocvm_file small_binary_file
```

SEE "ALSO"

```
read_aocvm(2)  
remove_aocvm(2)  
report_aocvm(2)
```

write_changes

Outputs netlist changes that have been recorded during this session.

SYNTAX

```
int write_changes [-format chg_format] [-output file_name] [-reset]
stringchg_format
stringfile_name
```

ARGUMENTS

-format *chg_format*

Output format. Available formats are *ptsh* (a PrimeTime script), *dctcl* (a Tcl script for Design Compiler), *icctcl* (a Tcl script for IC Compiler), and *text*. The default is *ptsh*.

-output *file_name*

Specifies the name of a file to which the the output is to be written. The default is the standard output.

-reset

Specifies that netlist change lists should be cleared so that any history of previous netlist editing commands is no longer maintained.

DESCRIPTION

The **write_changes** command outputs the netlist changes which have been made to the current design since it was linked. It is typically formatted as a script. By default, output goes to standard output. Use the **-output** option to write the output to a file.

The **-reset** option can be used to clear off the netlist change lists maintained by PrimeTime and PrimeTime SI. It does not undo the effect of the previous netlist editing commands. This option simply clears off the netlist change lists.

The following commands create entries in the change list:

size_cell, *insert_buffer*, *remove_buffer*, *connect_net*, *disconnect_net*, *create_cell*, *create_net*, *remove_cell*, *remove_net*, *rename_cell*, *rename_net*, *swap_cell*, *set_coupling_separation*, *remove_coupling_separation*.

As a script, the output of **write_changes** is a series of scoping commands (**current_instance**) and netlist editing commands. The text format is basically one line per atomic editing operation, in order, describing in words the operation, the instance in which it occurred, and the objects involved.

The change list is not a verbatim log of what you entered. The following are some cases when the change list may be somewhat different from the command history:

- When the objects were specified at a higher level of the hierarchy (for example, creating a cell within *u1/u2*), the change list would scope to *u1/u2* using **current_instance**, then create the cell.

- Some object arguments are made explicit by wrapping them within a command which creates a collection, such as **get_pins**.
- Often, multiple arguments to commands like **create_cell** are converted into separate commands in the change list.
- Sometimes, some arguments are dropped from the change list, such as when superfluous arguments are given to **remove_cell**.

write_changes will attempt to merge together some redundant operations to reduce and simplify the change file. For example, if a buffer is inserted then removed, this will simplify to a null operation. If a cell is resized multiple times, this is simplified to a single resize operation. Please note that no simplification is made for the text format.

The *dctcl* format can be used with Design Compiler version Z-2007.03.

Note that the following set of commands are not written to the Design Compiler Tcl scripts.

rename_cell, rename_net, swap_cell, set_coupling_separation,
remove_coupling_separation.

The *icctcl* format can be used with IC Compiler version Z-2008.12.

Note that the following set of commands are not written to the IC Compiler Tcl scripts.

rename_cell, rename_net, swap_cell, set_coupling_separation,
remove_coupling_separation, create_net if the net name contains wildcard character or hierarchy separator.

IC Compiler does not support: create_cell -exact. If the -exact is used, the command create_cell is written out without the -exact flag when the icctcl format is selected.

EXAMPLES

The following shows a set of netlist editing commands and the change list created by the edits. Notice how the creation of two cells in a single command is converted to two entries in the change list.

```
pt_shell> create_cell i1/u1 class/AN2
Uniquifying 'i1' (inter) as 'inter_0'.
Information: Created cell 'u1' in 'middle/i1'. (NED-014)
1
pt_shell> size_cell i1/u1 class/AN2P
Information: Sized 'i1/u1' with 'class/AN2P'. (NED-045)
1
pt_shell> create_net i1/net1
Information: Created net 'net1' in 'middle/i1'. (NED-016)
1
```

```

pt_shell> connect_net i1/net1 i1/u1/A
Information: Connected 'i1/u1/A' to 'net1'. (NED-018)
1
pt_shell> create_cell {u1 u2} class/AN2
Information: Created cell 'u1' in design 'middle'. (NED-014)
Information: Created cell 'u2' in design 'middle'. (NED-014)
1
pt_shell> write_changes -format ptsh
#####
# Change list, formatted for PrimeTime
#####
current_instance
current_instance i1
create_cell {u1} {class/AN2}
size_cell {u1} {class/AN2P}
create_net {net1}
connect_net {net1} [get_pins {u1/A}]
current_instance
create_cell {u1} {class/AN2}
create_cell {u2} {class/AN2}
1
pt_shell> write_changes -format text
Change list, formatted as text:
  1. create_cell in i1: new cell 'u1' lib_cell 'class/AN2'
  2. size_cell in i1: 'u1' sized to 'class/AN2P'
  3. create_net in i1: new net 'net1'
  4. connect_net in i1: pin 'u1/A' to net 'net1'
  5. create_cell in <top>: new cell 'u1' lib_cell 'class/AN2'
  6. create_cell in <top>: new cell 'u2' lib_cell 'class/AN2'
1

```

SEE ALSO

```

connect_net (2), create_cell (2), create_net (2), current_instance (2),
disconnect_net (2), insert_buffer (2), remove_buffer (2), remove_cell (2),
remove_net (2), rename_cell (2), rename_net (2), size_cell (2), swap_cell (2).
set_coupling_separation (2). remove_coupling_separation (2).

```

write_context

Writes the timing context information, for specified instances, as a pt_shell, dc_shell, or dc_shell-t script.

SYNTAX

```
string write_context [-timing] [-environment]
[-design_rules]
[-constant_inputs]
[-derive_file_name]
[-prefix file_prefix]
[-extension file_extension]
[-script_directory directory]
[-separator name_separator]
[-output file_name]
[-format script_format]
[-nosplit]
cell_list
```

```
string file_prefix
string file_extension
string directory
string name_separator
string file_name
string script_format
list cell_list
```

ARGUMENTS

-timing

Writes only timing information; for example, clocks, input and output delays, and timing exceptions. By default, all context information is written.

-environment

Writes only environment-related information; for example, operating conditions (process, temperature, and voltage), wire load model, capacitive loads on input and output pins, and driving cell information on input pins. By default, all context information is written.

-design_rules

Writes only design rules; for example, max_capacitance, max_transition, and max_fanout. By default, all context information is written.

-constant_inputs

Writes logic constants on input pins of the characterized instance. By default, all context information is written.

-derive_file_name

Derives the name of the script file from the instance name. This option and the **-output** option are mutually exclusive; if neither is specified, the script is written to standard output. If you use the **-prefix**, **-extension**, **-script_directory**, or **-separator** options, you must also use **-derive_file_name**.

-prefix *file_prefix*

If **-derive_file_name** is used, specifies the prefix for the derived constraint file name; by default, no prefix is used. You cannot use the slash (/) or apostrophe (') characters in the prefix. This option and the **-output** option are mutually exclusive. If you use this option, you must also use **-derive_file_name**.

-extension *file_extension*

If **-derive_file_name** is used, specifies the extension for the derived constraint file name; the default extension is *ptsh*. This option and the **-output** option are mutually exclusive. If you use this option, you must also use **-derive_file_name**.

-script_directory *directory*

If **-derive_file_name** is used, specifies the directory in which the constraint file is to be generated; the default is the current directory. This option and the **-output** option are mutually exclusive. If you use this option, you must also use **-derive_file_name**.

-separator *name_separator*

If **-derive_file_name** is used, specifies a single character string to use as a hierarchical separator in constructing the derived constraint file name generated for each instance; the default is the "at" sign (@). You cannot use the slash (/) or backslash (\) characters as name separators. This option and the **-output** option are mutually exclusive. If you use this option, you must also use **-derive_file_name**.

-output *file_name*

Writes the script to file *file_name*. This option and the **-derive_file_name** option are mutually exclusive; if neither is specified, the script is written to standard output. You also cannot use the **-prefix**, **-extension**, **-script_directory**, or **-separator** options with **-output**.

-format *script_format*

Specifies the output format for the script. Allowed values are *ptsh* (the default) for *pt_shell*, *dcsh* for *dc_shell*, and *dctcl* for *dc_shell-t*.

-nosplit

The **-nosplit** option prevents line-splitting. This is most useful for doing diffs on previous scripts, or for post-processing the script.

cell_list

Specifies a list of instances for which the context is to be written out as a script.

DESCRIPTION

Writes the timing context of the specified instances as a *pt_shell*, *dc_shell*, or *dc_shell-t* script. The context must be captured using the **characterize_context** command. The script contains a series of *pt_shell*, *dc_shell*, or *dc_shell-t* commands. When you execute these commands on the subdesign, PrimeTime initializes its timing environment to the information characterized from the context. This command is the mechanism used to export context information to other tools.

If **write_context** is issued without options, the script is written for all context information, in pt_shell format, to standard output.

Because the **characterize_context** command does not capture delays and parasitics annotated on internal nets of the characterized instance, the generated script does not include this information. To export this information to module level tools, use the **write_physical_annotations** command. To augment the script generated by the **write_context** command so that it imports the physical annotation files generated by the **write_physical_annotations** command, use the **-append** option of the **write_physical_annotations** command.

EXAMPLES

The following example writes to standard output the timing-related context information for instances I1 and I2, in dc_shell format.

