

Verification Continuum™

VCS Native Low Power (NLP) User Guide

R-2020.12-SP2, June 2021

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2021 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Third-Party Software Notices

VCS® and configurations of VCS includes or is bundled with software licensed to Synopsys under free or open-source licenses. For additional information regarding Synopsys's use of free and open-source software, refer to the third_party_notices.txt file included within the <install_path>/doc directory of the installed VCS software.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>. All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

Customer Support	3-xxxiii
Synopsys Statement on Inclusivity and Diversity	3-xxxiv
1. Getting Started	
Introduction to Low Power Simulation	1-2
Low Power Design Strategies	1-3
Increasing Challenges of Power	1-3
Types of Power Consumption	1-5
Dynamic Power	1-5
Static (Leakage) Power	1-7
Power Reduction Methods	1-9
Supply Voltage Reduction	1-9
Clock Gating	1-10
Multiple-Vt Library Cells	1-11
Multi-Voltage Design	1-12
Power Gating	1-14
Dynamic Voltage and Frequency Scaling	1-18
Specifying Power Intent of the Design Using UPF	1-19

2. Low Power Design Concepts

Power Architectural Elements	2-1
Library Requirements for Low Power Designs.	2-2
Level-Shifter Cells	2-2
Isolation Cells	2-3
Power Switch Cells.	2-4
Always-On Logic Cells	2-5
Retention Register Cells.	2-6
Power Domains	2-7
Power Distribution Elements	2-10
Supply Ports	2-10
Supply Nets	2-12
Power Switch	2-13
Supply Sets	2-14
Supply Set Handles	2-15
Power State Tables	2-17

3. Using UPF Commands to Specify Power Intent

UPF Commands Supported by VCS NLP	3-4
Basic Power Network Commands	3-5
associate_supply_set.	3-5
create_power_domain	3-6
create_power_switch	3-6
create_supply_port.	3-7
create_supply_net	3-8
connect_supply_net	3-8
create_supply_set	3-9

map_power_switch.....	3-9
set_domain_supply_net	3-10
Level Shifter Commands	3-11
set_level_shifter.....	3-11
map_level_shifter_cell	3-12
name_format	3-12
Isolation Commands	3-13
set_isolation.....	3-14
set_isolation_control.....	3-15
map_isolation_cell	3-15
Retention Commands	3-16
set_retention.....	3-17
set_retention_control	3-18
set_retention_elements	3-19
map_retention_cell.....	3-19
Power State Table Commands	3-20
create_pst.....	3-20
add_port_state	3-21
add_power_state	3-21
create_power_state_group.....	3-22
add_pst_state.....	3-22
Logic Editing Commands.....	3-23
create_logic_net.....	3-23
create_logic_port	3-23
connect_logic_net	3-24
Query Commands	3-24
find_objects	3-25
query_cell_instances	3-26
query_cell_mapped	3-26

query_power_domain_element	3-27
query_net_ports	3-27
query_port_net	3-27
query_pst	3-28
query_pst_state	3-28
query_power_switch	3-28
query_upf	3-29
query_design_attributes	3-29
query_hdl2upf_vct	3-30
query_isolation	3-30
query_power_domain	3-30
query_pg_info_cell	3-30
query_retention	3-31
query_retention_control	3-31
query_supply_net	3-31
Simulation/Verification Extension	3-32
set_design_top	3-32
set_partial_on_translation	3-32
Utility Commands	3-33
load_upf	3-33
set_scope	3-33
set_design_attributes	3-34
set_port_attributes	3-34
Extending Switch	3-35
set_power_switch	3-36
Merging Power Domains	3-36
merge_power_domains	3-36
Other Supported Power Intent Commands	3-37
bind_checker	3-38

create_composite_domain	3-38
create_hdl2upf_vct	3-39
create_upf2hdl_vct	3-39
describe_state_transition	3-40
add_state_transition	3-40
load_simstate_behavior	3-41
load_upf_protected	3-41
set_design_top	3-42
set_equivalent	3-42
set_repeater	3-42
set_simstate_behavior	3-43
upf_version	3-43
Using add_power_state Command	3-44
Use Model	3-44
Supply Expressions and Logic Expressions	3-46
Power State Evaluation of Supply Set in Presence of -supply_expr/-logic_expr	3-46
Defining Power State Tables Using the add_power_state Command 3-47	
Use Model	3-48
Defining PSTs Using add_power_state Options	3-50
Composing States Hierarchically	3-52
Examples	3-53
Interaction Between create_pst/add_pst_state and add_power_state Syntax	3-56
Limitations	3-59
Defining Power States for a Power Domain	3-59
Use Model	3-60
Referring Power Domain States in Group Logic Expression	3-62