```
pt_shell> write_context -timing -format dcsh {I1 I2}
```

The following command writes the environment and constant input context information for I2 to a pt_shell script file named des1.

```
pt_shell> write_context -env -const -out des1 I2
```

The following command writes all context information in dc_shell format. The two files FRST_I1%X.scr and FRST_I2.scr are generated in the directory ~/project/scripts.

```
pt_shell> write_context -derive_file_name -prefix FRST_  
-separator % -extension scr \  
-script_directory ~/projects/scripts \  
-format dcsh {I1/X I2}
```

The following command writes all context information in pt_shell format to a file named I1.ptsh in the current directory.

```
pt_shell> write_context -derive_file_name I1
```

SEE ALSO

characterize_context (2), **remove_context** (2), **report_context** (2),
write_physical_annotations (2).

write_ilm_netlist

Writes out a flattened Verilog netlist corresponding to the interface logic for the current design.

SYNTAX

```
int write_ilm_netlist
    [-include_all_net_pins]
    [-verbose]
    file_name
```

string *file_name*

ARGUMENTS

-include_all_net_pins

Indicates that all pins on non-clock interface logic nets and propagated clock interface logic nets are to be included in the netlist. By default, only pins that have the **is_interface_logic_pin** attribute are written out in the Verilog netlist. Use this option if the interface logic model (ILM) will be used in non-SDF flows and delay calculation will be performed using the information contained in the ILM. Preserving all pins on a net maintains correct pin capacitance information for the net. This option does not affect non-propagated clock nets; that is, nets in the clock network that have user-specified source latency.

-verbose

Indicates that information about the number of cells and nets in the original design and in the ILM netlist is to be written to standard output.

DESCRIPTION

Writes out a flattened Verilog netlist corresponding to the interface logic on the current design. This represents the netlist of the interface logic model (ILM).

Use the **-include_all_net_pins** option if the ILM will be used in non-SDF flows. You can use the **-verbose** option for approximating the reduction of the interface logic model (ILM) size when compared with the original netlist.

For a discussion of ILM creation and the associated commands, see the manual page for the **identify_interface_logic** command.

EXAMPLES

The following example writes out a Verilog netlist that includes statistics comparing the ILM with the original netlist.

```
pt_shell> write_ilm_netlist -verbose model.v
```

The following example writes out a Verilog netlist while keeping all pins on

interface logic nets in the netlist.

```
pt_shell> write_ilm_netlist -include_all_net_pins model.v
```

SEE ALSO

`get_ilm_objects` (2), `identify_interface_logic` (2), `write_ilm_parasitics` (2),
`write_ilm_script` (2), `write_ilm_sdf` (2).

write_ilm_parasitics

Writes out in SPEF format all annotated parasitics on the interface logic for the current design.

SYNTAX

```
int write_ilm_parasitics
    [-input_port_nets]
    [-constant_nets]
    [-compress compression]
    [-format format]
    file_name
```

string *file_name*

ARGUMENTS

-input_port_nets
Indicates that parasitic information is to be written out for nets connected to the input ports on the current design. By default, this information is not written out.

-constant_nets
Indicates that parasitic information is to be written out for constant nets also. By default, this information is not written out.

-compress *compression*
Specifies that the file should be compressed. The only valid value for *compression* is gzip.

-format *format*
Specifies the format of the file. The valid values for *format* are SPEF and SBPF. The default is SPEF.

file_name
Specifies the name of the output SPEF file.

DESCRIPTION

Writes out in SPEF format all annotated parasitics on the current design that are defined on its interface logic. This information is used as back-annotation information for the interface logic model (ILM).

For a discussion of ILM creation and the associated commands, see the manual page for the **identify_interface_logic** command.

EXAMPLES

The following example writes to the SPEF file model.spef the parasitics that apply to interface logic.

```
pt_shell> write_ilm_parasitics model.spef
```

The following example includes parasitic information for the input ports and writes all parasitic information to the SPEF file model.spef.

```
pt_shell> write_ilm_parasitics -input_port_nets  
model.spef
```

SEE ALSO

```
get_ilm_objects (2), identify_interface_logic (2), write_ilm_netlist(2),  
write_ilm_script (2), write_ilm_sdf (2).
```

write_ilm_script

Writes constraints, assertions and exceptions for interface logic as a script for PrimeTime or Design Compiler.

SYNTAX

```
int write_ilm_script
    [-instance]
    [-format script_format]
    [-compress compression]
    [-nosplit]
    file_name
```

```
string script_format
string file_name
string compression
```

ARGUMENTS

-instance

Indicates that the script written out should be appropriate for use when the interface logic is instantiated at the chip level. That is, assertions and exceptions must refer to boundary pins on a block instead of referring to ports on a design. The script does not include environment information because that is defined by the chip-level design. Do not use this option to generate a script for verifying the standalone ILM against the original design.

-format *script_format*

Specifies the output format for the script. Allowed values are *ptsh* (the default) for *pt_shell*, *dcsh* for *dc_shell*, and *dctl* for *dc_shell-t*.

-compress *compression*

Specify that the script is to be compressed; by default, the script is written as standard text. The only valid value for *compression* is *gzip*.

-nosplit

The **-nosplit** option prevents line-splitting. This is most useful for doing diffs on previous scripts, or for post-processing the script.

file_name

Specifies the name of the script file to write. If the **-compress** option is also specified, the extension ".gz" is added if it is not already present.

DESCRIPTION

Writes constraints, assertions and exceptions for interface logic as a script for *pt_shell*, *dc_shell*, or *dc_shell-t*.

For a discussion of ILM creation and the associated commands, see the manual page for the **identify_interface_logic** command.

EXAMPLES

The following example writes out in ptsh format the assertions and exceptions that apply to interface logic, to the file model.pt. This script would be used to verify an ILM against the original netlist.

```
pt_shell> write_ilm_script model.pt
```

The following example writes out the assertions and exceptions that can be applied to the ILM when it is instantiated at the chip level.

```
pt_shell> write_ilm_script -instance model.pt
```

SEE ALSO

```
get_ilm_objects(2), identify_interface_logic (2), write_ilm_netlist(2),  
write_ilm_parasitics(2), write_script(2), ilm_ignore_percentage(3).
```

write_ilm_sdf

Writes out a Version 2.1-compliant Standard Delay Format (SDF) back-annotation file for the interface logic of the current design.

SYNTAX

```
int write_ilm_sdf
    [-input_port_nets]
    [-output_port_nets]
    [-significant_digits digits]
    [-annotated]
    [-no_edge]
    [-compress compression]
    [-version sdf_version]
    file_name
```

```
int digits
string file_name
string compression
string sdf_version
```

ARGUMENTS

- significant_digits *digits***
Specifies the number of digits to the right of the decimal point that are to be written in SDF delay triplets. Allowed values are 0-13; the default is 3.
- input_port_nets**
Indicates that the SDF file is to include delays of nets connected to input ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available.
- output_port_nets**
Indicates that the SDF file is to include delays of nets connected to output ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available.
- annotated**
Indicates that the SDF is to include only timing arcs that have been annotated with the **read_sdf**, **set_annotated_delay**, or **set_annotated_check** commands.
- no_edge**
Indicates that the generated SDF is not to include any edges (posedge or negedge) for combinational IOPATHs.
- compress *compression***
Specify that the file should be compressed. The only valid value for *compression* is gzip.

`-version sdf_version`
Selects which SDF version to use. Supported SDF versions are 1.0, 2.1, and 3.0. SDF 2.1 is the default.

`file_name`
Specifies the name of the SDF file to write. If the **-compress** option is also specified, the extension ".gz" is added if it is not already present.

DESCRIPTION

Writes out in SDF format information on cell and net delays for the interface logic on the current design.

For a discussion of ILM creation and the associated commands, see the manual page for the **identify_interface_logic** command.

EXAMPLES

The following example writes out SDF information that applies to the interface logic; only SDF data that was previously back-annotated on a design is written out.

```
ptshell> write_ilm_sdf -annotated model.sdf
```

The following example writes out an SDF file, model.sdf, for the interface logic. All delay values are included whether or not they had been back-annotated. Information is included for the input and output ports on the design. Four significant digits to the right of the decimal point are written in SDF delay triplets.

```
ptshell> write_ilm_sdf -input_ports -output_ports -significant_digits 4  
model.sdf
```

SEE ALSO

get_ilm_objects (2), **identify_interface_logic** (2), **write_ilm_netlist**(2),
write_ilm_parasitics (2), **write_ilm_script** (2); **write_sdf** (2);

write_interface_timing

Generates an interface timing ASCII report for a gate-level netlist, an interface logic model (ILM), or an extracted timing model (ETM) design.

SYNTAX

```
int write_interface_timing  
file_name  
[-ignore_ports port_list]  
[-significant_digits digits]  
[-nosplit]  
[-timing_type ttype]  
[-verbose]  
[-include incl_list]  
[-latch_level level]
```

```
string file_name  
list port_list  
int digits  
string ttype  
list incl_list  
int level
```

ARGUMENTS

file_name

Specifies the name of the file to which the interface timing information is to be written.

-ignore_ports *port_list*

Specifies a list of ports that are to be excluded from the timing file. By default, all ports are included.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be reported following the method used in the **report_timing** command. Allowed values are 0-13. The default value is the maximum of six and the value of the **report_default_significant_digits** variable. Use this option if you want to override the default. Fixed-precision floating-point arithmetic is used for delay calculation; thus the actual precision might depend on the platform.

-nosplit

The **-nosplit** option prevents line-splitting. This is most useful for doing diffs on previous scripts or for postprocessing the script.

-timing_type *ttype*

Selects the type of timing data section to be reported. The field *ttype* accepts one of two entries: **slack** (default) or **arc**. When **slack** is selected, the slack associated with interface timing relationships is reported. When **arc** is selected, the timing arc values for interface timing relationships is reported. Slack values are a function of the external environment (clocks, arrival times), but arc values are not. If you are validating an ETM that

gave a MEXT-54 warning when generated, do not use arc values. When the MEXT-54 warning is given, some of the ETM arcs have been adjusted by a multiple of the clock period and will not match the arc values of the original netlist.

-verbose

The **-verbose** option shows details of the progress of this command and provides data on CPU time and memory usage. For longer runs, it shows a time estimate of completion.

-include *incl_list*

Selects the type of data to analyze. The field *incl_list* accepts two entries: **timing** (default) and **noise**. When **timing** is selected, the slack/arc, capacitance, transition time, and design rules sections are reported. When **noise** is selected, the noise sections *noise_detection* and *noise_calculation* are reported. If both **timing** and **noise** are selected, then both timing and noise data are reported.

-latch_level *level*

Specifies the number of levels of latch borrowing that are to occur at the interface of a design. The default value is 12.

DESCRIPTION

Generates an interface timing ASCII report for a gate-level netlist, an interface logic model (ILM), or an extracted timing model (ETM) design. The report is a file containing interface timing, slew, capacitance, design rule, and noise information for a gate-level netlist, an ILM, or an ETM design. The command performs an implicit **update_timing** and **update_noise** if necessary for the selected data sections. If the **timing_slew_propagation_mode** variable is set to *worst_arrival*, the **write_interface_timing** command ensures that interface timing information for an input port is not influenced by the transition time on other ports. A return code of **1** indicates that the command ran to completion; a **0** indicates an error.

You normally use the **compare_interface_timing** command to compare two report files that have been generated by the **write_interface_timing** command.

The output report file contains the following sections:

1a) **Slack Section** – Contains worst-case slacks for combinational paths from an input port to an output port, for input-to-register paths from an input port to each clock used to clock the input port, and for output-to-register paths from each clock used to clock an output port to the output port. For each port-to-port, register-to-port, and port-to-register timing relationship, it reports slacks for the following possible arc types: *min_seq_delay*, *max_seq_delay*, *min_combo_delay*, *max_combo_delay*, *setup*, *hold*, *recovery*, *removal*, *clock_gating_setup*, and *clock_gating_hold*.

1b) **Arc Section** – Contains worst-case setup/hold/delay arc values for combinational paths from an input port to an output port, for input-to-register paths from an input port to each clock used to clock the input port, and for output-to-register paths from each clock used to clock an output port to the output port. For each port-to-port, register-to-port, and port-to-register timing relationship, it reports arc values for the following possible arc types: *min_seq_delay*, *max_seq_delay*, *min_combo_delay*, *max_combo_delay*, *setup*, *hold*, *recovery*, *removal*, *clock_gating_setup*, and *clock_gating_hold*.

2) **Transition Time Section** – Contains actual transition times on all ports for the four delay types: min_fall, min_rise, max_fall, and max_rise.

3) **Input Capacitance Section** – Contains actual total (lumped) or effective capacitances on boundary nets connected to each port.

4) **Design Rule Section** – Contains design rule information for all ports: max_capacitance, min_capacitance, max_transition, max_fanout, and fanout_load.

4) **Noise Detection Section** – Contains noise slack information for all input ports for the four noise regions: below low, above low, below high, above high.

4) **Noise Calculation Section** – Contains driver steady-state resistance for all output ports for the four noise regions: below low, above low, below high, above high.

EXAMPLES

The following example writes timing information for the current design to the *model.rpt* file.

```
pt_shell> write_interface_timing -timing_type arc model.rpt
```

The following example writes noise information for the current design to the *noise_net.rpt* file.

```
pt_shell> write_interface_timing -include noise noise_net.rpt
```

SEE ALSO

```
compare_interface_timing (2), update_timing (2); update_noise (2);  
report_default_significant_digits (3).
```

write_parasitics

Writes out annotated parasitics information for the current design.

SYNTAX

Boolean **write_parasitics**

-format *file_fmt*

-nets *net_list*

-no_name_map

file_name

string *file_fmt*

string *file_name*

ARGUMENTS

-format *file_fmt*

Specifies the format of the output parasitics file. Currently, the only allowed values are SPEF (Standard Parasitic Exchange Format) and SBPF (Synopsys Binary Parasitics Format).

-nets *net_list*

Specifies the list of nets to output parasitics in the parasitics file. This option is supported for both SPEF (Standard Parasitic Exchange Format) and SBPF (Synopsys Binary Parasitics Format).

-no_name_map

This option is only for SPEF (Standard Parasitic Exchange Format). When specified for SPEF, it means that name maps should not be used for generating SPEF file. By default, a name map section is always generated.

file_name

Specifies the name of the output parasitics file.

DESCRIPTION

The **write_parasitics** command writes all parasitics annotated on the current design to the file specified in the *file_name* argument. **write_parasitics** supports the Standard Parasitic Exchange Format (SPEF). It also supports Synopsys Binary Parasitics format (SBPF) and requires a *PrimeTime SI* license for SBPF.

Binary parasitics are very useful for analysis iterations. You can read parasitics in one format (for example, SPEF), then use **write_parasitics** to output the parasitics in SBPF format, which loads much faster than ASCII formats and is much more compact. The **write_parasitics** command can write detailed RC, PI model, and lumped-capacitance networks.

SPEF writing can be useful for debugging if you read parasitics into PT via SBPF.

EXAMPLES

The following example reads the parasitics file top.spef, then writes it out in SBPF format.

```
pt_shell> read_parasitics top.spef
*****
Report : read_parasitics /u/designs/top.spef
Design : TOP
*****
      0 error(s)
      Format is SPEF
      Number of annotated nets           :          66
      Number of annotated capacitances   :        1390
      Number of annotated resistances    :        1333
      Number of annotated RC pi models   :           0
      Number of annotated Elmore delays  :           0
1
pt_shell> write_parasitics -format SBPF top.sbpf
Writing binary parasitics for 66 networks (66 RC, 0 PI, 0 LC)...
1
pt_shell> write_parasitics -format SPEF top.spef
1
```

SEE ALSO

read_parasitics (2).

write_physical_annotations

Writes annotated delays and parasitics for a hierarchical cell.

SYNTAX

```
string write_physical_annotations [-sdf sdf_file]  
[-version sdf_version]  
[-nets_only]  
[-cells_only]  
[-boundary_nets]  
[-parasitics parasitics_file]  
[-append script_file]  
[-format script_format]  
cell
```