Usage Examples	3-63
Error Scenarios	3-63
Limitations of add_power_state Support	3-65
Using associate_supply_set Command	3-66
Support for the UPF 2.1 associate_supply_set Command ..	3-66
Support for the UPF 3.0 associate_supply_set Command ..	3-67
Supply Set Association	3-69
Same Scope Association	3-70
Association Across Scopes	3-70
Checks for Supply Set Association	3-70
Update Supply Set Functions with Supply Nets.....	3-71
Common Ancestor Requirement for associate_supply_set	3-73
Using add_state_transition Command	3-75
Limitations	3-77
Using bind_checker Command	3-77
Using create_power_switch Command.....	3-81
Handling create_power_switch -ack_port Connections.....	3-81
Using Supply Set of the Power Switch for Acknowledge Port Corruption.....	3-82
Using create_supply_net Command	3-82
Support for -resolve parallel Option	3-83
Limitation	3-83
Specifying User-Defined Supply Net Resolution As a Verilog Function or Task in the Package	3-84
Limitations	3-85

Resolution Mechanism for Supply Net With “strong” and “weak” Resolutions	3-86
Resolution Mechanism for Supply Net With “strong” Resolution	3-86
Resolution Mechanism for Supply Net With “weak” Resolution	3-88
Design Namespace Conflict Resolution for the Creation of Supply Ports/Nets.	3-90
Using map_isolation_cell Command	3-90
Key Points to Note	3-91
Isolation Cell Modeling Example	3-92
Using Value Conversion Table (VCT) for Model Supply Connection	3-93
Limitations	3-95
Using query_cell_mapped Command	3-95
Limitations	3-96
Querying Power Domain Information of an Instance.	3-96
Use Model	3-97
Expected Output for Specific Scenarios	3-98
Example	3-99
Limitations	3-102
Using set_design_top Command	3-103
Using set_equivalent Command	3-103
Example	3-104
Limitations	3-104
Using set_isolation Command.	3-105
Use Model	3-105

Examples	3-108
Precedence Details	3-112
Defining Isolation Strategies	3-114
Error Checking for Isolation Strategy with -location parent on Top-level Ports	3-114
Using set_repeater Command	3-115
Effective Element List and Precedence Rules	3-119
Supported Insertion Semantics	3-120
Naming of Repeater	3-121
Interaction with set_port_attributes -repeater_supply	3-122
Error Scenarios	3-122
Limitations	3-123
Using set_related_supply_net Command	3-123
Example	3-124
Limitations	3-125
Using set_port_attributes Command	3-125
Use Model	3-126
Attributes	3-127
Limitations	3-128
Comparison with set_related_supply_net (SRSN)	3-128
set_port_attributes on Non-Boundary Ports	3-129
Ignoring SRSN/SPA Buffers Specified on Intermediate Boundary Ports	3-130
Using the -repeater_supply Option	3-130
Support for Unconnected Ports Inside Macro Blocks	3-132
Specifying Clamp Value for Ports	3-134

Inserting SPA and SRSN Buffers on HighConn	3-135
Defining Retention Strategies	3-136
Including Current Scope in the Power Domain	3-137
Examples	3-137
Supply Equivalence Checks by VCS NLP	3-139
Functionally Equivalent Supply Sources with Conflicting Port States	3-140
Functionally Equivalent Supply Sources with Missing Port States	3-141
Functionally Equivalent Supply Sources Having Conflicting Type	3-142
PST Entry with Conflicting Port States	3-143
Supply Source State Does Not Match with the State of Equivalent Supply Source	3-144
Using upf_version Command	3-146
Using upf_tool Variable	3-146
Usage Example	3-147
Refining Power Domain Elements	3-148
Power Domain With Empty Elements List	3-148
Support for Negative Voltages	3-148
Key Points to Note	3-148
Example	3-149
Using synopsys_program_name Variable	3-150
Usage Example	3-150
Handling Conflicting Port State Names in the add_port_state Command	3-151

Setting Supply Port Default Value	3-154
Use Model	3-154
Example	3-155
Driving Supply Expression Using Logic Expression	3-155
Use Model	3-155
Key Points to Note	3-156
Support for Auto-Completion of UPF Commands	3-156
Support for Feedthrough Paths Inside Macro Blocks	3-157
Key Points to Note	3-158
Limitations	3-159
Generic Handles Support	3-159
Generic Handles Support in Retention Strategy	3-160
Limitations	3-162
Generic Handles Support in the bind_checker Command	3-162
Generic Handles Support for Isolation in the bind_checker Command	3-162
Generic Handles Support for Retention in the bind_checker Command	3-166
Bias Support for Low Power Designs	3-168
Enabling Bias Feature	3-169
Establishing Connections to Bias Pins/Supplies	3-172
Controlling Assertions	3-173
Examples	3-174
Querying Power Domains to Find Always-On and Collapsible Power Domains.	3-175
Querying Supply Information of a Cell Instance	3-176

Limitations	3-178
Searching the Logic Hierarchy for the Supply Ports	3-178
Key Points to Note	3-179
Supply Equivalence Checks by VCS NLP.....	3-181
Functionally Equivalent Supply Sources with Conflicting Port States	3-182
Functionally Equivalent Supply Sources with Missing Port States 3-183	
Functionally Equivalent Supply Sources Having Conflicting Type 3-184	
PST Entry with Conflicting Port States.....	3-185
Supply Source State Does Not Match with the State of Equivalent Supply Source	3-186
Interpretation of '/' in the UPF Commands	3-188
Definitions	3-188
Rules of Interpretation of '/' in the UPF	3-188
Examples.....	3-190
UPF Structural Components	3-191
UPF Script Examples	3-192
Simple Multi-Voltage Design	3-193
Bottom-Up Power Intent Specification.....	3-193
Top-Down Power Intent Specification	3-194
Supply Set Example	3-195
4. Wildcard Support in UPF Commands	
Wildcard Usage with Isolation	4-4