```
stringssdf_file  
stringssdf_version  
stringparasitics_file  
stringsscript_file  
stringsscript_format  
string cell
```

ARGUMENTS

-sdf *sdf_file*

Indicates that annotated delays are to be written in Standard Delay Format (SDF) to the specified *sdf_file*. You must specify either **-sdf** or **-parasitics**, but you cannot specify both. If you specify **-version**, **-nets_only**, or **-cells_only**, you must also specify **-sdf**.

-version *sdf_version*

Specifies the version of the SDF to use. Allowed values are *1.0* and *2.1* (the default). If you specify **-version**, you must also specify **-sdf**.

-nets_only

Indicates that only delays annotated on nets be written out as SDF. By default, all annotated delays are written out. If you specify **-nets_only**, you must also specify **-sdf**.

-cells_only

Indicates that only delays and timing checks annotated on cells be written out as SDF. By default, all annotated delays are written out. If you specify **-cells_only**, you must also specify **-sdf**.

-boundary_nets

Indicates that parasitics annotated on boundary nets of the specified cell must also be written out to the parasitics file. By default, only the annotated parasitics of internal nets are written out. Do not use this option if you issue this command in conjunction with the **characterize_context** command to avoid overwriting characterized capacitance on boundary nets. If you specify **-boundary_nets**, you must also specify **-parasitics**.

-parasitics *parasitics_file*

Indicates that annotated capacitance and resistance on nets are to be written to the specified *parasitics_file* as a script. Annotated capacitance is written out as a **set_capacitance** command for pt_shell and a **set_load** command for dc_shell. Annotated resistance is written out as a **set_resistance** command. You must specify either **-sdf** or **-parasitics**, but you cannot specify both. If you specify **-boundary_nets**, you must also specify **-parasitics**.

-append *script_file*

Appends commands to import the generated files in the given script file. The SDF file is read using the **read_sdf** command in pt_shell and the **read_timing** command in dc_shell. The parasitics file is included in the script file.

-format *script_format*

Specifies the format in which to output the commands in the script file and the parasitics file. Allowed values are *ptsh* (the default), *dcsh*, or *dctl*.

cell

Specifies the name of the cell for which to export the annotations. The cell must be hierarchical.

DESCRIPTION

The **write_physical_annotations** command writes physical information for a hierarchical block in the design. The exported information includes annotated net and cell delays using the Standard Delay Format (SDF). Annotated parasitics can also be exported as a series of pt_shell or dc_shell commands.

write_physical_annotations is most commonly used in conjunction with the **characterize_context** and **write_context** commands to export timing and physical information for a block from the chip-level environment to module level tools. A typical sequence of commands is to first characterize the timing environment of a block using the **characterize_context** command and write this information as a dc_shell script using the **write_context** command. The **write_physical_annotations** command is then used to export annotated delays and parasitics for internal nets of the block. To avoid overwriting characterized capacitance on boundary nets, do not specify the **-boundary_nets** option. You can use the **-append** option to append commands to read the SDF and include the parasitics script at the end of the script file generated by the **write_context** command.

EXAMPLES

The following command writes the annotated delays on nets for the hierarchical cell 'I1':

```
pt_shell> write_physical_annotations
-sdf I1.sdf -nets_only I1
```

The following command is used in conjunction with the **characterize_context** and **write_context** commands to export the timing and physical environment for cell 'I2' as a dc_shell script. The command uses the **append** option to augment the script generated by the **write_context** command with commands to import the generated delay and parasitic files.

```
pt_shell> characterize_context I2
pt_shell> write_context -format dcsh
-out I2.dcsh I2
pt_shell> write_physical_annotations
-sdf I2.sdf -parasitics I2.rc
-format dcsh -append I2.dcsh
I2
```

SEE ALSO

`characterize_context` (2), `read_sdf` (2), `remove_context` (2), `report_context` (2),
`set_capacitance` (2), `set_resistance` (2), `write_context` (2), `write_sdf`(2).

write_profile

Write profiling information for PrimeTime scripts.

SYNTAX

Boolean **write_profile**

ARGUMENTS

DESCRIPTION

The **write_profile** command can be used to write out profiling information for PrimeTime scripts in profiling session that you started previously using **start_profile** command. You should use this command in conjunction with **start_profile** and **stop_profile** to write out profiling information for PrimeTime scripts.

Writing the profiling information can be done at any time after **start_profile** command even after **stop_profile**.

This command succeeds all the time except when no profiling information is present.

EXAMPLES

The following gives an example of profiling

```
pt_shell> start_profile
pt_shell> ....    # Usual PT script
pt_shell> stop_profile
pt_shell> write_profile prof.data
```

SEE ALSO

start_profile (2)

stop_profile (2)

write_saif

Writes a backward Switching Activity Interchange Format (SAIF) file.

SYNTAX

```
int write_saif file_name
    [-cells cell_list]
    [-derate_glitch value]
    [-rtl]
    [-propagated]
    [-exclude_sdpd]
    [-no_hierarchy]
```

```
string file_name
list cell_list
float value
```

ARGUMENTS

file_name
Specifies the name of the output SAIF file.

-cells *cell_list*
Specifies an optional list of hierarchical instances for which the SAIF file is generated. If this argument is not specified, the SAIF file is generated for the current instance.

-derate_glitch *value*
Specifies a derating factor value to be used for converting part of glitches into inertial glitches. If this argument is not specified, a default derating factor of 0.5 is used.

-no_hierarchy
This option specifies that the SAIF file contains switching activity information for only the top-level design objects. When this option is not specified, the **write_saif** command generates a SAIF file containing switching activity information for all the objects in the hierarchy.

-rtl
Specifies that the generated SAIF file contains the switching activity for the synthesis invariant objects. These are the design ports, hierarchical cell pins, and outputs of sequential and tri-state cells. Note that the **-rtl_direct** of the **read_saif** command should not be used when reading SAIF files generated by the **write_saif** command.

-propagated
When this option is used, the **write_saif** command propagates the user-annotated switching activity to estimate the switching activity of unannotated objects, and generate a SAIF file containing both the annotated and estimated switching activity. When this option is not specified, the **write_saif** command generates a SAIF file containing only the user-annotated switching activity.

-exclude_sdpd

The **write_saif** command outputs user-annotated SDPD information by default. This option allows the **write_saif** command to disable the SDPD switching activity generation. When **write_saif** is called with the **-propagated** option, nonannotated SDPD information is also propagated and generated in the SAIF file. You can use the **-propagated** and **-exclude_sdpd** flags together to output the propagated simple switching activity and no SDPD information.

DESCRIPTION

The **write_saif** command generates a backward SAIF file containing the user-annotated and propagated simple and SDPD switching activity in the current design.

The top-level instance name in the generated SAIF file is the current instance name. You must specify this name as the instance name when reading the SAIF file with the **read_saif** command.

EXAMPLES

The following example shows how a SAIF file containing the user-annotated switching activity information on all design objects

```
pt_shell> write_saif file
```

The following example shows how a SAIF file containing both user annotated and propagated non-SDPD switching activity information can be generated.

```
pt_shell> write_saif file -propagated -exclude_sdpd
```

The following example shows how a SAIF file on a synthesized design object can be generated by the **write_saif** command and read back using the **read_saif** command.

```
pt_shell> current_design top pt_shell> current_instance U1 pt_shell>  
write_saif esign.saif pt_shell> reset_switching_activity pt_shell>  
read_saif -input design.saif -instance U1
```

SEE ALSO

read_saif (2), **set_switching_activity** (2), **set_rtl_to_gate_name** (2),
get_switching_activity (2), **reset_switching_activity** (2), **report_switching_activity**
(2), **report_power** (2).

write_script

Writes design constraints as a script of commands for PrimeTime or Design Compiler.

SYNTAX

```
int write_script
[-no_annotated_delay]
[-no_annotated_check]
[-format script_format]
[-output file_name]
[-compress compression]
[-include categories list]
[-nosplit]
```

```
string script_format
string file_name
string compression
list categories list
```

ARGUMENTS

- no_annotated_delay
Specifies that **set_annotated_delay** commands are not to be written.
- no_annotated_check
Specifies that **set_annotated_check** commands are not to be written.
- format *script_format*
Specifies the output format for the script. Valid values are **ptsh** (the default) for pt_shell, **dcsh** for dc_shell, and **dctcl** for dc_shell-t.
- output *file_name*
Specifies the name of a file to which output is to be written. The default is the standard output. If the **-compress** option is also specified, the extension ".gz" is added if it is not already present.
- compress *compression*
Specifies that the script is to be compressed. By default, the script is written as standard text. The only valid value for *compression* is **gzip**. If you specify this option, you must also specify the **-output** option.
- include *include*
Write specified command categories only. The only valid value for *include* is **exceptions**.
- nosplit
The **-nosplit** option prevents line-splitting. This is most useful for performing diffs on previous scripts or for post-processing the script.

DESCRIPTION

Writes design constraints as a script of commands for PrimeTime or Design Compiler (in dcsh or dtcl formats). The script is used to recreate design intent.

The **write_script** command writes the following information:

- Clock information: clock creation, generated clock creation, clock latency, clock uncertainty, interclock uncertainty.
- Timing information: disable timing, max time borrow. Disable timing set on library objects is written by default, but can be suppressed by setting the **write_script_include_library_constraints** shell variable to *false*.
- Point-to-point exceptions: false paths, min delay, max delay, multicycle paths, group path, input delay, output delay, annotated delay and annotated checks.
- Net attributes: capacitance, resistance. These are no longer written by default; for more information, see the man page for the **write_script_output_lumped_net_annotation** shell variable.
- Port attributes: fanout, capacitance, resistance.
- Design environment: wire load model, operating condition, drive, driving cell, input transition.
- Design rules: min capacitance, max capacitance, min transition, max transition, min fanout, max fanout.

EXAMPLES

The following example writes thescript in ptsh format to the des.pt file.

```
pt_shell> write_script -output des.pt
```

SEE ALSO

```
characterize_context (2), write_context (2);  
write_script_output_lumped_net_annotation (3).
```

write_sdc

Writes out a script in Synopsys Design Constraints (SDC) format.

SYNTAX

```
int write_sdc  
file_name  
[-version sdc_version]  
[-compress compression]  
[-include categories_list]  
[-nosplit]
```

Data Types

<i>file_name</i>	string
<i>sdc_version</i>	string
<i>compression</i>	string
<i>categories_list</i>	list

ARGUMENTS

file_name
Specifies the name of the file to which the SDC script is to be written.

-version *sdc_version*
Specifies the version of SDC to write. Allowed values are *1.2*, *1.3*, *1.4*, *1.5*, *1.6*, *1.7* and *latest* (the default).

-compress *compression*
Specify that the script should be compressed. The only valid value for *compression* is *gzip*.

-include *categories_list*
Write specified command categories only. The only valid value for *categories_list* is *exceptions*.

-nosplit
The **-nosplit** option prevents line-splitting. This is most useful for doing diff on previous scripts or for postprocessing the script.

DESCRIPTION

The **write_sdc** command writes out a script file in the latest Synopsys Design Constraints (SDC) format. This script contains commands that can be used with PrimeTime or with Design Compiler. SDC is also licensed by external vendors through the Tap-in program. SDC formatted script files are read into PrimeTime or Design Compiler using the **read_sdc** command.

Design Compiler supports **read_sdc** only in its Tcl mode, while the **write_sdc** command is available from either shell interface.

By default, the script is written as simple text. Optionally, the script can be compressed using the `-compress` option. In some cases, a file extension will be appended to `file_name` if it is omitted. For example, with gzip formatted files, if `file_name` does not end with ".gz", then ".gz" will be appended to the file name.

By default, the latest version of SDC is written. Some earlier versions of SDC can be written using the `-version` option.

SDC is a subset of the commands already supported by PrimeTime and Design Compiler. Of the commands supported in the latest SDC version, the following may be written by **write_sdc**. Those added for versions 1.3 and later are noted.

General Purpose Commands:

- list

Object Access Functions:

- current_design
- current_instance
- get_cells
- get_clocks
- get_libs
- get_lib_cells
- get_lib_pins
- get_nets
- get_pins
- get_ports
- set_hierarchy_separator

Basic Timing Assertions:

- create_clock
- create_generated_clock (1.3)
- set_clock_gating_check
- set_clock_latency
- set_clock_transition
- set_clock_uncertainty
- set_false_path
- set_input_delay
- set_max_delay
- set_min_delay
- set_multicycle_path
- set_output_delay
- set_propagated_clock

Secondary Assertions:

- set_disable_timing
- set_max_time_borrow
- set_data_check (1.4)
- set_timing_derate (1.5)

Environment Assertions:

- set_case_analysis

```

set_drive
set_driving_cell
set_fanout_load
set_input_transition
set_load
set_max_area
set_max_capacitance
set_max_fanout
set_max_transition
set_min_capacitance
set_min_fanout
set_operating_conditions
set_port_fanout_number
set_resistance
set_wire_load_min_block_size
set_wire_load_mode
set_wire_load_model
set_wire_load_selection_group

```

Similar to the **write_script** command, the **write_sdc** command writes out commands relative to the top of the design, regardless of the current instance. SDC files written by the **write_sdc** command must be read in from the top of the design.

For a complete guide to using SDC with Synopsys applications, see the *Using the Synopsys Design Constraints Format Application Note* in the Documentation on the Web, which is available through SolvNet at <http://solvnet.synopsys.com>.

The usage of some of these commands is restricted when reading SDC. In some cases, some options are not allowed. The **write_sdc** command supports the restricted usage by restricting what is written. The following restriction apply for SDC Version 1.3 or later:

Note: For the **set_port_fanout_number** command, the *-min* and *-max* options are not supported.

When hierarchy has been partially flattened, embedded hierarchy separators can make names ambiguous - it is unclear which hierarchy separator characters are part of the name and which are real separators. Beginning with SDC version 1.2, hierarchical names can be made unambiguous using the **set_hierarchy_separator** SDC command and/or the *-hsc* option available from the **get_cells**, **get_lib_cells**, **get_lib_pins**, **get_nets**, and **get_pins** SDC object access commands.

By default, PrimeTime and Design Compiler writes an SDC file using these features to create unambiguous names. Synopsys recommends that you write SDC files that contain unambiguous names. However, if you are using a third-party application that does not fully support SDC 1.2 or later (that is, it does not support the unambiguous hierarchical names features of SDC), you can suppress these features by setting the **sdc_write_unambiguous_names** variable to *true*.

The following features are added for SDC Version 1.5 or later:

- *-clock* option of the **set_clock_latency** command

- **set_timing_derate** command with all options
- *-clock_path -data_path -rise -fall* options of the **set_max_transition** command
- *-rise_from, -rise_to, -fall_from, -fall_to* options, of the **set_clock_uncertainty** command
- *-object_list* option of the **set_operating_conditions** command
- Synonyms for all of the **get_object** commands
- **get_objects -nocase** support independently with the *-regexp* option
- **get_objects -regexp** support
- **get_objects -of_objects** support for **get_cells/get_nets/get_pins**

EXAMPLES

The following command writes the SDC script to the file top.sdc.

```
prompt> write_sdc top.sdc
1
prompt>
```

SEE ALSO

```
read_sdc(2)
write_script(2)
sdc_write_unambiguous_names(2)
```

write_sdf

Writes a Standard Delay Format (SDF) back-annotation file.

SYNTAX