Wildcard Usage with Retention	4-4
Wildcard Usage with set_design_attributes.....	4-5
Wildcard Usage with set_related_supply_net	4-5
Wildcard Usage with connect_supply_net	4-5
Treating Generate Instance as a Single Token for Wildcard Matching	4-6
Limitations of Wildcard Support.	4-7
5. Supply Set Based Power Intent Specification	
Converting an Existing Supply Net Based UPF to a Supply Set.	5-2
Explicit Supply Sets with Implicit Supply Nets.....	5-4
Implicit Supply Sets.....	5-4
Supply Network Initialization	5-5
Supply Set Association	5-7
Disabling the Propagation of Supply Set State Names to Functional Nets	
5-7	
Backward Compatibility	5-10
Valid Usage Examples.....	5-11
Error Scenarios	5-12
Supply Sets in Strategies	5-14
Limitation	5-14
Setting Power State Using set_supply_set_state Function from Testbench	5-14
Simstates.	5-15

6. Low Power Simulation

Running Low Power Designs	6-2
VCS NLP Low Power Simulation Flow	6-2
Using Low Power in VCS Two-Step Flow	6-3
Specifying Power Top.	6-4
Power Techniques Supported in VCS	6-5
Using Low Power in VCS Three-Step Flow	6-5
Power Techniques Supported in VCS Three-step Flow	6-6
Limitations	6-6
HDL Constructs and Techniques in VCS.	6-7
Support for UPF Supply Functions	6-8
UCLI Support for Supply Functions	6-10
Voltage-Based supply_on	6-11
Monitoring the Simstate of a Power-Managed Instance	6-12
Initial Block Retriggering	6-14
Enabling Initial Block Retriggering in VCS NLP.	6-14
Initial Block Retriggering Supported Attributes.	6-15
Reinit Property	6-16
Transitive Reinit.	6-17
Examples for Different Reinit Scenarios	6-17
Transitive Reinit Limitations	6-18

7. Compile-Time Static Reports

Redirecting/Renaming mvsim_native_reports Directory	7-2
Key Points to Note	7-2

Example	7-2
Compile-Time Static Report for set_design_attributes	7-3
Default Reports Dumped Under mvsim_native_reports Directory	7-4
flop_inference.rpt	7-7
hierarchical_flop_inference.rpt	7-8
hierarchical_flop_inference_vhdl.rpt	7-9
isolation_association.rpt	7-10
isolation_association_summary.rpt	7-11
isolation_insertion_info.txt	7-12
library_mapping.rpt	7-13
library_mapping_after_upf.rpt	7-14
library_match.rpt	7-16
repeater_port_association.rpt	7-17
srsn_spa_association.rpt	7-18
upf_supply_set_power_state.rpt	7-19
latch_inference.rpt	7-20
8. Handling Corruption in Low Power Designs	
Simulation Semantics for Corruption	8-2
Overriding Corruption with HDL Forces	8-3
Random Corruption	8-4
Usage	8-4
Key Points to Note	8-5
Examples	8-6
Limitation	8-8
Preventing Corruption of Memories During Power Shutdown	8-8
Limitations	8-8

Controlling the Corruption and Wake-up Semantics of the Initial and Always Blocks	8-9
Treating Continuous Assignments as Pass Through.....	8-11
Limitations	8-12
Configuring Semantics for Runtime Forces During Power Shutdown and Wake-up.....	8-12
Force Priority	8-13
Force and Corrupt	8-14
Flush Forces	8-14
Defer Forces	8-15
Use Model	8-16
Limitations	8-17
CORRUPT_ON_ACTIVITY Simstate (Low-Vdd Standby)	8-17
Examples.....	8-18
Simulation Behavior.....	8-18
Limitations	8-19
Using UPF_dont_touch Attribute.....	8-19
Use Model	8-20
Module Support.....	8-21
Module Named Block Support.....	8-21
Module Signal Support	8-22
Support for a Module Inside a Library	8-22
Usage Examples	8-23
Limitations	8-23
Skipping Corruption.....	8-24

Corrupting Isolation Cell Output on Assertion/De-assertion of Isolation Enable	8-26
Use Model	8-28
Example	8-29
Setting User-Defined Values for the Corrupted Real Data Type Signals	
8-31	
Precedence Details	8-33
Key Points to Note	8-34
Limitations	8-35
Power Black Box	8-35
Use Model	8-36
UPF Requirements	8-37
Boundary Constraints	8-38
UPF Commands	8-39
9. Isolation Support	
Isolation Support for Mixed Designs	9-2
Limitations	9-2
Isolation Support for SystemVerilog Design Constructs	9-3
Limitations	9-3
Isolating Interface Modports	9-4
Limitations	9-4
Skipping Isolation on the Undriven/Floating Ports	9-4
Inserting NOR-Style Isolation Cells as per UPF 3.0 LRM	9-5
Use Model	9-6
Key points to note	9-6