```
string write_sdf  
[-version sdf_version]  
[-no_cell_delays]  
[-no_timing_checks]  
[-no_net_delays]  
[-input_port_nets]  
[-output_port_nets]  
[-significant_digits digits]  
[-enabled_arcs_only]  
[-no_internal_pins]  
[-instance inst_name]  
[-context sdf_context]  
[-map sdf_map_file_list]  
[-annotated]  
[-levels level]  
[-no_edge]  
[-compress compression]  
[-include include_list]  
[-exclude exclude_list]  
[-no_negative_values values_list]  
[-no_edge_merging arc_type_list]  
[-delay_calculation_only_mode]  
file_name
```

Data Types

<i>sdf_version</i>	string
<i>digits</i>	int
<i>inst_name</i>	string
<i>file_name</i>	string
<i>sdf_context</i>	string
<i>sdf_map_file_list</i>	list
<i>level</i>	int
<i>compression</i>	string

ARGUMENTS

-version *sdf_version*
Selects which SDF version to use. Supported SDF versions are 1.0, 2.1, and 3.0. SDF 2.1 is the default.

-no_cell_delays
Indicates that no cell delays are be written in the SDF file. By default, all cell pin-to-pin delays are written to the SDF file. Cell delays include the load delay of the cell. Following the SDF conventions, only cell input pin to cell output pin delays are written. In case one cell output is unbuffered, delays are usually represented in libraries by a delay from an output pin to

another output pin. Because this is not allowed by the SDF convention, the SDF delay for an unbuffered output is specified from cell inputs.

-no_timing_checks

Indicates that no cell timing checks are to be written in the SDF file. By default, all cell timing checks (for example, setup, hold, recovery, and removal) are written to the SDF file.

-no_net_delays

Indicates that no net delays are to be written in the SDF file. By default, all net pin-to-pin delays are written to the SDF file.

-input_port_nets

Indicates that the SDF file is to include delays of nets connected to input ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available. If the *-instance* option is specified, all net delays across the instance boundary leading to a pin inside the instance are included instead. The pin must be found on any of levels 1 to *level* of hierarchy if the *-levels level* option is specified.

-output_port_nets

Indicates that the SDF file is to include delays of nets connected to output ports of the current design. By default, these delays are not written to the SDF file because the external connectivity information for ports is not available. If the *-instance* option is specified, then all net delays across the instance boundary leading from a pin inside the instance are included instead. The pin must be found on any of levels 1 to *level* of hierarchy if the *-levels level* option is specified.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be written in SDF delay triplets. Allowed values are 0-13; the default is 3.

-enabled_arcs_only

Indicates that the SDF file is to contain delays only of enabled timing arcs, and is not to include delays of currently-disabled timing arcs. By default, delays of all timing arcs in the design are written to the SDF file, whether they are disabled or enabled.

-no_internal_pins

Indicates that the SDF file is not to include delay timing arcs from or to internal pins. Timing arcs to or from internal pins are expanded into delays from and to primary input and output of the given cell.

-instance *inst_name*

Specifies that the SDF file is to be written only for the instance named *inst_name*. By default, all pin names are relative to the *inst_name*. However, if boundary net delays are included (using the *-input_port_nets* or *-output_port_nets* option) all pin names are relative to the top design. Note that in general, if the *-input_port_nets* or *-output_port_nets* option is specified, boundary nets are written leaf-to-leaf and do not start or end on hierarchical pins. If boundary nets are required to start or end on hierarchical pins, see the **write_physical_annotations** command.

`-context sdf_context`

Specifies the context for writing bus names in SDF. Valid values are Verilog, VHDL, or none (the default). In the verilog context, when pin names are displayed, the last two square bracket characters ("[" and "]") are not escaped. In the VHDL context, the last two parenthesis characters "(" and ")") in a pin name are not escaped. In the default context none, all bus-delimiting characters are escaped with a backslash character ("always escaped. When used with the `-map` option, the `-context` option also affects the way names are printed in mapped SDF files. In the Verilog context, names are printed in `%s[%d]` format; in the VHDL context, names are printed in `%s(%d)` format.

Note: Names are affected only if they are mapped using the `bus(name_to_be_changed)` function in the mapping file.

`-map sdf_map_file_list`

Specifies a list of mapping files the SDF writer is to use when writing out the SDF file. A mapping file contains a user-specified format for printing SDF cell delays and constraints. When writing out SDF for a cell, the SDF writer takes the user-specified mapping, if present, to print out SDF for the cell. If no user-specified mapping is present for a cell, the SDF writer writes out SDF in the normal way.

`-annotated`

Indicates that the SDF is to include only timing arcs that have been annotated with the **read_sdf**, **set_annotated_delay**, or **set_annotated_check** command.

`-levels level`

Specifies the number of levels of hierarchy for which the SDF is written out. Level 1 means only the top design or `inst_name`. Value of N means all levels of hierarchy, 1 to N. By default, all levels of hierarchy are written out. Note that boundary net delays (using the `-input_port_nets` or `-output_port_nets` options) typically have some net arcs from or to pins outside the `inst_name` option. The location of such outside pins is not limited by the `-levels` option. That is, the `-levels` and `-instance` options let you choose which boundary arcs are included, but do not restrict where the arcs lead outside of the `inst_name` option.

`-no_edge`

Indicates that the generated SDF is not to include any edges (posedge or negedge) for both combinational and sequential IOPATHs.

`file_name`

Specifies the name of the SDF file to be written.

`-compress compression`

Specifies a format to be used to compress the file. The only valid value for `compression` is `gzip`. By default, files are not compressed.

`-include include_list`

Specifies a list of constructs to include in the SDF file; these replace one or more constructs from the set of default constructs. Allowed values are one or more of the following:

- **SETUPHOLD**, which indicates that all **SETUP** and **HOLD** constructs are to be replaced by **SETUPHOLD** constructs. If a pair of setup and hold arcs are found

between the same pin edges, timing information for the/both arc/arcs is written in a single SETUPHOLD construct. If a single setup/hold arc is found then the arc will be written in a single SETUPHOLD construct with no timing information for the hold/setup portion. SETUPHOLD supports negative values and can be written only for versions 2.1 and 3.0.

- RECREM, which indicates that all RECOVERY and REMOVAL constructs are to be replaced by RECREM constructs. If a pair of recovery and removal arcs are found between the same pin edges, timing information for both arcs is written in a single RECREM construct. If a single recovery/removal arc is found then the arc will be written in a single RECREM construct with no timing information for the removal/recovery portion. RECREM supports negative values and can be written only for version 3.0.

-exclude *exclude_list*

Specifies a list of timing values of construct types to be either excluded from the SDF file in order to reduce its size, or to be replaced by another construct, as in the case of condense. Allowed values are one or more of the following:

- constant_nets, which indicates that nets are to be omitted from the SDF file if they propagate a constant.
- constant_delay_arcs, which indicates that delay arcs are to be omitted from the SDF file if they propagate a constant, either from case analysis or logical inputs.
- default_cell_delay_arcs, which indicates that all default cell delay arcs are to be omitted from the SDF file if conditional delay arcs are present. If there are no conditional delay arcs, the default cell delay arcs are written to the SDF file.
- wlm_load_delay, which indicates that net delays and cell delays calculated using WLM are to be omitted from the SDF.
- checkpins, when library compiler finds both combinational and sequential arcs between pins, a checkpin is created so that all arcs are expanded in the db so that a single arc pinA->pinB is replaced by the combination of a positive unate arc pinA->pinAcheckpin1 with zero delay and an arc pinAcheckpin1->pinB with the same sense and values as the original arc. When this option is set the SDF is written out as if all checkpins were never created.
- no_condense, indicates that PrimeTime will not use the condense statement to write out default iopaths. By default PrimeTime will replace default iopaths with the condense construct. Specifying this option will result the condense statement being replaced by a default iopath. This option should be used for generating simulator compatible SDF.

-no_negative_values

Specifies a list of timing value types whose negative values are to be zeroed out when writing to the SDF file. Allowed values for timing values are timing checks, cell delays and net delays. This option should be used when the sdf file is intended for simulator use. Using this option leads to inaccurate delay estimation in PrimeTime, so the user should use caution with this

option.

-no_edge_merging

Specifies a list of arc types which are not to be compressed in the SDF file through edge merging. Allowed values are one or more of the following

- **timing_checks**, which indicates that timing checks are not to be compressed in the SDF file through edge merging.
- **cell_delays**, which indicates that cell delays are not to be compressed in the SDF file through edge merging.

-delay_calculation_only_mode

This option is intended for speeding-up flows that use PrimeTime for delay calculation. The **write_sdf** command first checks if the design is in the updated state. If it is, it simply writes the already computed delays and leaves the design in the updated state, regardless of the presence of the **-delay_calculation_only_mode** option. If the design is not in the updated state, by default, the command triggers the **update_timing** command, which brings the design in the updated state. However, if the **-delay_calculation_only_mode** option is used, PrimeTime does not trigger the **update_timing** command; it triggers only the operations that are necessary for writing the SDF file; the design is left in the not-updated state.

DESCRIPTION

This command writes leaf cell pin-to-pin timing information to a disk file. Timing information is written in SDF format using versions v1.0, v2.1 or v3.0. The timing file contains data associated with the netlist from which it is created.

By default, the file is written as simple text; you can use the **-compress** option to compress the file. In some cases, if you omit the file extension, the **write_sdf** command appends it to *file_name*. For example, for gzip formatted files, the **write_sdf** command appends **.gz** to the file name if **.gz** was not specified.

Use the **write_sdf** command only when the instance names in the design agree with the naming conventions of the system to which the timing file is written.

Timing information can be written in multiples of ns, ps, and us. The time unit in the SDF file is that specified in the technology library, and is written in the timing file under "timescale". If there is no time unit in the library, the unit is assumed to be a multiple of nanoseconds.

For efficiency, PrimeTime merges rise/fall delay triplets in SDF, if they are identical to within a very small tolerance.

EXAMPLES

The following example writes timing information for the *MULT16* design to a disk file called *mult16.sdf*, using SDF version 2.1.

```
pt_shell> write_sdf -version 2.1 mult16.sdf
```

You can use the `-level` option to write out inter-block net delays. Assume a design with three hierarchical cells, `h`, `h/h1`, and `h/h2`; and two leaf cells, `h/h1/u1` and `h/h1/u2`. The following command writes out all arcs that begin or end inside `h` but outside `h/h1` and `h/h2`.

```
pt_shell> write_sdf -levels 1 -instance h h.sdf
```

The following example writes out the same arcs as the previous example, plus arcs that cross the boundary of `h`. Note that either the from or the to pin must be outside `h` and the other inside `h`. Therefore, the command does not write a feedthrough net through `h` on this particular design. Similarly, a net arc representing net `h/h1/u1/Y -> h/h1/out -> h/h1/in -> h/h1/u2/A` is not written out, because both pins `h/h1/u1/Y` and `h/h1/u2/A` are inside `h/h1`; that is, both are outside the region defined by the `-level 1 -instance h` option.