Corruption Semantics for NOR-Style Isolation	9-7
Limitations	9-8
Support for Conservative diff_supply_only Isolation	9-8
Name-Based Isolation Cell Association in Netlist	9-10
Support for Isolation Terminal Points at the Hierarchical Boundaries	9-12
Introduction	9-12
Use Model	9-12
Enhancement to the -driver_supply and -receiver_supply Port Attributes	9-13
Examples	9-14
Example-1: Isolation with -source	9-14
Example-2: Isolation with -sink	9-15
Example-3: Isolation with receiver_supply	9-17
Example-4: Isolation with driver_supply	9-18
Limitations	9-19
Path-Based Isolation Support for Heterogeneous Loads	9-20
Key Points to Note	9-21
Heterogeneous Load Scenarios	9-21
Precedence Rule Scenarios	9-25
Case-1	9-25
Case-2	9-27
Support for -exclude_elements and -update Options	9-29
Use Model	9-29
Using -update with set_isolation Command	9-30
Support for -applies_to with -elements for Isolation	9-31
Key Points to Note	9-31

Support for Specifying Instances with <code>set_isolation</code> -elements	9-33
Fanout Isolation Instrumentation Support for Signals of SV Interface/Modport	9-34
Generating Virtual Isolation Reports	9-34
Virtual Isolation Insertion Report	9-35
Reporting Format	9-35
Virtual Isolation Inference Report.	9-36
Isolation Inference Report Format	9-36
Isolation/Corruption on Ports with Logical Expressions in Port Map	9-38
Limitations	9-38
Supporting Lower Domain Boundary for Isolation.	9-39
Use Model	9-40
Key Points to Note	9-40
Examples	9-41
Block-Level Control of Lower Domain Boundary	9-42
Support for OFF-Within-ON Integration	9-43
Examples	9-44
Support for ON-Within-OFF Integration	9-44
Use Model	9-45
Example	9-45
Location Parent Strategy Support on Lower Domain Boundary	
9-46	
Restricting Application of Filters to Specified Boundary	9-47
Limitation	9-50
Behavior of the Isolation Commands.	9-50
Commands Supported for Delayed Error Check	9-50

Impact on Isolation Engine	9-51
General isolation strategies (without -elements)	9-51
Specific Isolation Strategies (with -elements)	9-52
Inserting Back-to-Back Isolation Cells in the Lower Domain Boundary	9-53
Order of Precedence of Isolation Strategies	9-54
Limitations	9-54
Implementing Assertion Checks for Isolation Cells	9-55
Updating Isolation Supplies.....	9-56
Using Isolation Power and Ground Nets With the -update Option 9-56	
Using -isolation_supply_set With the -update Option	9-58
Specifying Isolation Strategies Without Defining Isolation Enable Example	9-60
9. Handling of Constants for Corruption and Isolation	
Default Behavior	10-1
Skip Corruption on Constants	10-2
10. State Retention Support	
Introduction	11-2
Using set_retention command.....	11-3
Support for the -update Option	11-4
Use Model	11-4
Using -no_retention Option	11-5
Using -use_retention_as_primary Option	11-5

Supported Retention Schemes	11-6
Two-Pin Retention	11-6
Semantics	11-7
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-7
Single-Pin Retention	11-8
Semantics	11-9
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-9
Zero-Pin Retention	11-11
Conditions	11-14
<code>-save_condition/-restore_condition</code>	11-14
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-14
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-15
<code>-retention_condition</code>	11-16
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-16
Mutexes	11-17
<code>assert_s_mutex</code>	11-17
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-18
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-19
<code>assert_r_mutex</code>	11-20
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-21

Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-22
<code>assert_rs_mutex</code>	11-22
Example: <code>outp1</code> is the primary register, <code>save_outp1</code> is the shadow register	11-23
Parameters	11-24
Supported Parameters with <code>set_retention_control</code>	11-25
Supported Parameters with <code>set_retention</code>	11-30
Creating Retention List	11-32
Key Points to Note	11-33
Usage Examples	11-34
Invalid Usage Scenarios	11-35
Limitations	11-36
Ignoring Latches From Retention	11-37
Automated Assertions Supported for Retention	11-37
Generic Handles	11-37
Hierarchical Flop Inference Report	11-38
Limitations of Retention Support	11-39
12. Unified Low Power Messaging	
Message Format	12-2
Message Control	12-2
Runtime Configuration File	12-3
Logging Mode	12-3
Turn ON/OFF Messages	12-4

Override the Default Severity of Messages	12-4
Filter Messages Based on Severity	12-5
Function Calls from Testbench.....	12-5
Override the Default Severity of Messages	12-6
Turn ON/OFF Messages	12-6
Filter Messages Based on Severity	12-6
Wildcard Support.....	12-7
Example	12-7
User-Defined Messages in LP_MSG Format	12-7
Register Custom Message.....	12-8
Printing the Registered Messages	12-8
Checking if the Registered User-Defined Message is ON/OFF 12-9	
Power Aware Verification Environment	12-9
Custom Assertions.....	12-10
Use Model	12-10
Specifying File List with Multiple Bind Checker Tcl Files During Compilation	12-11
Writing Custom Low Power Assertions	12-12
Example.....	12-14
Compile-Time UPF Parser Checks	12-17
Lint Check for Explicit Primary Supplies for a Domain.....	12-18
Error Checks for Non-Existing Design Objects Referred in UPF.....	12-19
Lint Check for Mismatch Between Verilog and Liberty Definitions of a Cell	12-21

[13. Debugging Low Power Designs](#)

[14. Coverage Support for Low Power Objects](#)

Creating Covergroups for the Low Power Objects	14-2
Enabling Coverage on Low Power Objects.....	14-3
Runtime Control for Low Power Coverage	14-6
Excluding UPF Objects from Low Power Coverage	14-6
Use Model	14-6
Example.....	14-7
Limitations	14-8
Constraining Low Power Coverage Using the describe_state_transition Command	14-9
Key points to note.....	14-11
Limitations	14-12
Examples	14-12
Error Conditions	14-13
Example of describe_state_transition	14-13
Post-Processing Options for Low Power Functional Coverage..	14-16
Filtering Covergroups in URG	14-17
Use Model	14-17
Filtering Covergroups in UCAPI.....	14-19
Coverage Reports	14-20
Low Power Coverage Report View in URG	14-20
Use Model	14-20
Low Power Functional Coverage Report Example	14-24
Examples for Different Scenarios.....	14-25

Dumping Coverage Exclusion Files for Low Power Objects	14-32
15. Partition Compile Support	
Limitations	15-1
16. Gate-level Netlist Simulations (Simulating with Liberty (.db) Files)	
Library Mapping.	16-2
Use Model	16-2
Specifying Library Path Using Environment Variable in the - power_config File	16-3
Following is the use model to specify library path using environment variable in the -power_config file:	16-3
Using Connect Supply Net	16-4
Example	16-4
DB Matching Criteria	16-5
Simulation Semantics for Cells.	16-5
Supported .db Attributes	16-6
Support for Nor-Style Cells	16-7
Power Aware and Non-Power Aware Models	16-8
Simulation Behavior for Power Aware Models.	16-9
Simulation Behavior for Non-Power Aware Models.	16-9
Implicit Supply Net Connections	16-9
UPF2SV/SV2UPF Connections	16-11
SV2UPF Connection	16-11
UPF2SV Connection	16-11

Value Conversion Tables (VCTs)	16-12
Supported VCTs.	16-13
Attributes Supported for Overriding the Default VCTs	16-15
Precedence of VCT in High to Low Order	16-15
Limitations	16-16
User-Defined Value Conversion Tables.	16-16
Using the <code>create_upf2hdl_vct</code> Command.	16-17
Using the <code>create_hdl2upf_vct</code> Command.	16-17
Syntax Checks	16-18
Semantic Checks for the <code>connect_supply_net -vct</code> Command	16-20
Limitations	16-21
Multi-Rail Macro Simulation.	16-21
Use Model	16-21
Usage Example	16-23
Configuring Port-Based Corruption on Macros	16-26
Use Model	16-27
Example	16-28
Association of Pre-Inserted Isolation Cells	16-29
Support for Timing Simulation With SDF in Presence of UPF	16-32
Specifying Default ON Voltage for Internal PG Pins of a Macro	16-34
Example	16-35
17. Connecting UPF Supply Network to HDL Ports	
Support for SPICE Blocks in UPF	17-2
Use Model	17-2

Connecting UPF Supply Network to Wreal Supply Ports	17-3
Connecting UPF Supply Nets to Wreal Input and Inout Ports	17-4
Example	17-5
Connecting UPF Supply Nets to Wreal Output Ports.	17-7
Example	17-8
Connecting UPF Supply Network to SystemVerilog Nettype With Real Base Type	17-11
UPF Supply Nets Driving Real-Valued SystemVerilog Nettype Ports	
17-12	
Example	17-13
Real-Valued SystemVerilog Nettype Ports Driving UPF Supply Nets	
17-16	
Example	17-17
Driving UPF Supply Network Through SPICE.	17-19
Use Model	17-20
Multiple SPICE Output Ports and UPF Drivers Driving the UPF Supply Net	17-20
SPICE Ports Connected to the UPF Supply Net Through HDL	
17-22	
Support for Multiple View Modules for SPICE with UPF	17-23
Use Model	17-25
Supporting UPF in the VCS and CustomSim Cosimulation Flow	17-26
Use Model	17-27
Driving SPICE Supply Ports.	17-28
Using UPF to Specify Supply Pin Connections to the SPICE Blocks	17-28
Driving SPICE Supply Ports Through HDL Supply.	17-30