```
pt_shell> write_sdf -levels 1 -instance h -input_port_nets  
-output_port_nets h.sdf
```

SEE ALSO

```
read_sdf(2)  
set_annotated_check(2)  
set_annotated_delay(2)  
remove_annotated_check(2)  
remove_annotated_delay(2)  
report_annotated_check(2)  
report_annotated_delay(2)
```

write_sdf_constraints

Writes to a disk file the constraints for the place and route layout tools.

SYNTAX

```
int write_sdf_constraints
[-version 1.0 | 2.1]
[-max_paths max_path_number]
[-nworst nworst_number]
[-slack_lesser_than slack_value]
[-cover_design]
[-from from_list
    | -rise_from rise_from_list
    | -fall_from fall_from_list]
[-through through_list]
[-rise_through through_list]
[-fall_through through_list]
[-to to_list]
[-significant_digits digits]
[-compress compression]
[-context sdf_context]
file_name
```

Data Types

<i>max_path_number</i>	int
<i>nworst_number</i>	int
<i>slack_value</i>	float
<i>from_list</i>	list
<i>rise_from_list</i>	list
<i>fall_from_list</i>	list
<i>through_list</i>	list
<i>rise_through_list</i>	list
<i>fall_through_list</i>	list
<i>to_list</i>	list
<i>digits</i>	int
<i>compression</i>	string
<i>sdf_context</i>	string
<i>file_name</i>	string

ARGUMENTS

.IP "-version 1.0 | 2.1" Specifies the SDF version in which timing information is to be written. The default is 2.1.

-max_paths *max_path_number*

A positive, nonzero integer that specifies the maximum number of timing paths to select for each path group. The default is 1, meaning that only the path with the smallest timing slack is selected.

-nworst *nworst_number*

A nonzero integer that specifies the maximum number of paths to consider at

write_sdf_constraints

each end point. The default is 1, meaning that only the worst path to an endpoint is considered. This option is ignored if you use the *-cover_design* option.

-slack_lesser_than slack_value

A positive or negative floating point number that specifies the maximum slack value to use in selecting paths. Paths with slack values greater than *slack_value* are not selected. Slack is calculated as the required arrival time minus the actual arrival time, so a negative slack value implies a constraint that has not been met.

-cover_design

Indicates to generate just enough path timing constraints to cover the worst path through every driver-load pin pair in the design. Path and net timing constraints are then based on this path set. The *-nworst*, *-to*, *-from*, and *-through* options are ignored when you use this option.

-from from_list

Specifies a list of pins or ports. Constraints are written to the constraints file for paths that start at the pins or ports on *from_list*. This option is ignored if you use the *-cover_design* option.

-rise_from rise_from_list

This option is similar to the *-from* option, except that the path must have a rising transition on the objects specified. This option is ignored if you use the *-cover_design* option.

-fall_from fall_from_list

This option is similar to the *-from* option, except that the path must have a falling transition on the objects specified. This option is ignored if you use the *-cover_design* option.

-to to_list

Specifies a list of pins or ports. Constraints are written to the constraints file for paths that end at the pins or ports on *to_list*. This option is ignored if you use the *-cover_design* option.

-through through_list

Specifies a list of path throughpoints (ports, pins, cells, or nets). Constraints are written to the constraints file for paths that go through points in the *through_list*. This option is ignored if you use the *-cover_design* option.

You can use multiple *-through*, *-rise_through*, and *fall_through* options in a single command to specify paths that traverse multiple points in the design. The order in which these options are specified is respected. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

-from A1 -through B1 -through C1 -to D1

If more than one object is specified within one *-through*, *-rise_through*, or *fall_through* option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

-from A1 -through {B1 B2} -through {C1 C2} -to D1

-rise_through *rise_through_list*

This option is similar to the *-through* option, except that the path must have a rising transition on the objects specified. This option is ignored if you use the *-cover_design* option.

-fall_through *fall_through_list*

This option is similar to the *-through* option, except that the path must have a falling transition on the objects specified. This option is ignored if you use the *-cover_design* option.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point that are to be written in SDF delay triplets. Allowed values are 0-13; the default is 3.

-compress *compression*

Specifies a format to be used to compress the file. The only valid value for *compression* is *gzip*. By default, files are not compressed.

-context *sdf_context*

Specifies the context for writing bus names in SDF. Valid values are *verilog*, *vhdl*, or *none* (the default). In the *verilog* context, when pin names are printed, the last pair of square bracket characters ("[" and "]") is not escaped. In the *vhdl* context, the last pair of parenthesis characters "(" and ")") in a pin name is not escaped. In the default context *none*, all bus-delimiting characters are escaped with a backslash character ("always escaped).

file_name

Specifies the name of the output file to which constraints for the current design are to be written.

DESCRIPTION

Writes specified constraints for the current design to a disk file. If you specify the *-cover_design* option, just enough paths are generated so that the worst path through every driver-load pin pair is covered. The *-from*, *-rise_from*, *-fall_from*, *-to*, *-through*, *-rise_through*, *-fall_through*, and *-nworst* options are ignored when the *-cover_design* option is used, and constraint generation is based on this "covering" path set. Using the *-max_paths* option with *-cover_design* is not recommended, because the objective of the *-cover_design* option is to obtain a path set that covers the entire design.

To write constraints for the current design, arrival totals and slacks must be available throughout the design, not just at endpoints. To achieve this, you should set the **timing_save_pin_arrival_and_slack** variable to *true* when this command is issued. If this variable has not been set to *true*, the command automatically sets it to *true* and updates the design timing before the command executes.

If you intend to use this command it is recommended that you set the

timing_save_pin_arrival_and_slack variable to *true* before the first timing update, thus preventing the cost of an additional timing update.

EXAMPLES

The following example writes constraints in SDF 2.1 format for the most critical path in each path group in the current design.

```
pt_shell> write_sdf_constraints cstr.sdf
```

The following example writes a constraints file in SDF 1.0 format for the 10 most critical paths in each path group in the current design "counter".

```
pt_shell> write_sdf_constraints -max_paths 10 -version 1.0 counter_cstr.sdf
```

The following example writes a constraints file in SDF 2.1 format with just enough paths so that the worst path through every driver-load pin pair is covered.

```
pt_shell> write_sdf_constraints -cover_design counter_cstr.sdf
```

SEE ALSO

`write_sdf(2)`

write_spice_deck

Writes to a SPICE deck the paths or arcs generated by the **get_timing_paths** or **get_timing_arcs** command.

SYNTAX