Isolating Digital/Analog Interface Signals.....	17-32
Key Points to Note.....	17-32
Limitations	17-33
18. Automated Assertions	
Power-on Reset Assertions Support	18-2
Example.....	18-2
Use Model	18-4
Limitations	18-5
Checking for Assertion of Isolation Enable When Source Supply is OFF 18-5	
Key Points to Note	18-6
Limitations	18-6
Identifying the Crossover Ports That do not Toggle.....	18-7
Limitations	18-8
Handling of Isolation Cells on Top-Level Ports with Pad Cells ..	18-9
Reporting Write Attempt on OFF Domain Element	18-10
Supported Scenarios	18-10
Use Model	18-11
Limitations	18-12
19. UPF Encryption Support	
UPF Encryption Using upf_protect1 Option	19-2
UPF Encryption Using gen_ip Binary	19-3
Encrypted UPF File Usage	19-5

20. Generating Region Based SAIF Reports for Power Analysis	
Example	20-9
21. Appendix	
Low Power Assertions	21-2
List of Automated Low Power Messages	21-2
Enabling Runtime Isolation Messages	21-22
Verbose Illegal PST State Change Messages	21-22
Custom Assertion Checking for Clock/Reset During Shutdown	21-26
Supported UPF Query Commands	21-27
query_power_domain	21-27
query_supply_net	21-28
query_pst	21-29
query_pst_state	21-30
query_power_switch	21-31
query_isolation	21-33
query_retention	21-35
query_retention_control	21-37
Warning Messages for Unconnected Supply Ports	21-39
Root Supply Driver	21-41
TFIPC-L Lint Check in Presence of UPF Connections	21-42
22. VCS NLP Quick Reference	
Command Line Options	22-2
Compile-Time Options	22-2
List of -power Compile-Time Options	22-3

Simulation Option	22-11
Design Attributes to be Specified in UPF	22-14
HDL Attributes	22-18

[Feedback](#)

Preface

VCS Native Low Power (VCS NLP) add-on for VCS enables you to specify the UPF based power-intent of your design directly to VCS and generate a simulation model, which contains all power-objects directly instrumented in it.

The preface discusses the following:

- [Customer Support](#)
- [Synopsys Statement on Inclusivity and Diversity](#)

Customer Support

For any online access to the self-help resources, you can refer to the documentation and searchable knowledge base available in SolvNetPlus.

To obtain support for your VCS product, choose one of the following:

- Open a Case through SolvNetPlus.
Go to <https://solvnetplus.synopsys.com/s/contactsupport> and provide the requested information, including:

- **Product - L1** as VCS
- **Case Type**

Fill in the remaining fields according to your environment and issue.

- Send an e-mail message to vcs_support@synopsys.com
Include product name (L1), sub-product name/technology (L2), and product version in your e-mail, so it can be routed correctly.
Your e-mail will be acknowledged by automatic reply and assigned a Case number along with Case reference ID in the subject (**ref:_...:ref**).
For any further communication on this Case via e-mail, **please send an e-mail to vcs_support@synopsys.com and ensure to have the same Case ref ID in the subject header** or else it will open duplicate Cases.

Note:

In general, we need to be able to reproduce the problem in order to fix it, so a simple model demonstrating the error is the most effective way for us to identify the bug. If that is not possible, then please provide a detailed explanation of the problem along with complete error and corresponding code, if any/permissible.

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering

and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

[Feedback](#)

1

Getting Started

Power consumption is a critical aspect of electronic circuit design. You can adopt different methods to reduce the power consumption. This chapter introduces low power simulation, low power principles, power reduction strategies, and UPF commands supported by VCS Native Low Power (VCS NLP). This chapter has the following sections:

- [Introduction to Low Power Simulation](#)
- [Low Power Design Strategies](#)
- [Specifying Power Intent of the Design Using UPF](#)

Introduction to Low Power Simulation

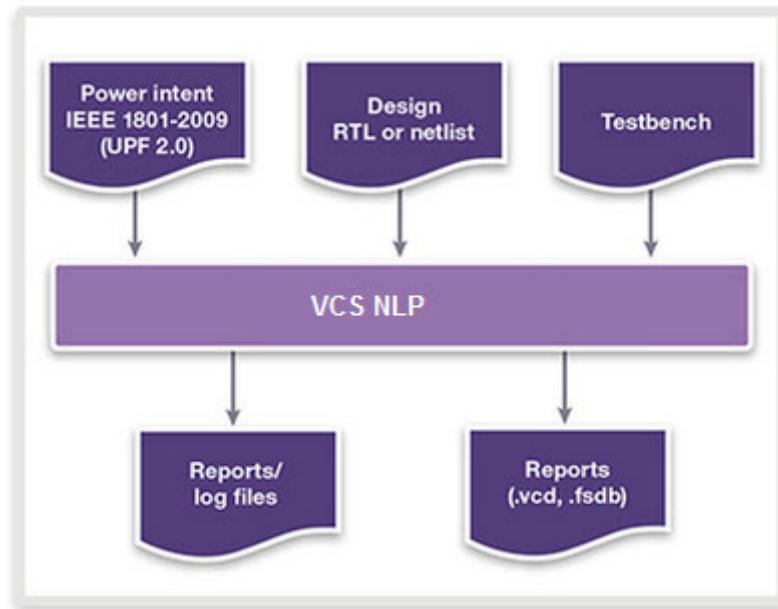
Low power designs have design components operating under different supply voltages from multiple supply rails. Traditional simulators do not have the intelligence to understand the relationship between different supply voltages and what they drive. This can cause potential silicon failures that go undetected in simulation.

VCS NLP equips VCS to natively perform voltage-level aware simulation with a complete understanding of the UPF-defined power network, including at RTL prior to implementation flows. This allows you to comprehensively verify correct behavior of designs that use advanced voltage control techniques for power management, and catch potentially design-killing low power bugs early in the design process.

VCS NLP takes in the same Verilog/VHDL, RTL, or gate-level netlist representation of the design, and accepts the same testbench that is used in the standard flows (optionally augmented for low power checks). However, the native low power flow requires power intent to be specified in an UPF-format file, which is loaded into VCS along with the design and the testbench.

VCS NLP equips VCS to read the UPF, model the entire power network described in the UPF, and accurately understand the low power policies and voltage events. VCS in native low power mode generates a log file as output and provides an error and warnings report for all violations related to multi-voltage checks, as illustrated in [Figure 1-1](#).

Figure 1-1 VCS NLP Low Power Simulation Flow



Low Power Design Strategies

The power principles and power reduction strategies are discussed in the following sections:

- Increasing Challenges of Power
- Types of Power Consumption
- Power Reduction Methods

Increasing Challenges of Power

In earlier generations of IC design technologies, the main concerns were timing and area. EDA tools were designed to maximize the speed while minimizing area. Power consumption was a lesser

concern. CMOS was considered as a low power technology, with fairly low power consumption at the relatively low clock frequencies used at the time, and with negligible leakage current.

In recent years, however, device densities and clock frequencies have increased dramatically in CMOS devices, thereby increasing the power consumption dramatically. At the same time, supply voltages and transistor threshold voltages have been lowered, causing leakage current to become a significant problem. As a result, power consumption levels have reached acceptable limits, and power has become as important as timing or area.

High power consumption can result in excessively high temperatures during operation. Therefore, expensive ceramic chip packaging must be used instead of plastic, and complex and expensive heat sinks and cooling systems are often required for product operation. Laptop computers and hand-held electronic devices can become uncomfortably hot to the touch. Higher operating temperatures also reduce reliability because of electromigration and other heat-related failure mechanisms.

High power consumption also reduces battery life in portable devices such as laptop computers, cell phones, and personal electronics. As more features are added to a product, power consumption increases and the battery life is reduced, requiring a larger, heavier battery or shorter life between charges. Battery technology has lagged behind the increased demands for power.

Another aspect of power consumption is the higher cost of electrical energy used to power millions of computers, servers, and other electronic devices. Even a small reduction in power consumption of a microprocessor or other device used on a large scale can result in large aggregate cost savings to users and can provide significant benefits to the environment as well.

Types of Power Consumption

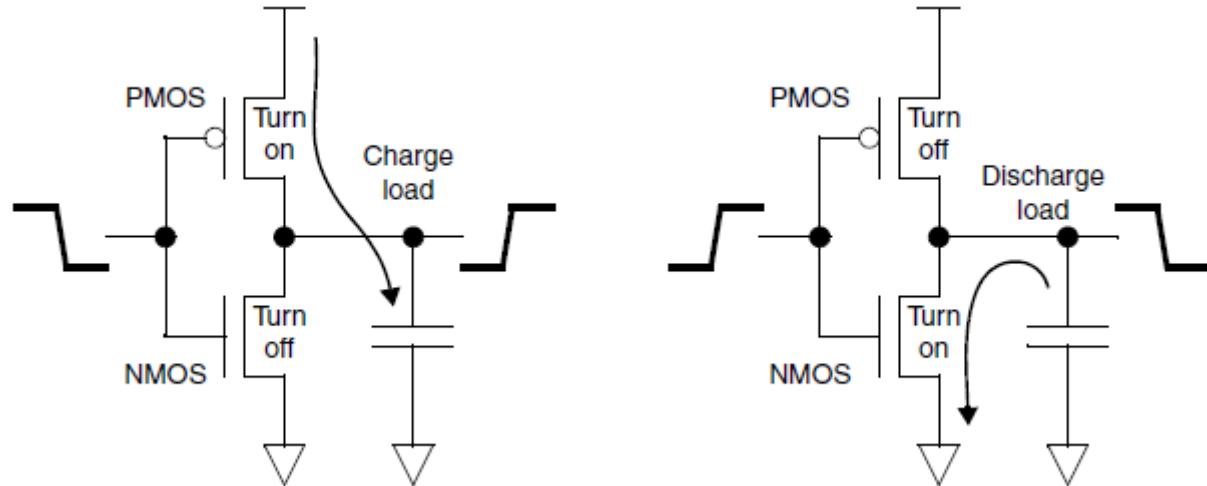
Designers must consider two types of power consumption, dynamic and static. Dynamic power is consumed during the switching of transistors, and hence it depends on the clock frequency and switching activity. Static power is the transistor leakage current that flows whenever power is applied to the device, and it is not related to the clock frequency or switching activity.