```
int write_spice_deck
[-align_aggressors]
[-analysis_type type]
[-header header_file_name]
[-initial_delay delay]
[-logic_one_name v1name]
[-logic_one_voltage v1]
[-logic_zero_name v0name]
[-logic_zero_voltage v0]
[-minimum_transition_time trans]
[-output file_name]
[-sub_circuit_file spice_sub_circuit_file]
[-sweep_size number_of_points]
[-sweep_step num]
[-time_precision precision]
[-transient_size tran_size]
[-transient_step tran_step]
[-use_probe]
[-user_measures user_measure_list]
[-sample_size number_of_samples]
paths_arcs_list
```

Data Types

<i>header_file_name</i>	string
<i>delay</i>	float
<i>v1name</i>	string
<i>v1</i>	float
<i>v0name</i>	strong
<i>v0</i>	float
<i>trans</i>	float
<i>file_name</i>	string
<i>spice_sub_circuit_file</i>	string
<i>number_of_points</i>	unsigned
<i>num</i>	float
<i>precision</i>	unsigned
<i>tran_size</i>	float
<i>tran_step</i>	float
<i>number_of_samples</i>	int

ARGUMENTS

-align_aggressors

Applies only to a *net* timing arc. The relative switching time of active aggressors on the net arc compute by the crosstalk delay or noise is written out in the corresponding piecewise linear (PWL) statement. It is effective

only if the net has a coupled RC network annotated. The victim switches after the *initial_delay* and the active aggressors switches relative to that. SPICE uses the calculation engine to get the worst case alignment, and uses a similar setup so that relative alignment is valid. For example, the filtered aggressors are not considered as effective during calculation and their coupling capacitances are grounded.

-analysis_type *type*

Specifies the type of crosstalk or noise analysis for the generated SPICE deck. The possible crosstalk delay types are *max_rise*, *max_fall*, *min_rise*, and *min_fall*. The possible noise types are *above_high*, *above_low*, *below_high*, and *below_low*. This option has no effect on the timing path. The default value for a timing arc is *max_rise*.

-header *header_file_name*

Specifies the path to the user header file whose content is copied to the generated SPICE deck. You can use this file to identify the SPICE deck, to include the library files, or to copy text to a SPICE deck for any other purposes to facilitate the SPICE run.

-initial_delay *delay*

Specifies the initial delay, in library unit, added to all PWL statements. The default value is the longest clock period, or 1.0 library unit for asynchronous designs. Note: Setting *delay* to zero makes generating a ramp difficult and is not recommended.

-logic_one_name *vame*

Specifies the name of the default upper rail voltage source. The default is VDD.

-logic_one_voltage *v1*

Specifies the upper-rail of the voltage swing of the gate input pins. This is used in the PWL and voltage sources generated by the command. The default value is main library voltage.

-logic_zero_name *vame*

Specifies name of the default lower-rail voltage source. The default is VSS.

-logic_zero_voltage *v0*

Specifies the lower-rail of voltage swing of the gate input pins. The default value is 0 volts.

-minimum_transition_time *trans*

Specifies the minimum transition time in nanoseconds (ns), to be used in all generated PWL if the transition time computed by PrimeTime is smaller than *trans*. The default value is 0.001 ns. Transition times less than 0.0001 ns are not recommended.

-output *name*

If the *-sample_size* option is not used, this option specifies the name of the SPICE deck file to be written for the first timing path. SPICE deck files related to subsequent timing paths are also based on this name. If the *-sample_size* option is used, this option specifies the name of the directory to be created for writing the sampled SPICE deck files.

`-sub_circuit_file spice_sub_circuit_file`
 Specifies the path to the file that contains all the SPICE .subckt definitions of logic gates in the timing paths. By default, a subcircuit call uses the pin order in the Synopsys .lib file. Use this option if the SPICE subcircuit has a different pin order from that of the .lib file.

`-sweep_size number_of_points`
 Use in conjunction with the `-align_aggressors` option. Indicates the number of sweep points to be generated for each active aggressors of the net arc. The number of simulations increases geometrically with the number of the active aggressor.

`-sweep_step num`
 Use in conjunction with the `-align_aggressors` and `-sweep_size` options. Indicates the maximum time interval between sweep points generated for each active aggressors of the net arc. The unit is in nanoseconds. The default is 0.1 ns.

`-time_precision precision`
 Specifies the number of precision digits for time in the PWL generated. The default value is 6, and the range is from 1 to 20.

`-transient_size tran_size`
 Specifies the total transient time used in the SPICE .tran statement. The unit is in nanoseconds.

`-transient_step tran_size`
 Specifies the transient step size used in the SPICE .tran statement. The unit is in nanoseconds. The default is 0.001 ns.

`-use_probe`
 Use the .probe statement to output the node voltage instead of the .print statement.

`-user_measures user_measure_list`
 Use to add your own measures instead of the ones generated automatically by the SPICE deck. The empty *user_measure_list* could be used as a way to remove all auto generated .measure and .print statements from SPICE deck.

`-sample_size`
 Specifies the number of SPICE deck files that have to be created while performing variation-aware timing analysis. This option takes the name of the directory through the `-output` option and creates multiple SPICE deck files that correspond to various samples of the variations defined.

`paths_arcs_list`
 Specifies the collection of timing paths or timing arcs that their circuits are written out.

DESCRIPTION

This command writes out a number of SPICE decks, one file for each timing path or timing arc from the collection generated by the **get_timing_paths** or **get_timing_arcs** command. For example, if you specify the `-output timing_path.spo` option for the

first timing path, the second file name is `timing_path_00001.spo`, the third file is `timing_path_00002.spo`, and so on. The files contain the circuit generated and the side pin voltages. The **`write_spice_deck`** command also puts the corresponding stimulus, such as PWL and PULSE statements, in a separate stimulus files. The example of the stimulus file names are `timing_path_stim.spo`, `timing_path_00001_stim.spo`, ..., etc. The stimulus file is included by the main circuit file by the SPICE `".include"` statement.

If you specify a subcircuit file using the `-sub_circuit_file` option, all lines except `".include ..."` and `".subckt ..."` are ignored. The file on the `.include` line is further opened and parsed for other `.include` and `.subckt` lines. The pin order of each `.subckt` line is recorded and applied when a subcircuit call is generated. If a gate used is not in the subcircuit file, the `.lib` pin order is used. A warning is generated if a pin in the subcircuit definition is not found in the `.lib` file. This mismatched pin is written out to the SPICE deck in the same order as it appears in the subcircuit file. An error message is generated when a `.lib` gate pin is not found in the subcircuit definition.

If you want PrimeTime to sensitize the side input of the gates that take the timing path into consideration, use the **`get_timing_paths -justify`** command. Otherwise, the side input pins are sensitized locally. Note that using this command can result in long runtime.

The voltage used in PWL and PULSE statements depend on the pin that the voltage source is attached to unless the **`library_thresholds_use_main_lib`** variable is set to `true`.

The timing PWL voltage sources are defined by the **`set_library_driver_waveform`** command. The default is synopsys pre-driver. The **`set_library_driver_waveform`** command defines the shape of the waveform the timing library is characterized with.

The `-align_aggressors` option applies only to active aggressors of the net timing arcs. The non-active or filtered aggressors are set to be quiet and their coupling capacitors grounded. To find the active aggressors of a victim, use **`report_delay_calculation -cross_talk`** or **`report_noise_calculation`**. While the `-aligned_aggressors` option is set, the `-sweep_size` option with more than zero sweeping points writes out the PWL in sweep form. A third file with the sweep data is generate. An example of the name of this file is `timing_arc_sweep.spo`. It is also included by the main circuit file.

The sweep points center around the aligned aggressor switching time generated

without the `-sweep_size` option. If an even number is given, one more sweep point on the smaller switch time side is generated. The time difference between the sweep points are controlled by the `-sweep_step` and `-sweep_size` options and the aggressor sweep range is computed by the crosstalk delay engine. The lesser of the $(\text{sweep\_range})/(\text{number\_of\_sweep\_points}-1)$ for each aggressor and the `-sweep_step` option is used.

Be very careful with the number of points specified by the `-sweep_size` option. The number of SPICE simulation grows geometrically larger as the number of the active aggressors increases. For example, three sweep points for five active aggressors results in 3^5 (243) simulations.

The `-user_measures user_measure_list` option, allows you to create specific measure statements instead of automatic generated measures. The `user_measure_list` option is a list of user-defined measures. Each user-defined measure can be one of the following types,

```
{delay from pin/port_name to pin/port_name [meas_name]}
{slew pin_or_port_name [meas_name]}
{noise_peak pin_or_port_name [meas_name]}
{user_string "user given string"}
```

The **write_spice_deck** command uses these user-defined measure instructions and creates SPICE .measures with proper trip points and switching directions. The SPICE measures are in the same order as the measure generated when using the `user_measure_list` option. When this option is provided, the user-defined measure overrides the automatic .measure and

You could give zero or more measures in the list. When there is no user-defined measure in the list, for example, when `-user_measures { }`, SPICE deck does not have any kind of measures in the user_string. The user-defined string is inserted as it is. The **write_spice_deck** command does not verify the validity of the user string. For the `delay`, `slew`, and `noise_peak` types, you can give a name to the measurement as `meas_name`. If it is missing, the `write_spice_deck` command automatically generates a name of the measurement. If the user-defined measure does not match with any of the above format, the command fails.

If you specify the `-sample_size` option while performing variation aware timing analysis, a directory is created and multiple SPICE deck files are written with sampled values inside it.

To use the **write_spice_deck** command, you must have a PrimeTime SI license.

EXAMPLES

The following example writes a SPICE deck for timing paths modified by **get_timing_paths -justify**, using the my_subckt.sp file as the source of all SPICE .subckt definitions of all gates in the timing paths. The other two examples are for timing arcs.

```
pt_shell> write_spice_deck -output path.spo \  
          -sub_circuit_file my_subckt.sp [get_timing_paths -justify]  
  
pt_shell> write_spice_deck -output path.spo \  
          -analysis_type above_low \  
          -sub_circuit_file my_subckt.sp [get_timing_arcs -from U1/A]  
  
pt_shell> write_spice_deck -output ./arc.spi \  
          -analysis_type max_rise \  
          -align_aggressors -sweep_size 3 -sweep_step 0.01 \  
          [get_timing_arc -from I1_1/Z -to I1_2/A ]
```

This example shows how to generate measure statement to measure the slew at the path output and delay from input to output.

```
pt_shell> write_spice_deck [get_timing_path -from [get_port i2] \  
          -to [get_port o2]] -user_measures { {slew o2 my_out_slew } \  
          {delay i2 o2 path_delay} {user_string ".print o2"} }  
  
          .measure tran my_out_slew  
          + trig v(o2) val = 2.16 td = 7.5e-09 fall = 1  
          + targ v(o2) val = 0.54 td = 7.5e-09 fall = 1  
          .measure tran path_delay  
          + trig v(i2) val = 1.35 td = 7.5e-09 fall = 1  
          + targ v(o2) val = 1.35 td = 7.5e-09 fall = 1  
          .print o2
```

1

SEE ALSO

```
get_timing_paths(2)  
get_timing_arcs(2)  
report_delay_calculation(2)  
report_noise_calculation(2)  
set_library_driver_waveform(2)
```