Dynamic Power

Dynamic power is the energy consumed during logic transitions on nets, consisting of two components, switching power and internal power. Switching power results from the charging and discharging of the external capacitive load on the output of a cell. Internal power results from the short-circuit (crowbar) current that flows through the PMOS-NMOS stack during a transition.

The cause of switching power is illustrated in [Figure 1-2](#). A transition from 0 to 1 on the output of the inverter charges the capacitive load of the output net through the PMOS transistor. A transition from 1 to 0 discharges the same capacitive load through the NMOS transistor.

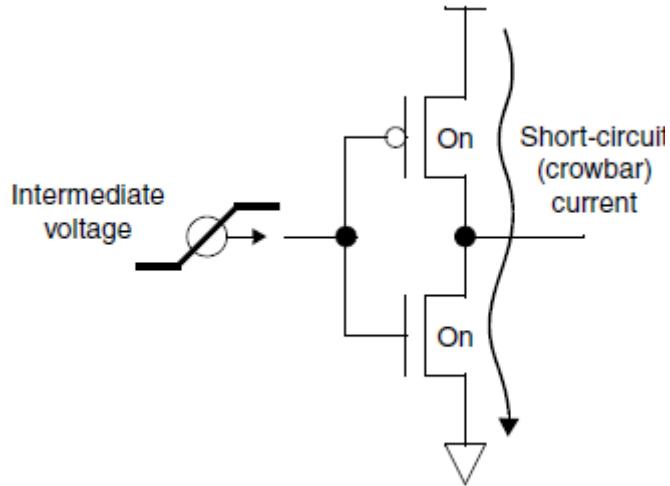
Figure 1-2 Switching Power



The amount of energy dissipated in each transition depends on the supply voltage and the capacitive load of the net. Also, because the current flows only during logic transitions on the net, the long-term dynamic power consumption depends on the clock frequency (possible transitions per second) and the switching activity (presence or absence of transitions actually occurring on the net in successive clock cycles).

Internal power is consumed during the short period of time when the input signal is at an intermediate voltage level, during which both the PMOS and NMOS transistors can be conducting. This condition results in a nearly short-circuit conductive path from VSS to ground, as illustrated in [Figure 1-3](#). A relatively large current, called the crowbar current, flows through the transistors for a brief period of time. Lower threshold voltages and slower transitions result in more internal power consumption.

Figure 1-3 Internal Power

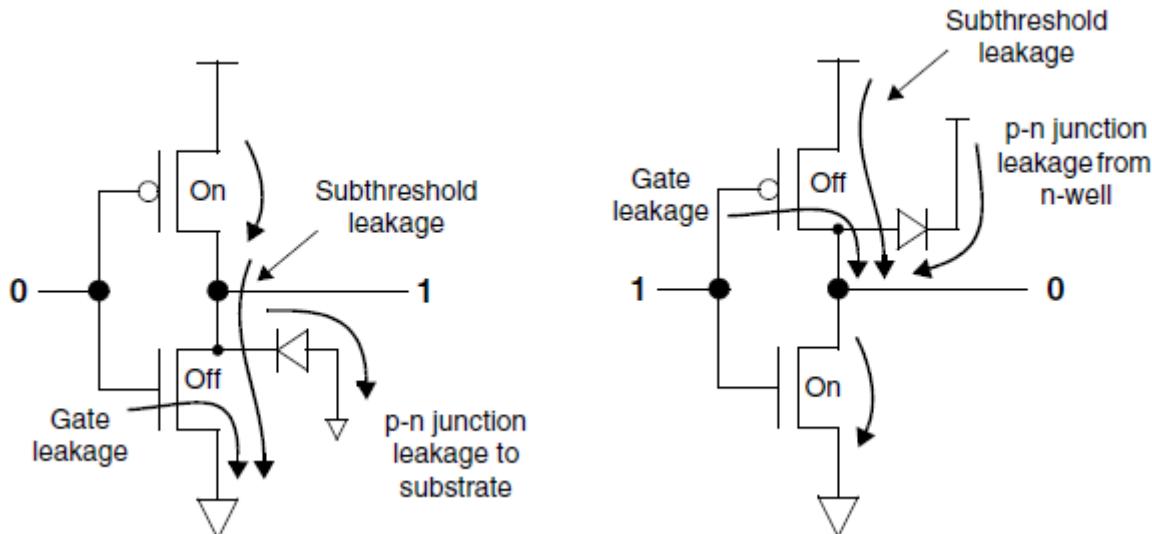


Static (Leakage) Power

Leakage current was negligible in earlier CMOS technologies. However, with shrinking device geometries and reduced threshold voltages, leakage power is becoming increasingly significant, sometimes approaching the levels of dynamic power dissipation.

The main causes of leakage power are reverse-bias p-n junction diode leakage, subthreshold leakage, and gate leakage. These leakage paths in a CMOS inverter are shown in [Figure 1-4](#).

Figure 1-4 Static Leakage Currents



Leakage at reverse-biased p-n junctions (diode leakage) has always existed in CMOS circuits. This is the leakage from the n-type drain of the NMOS transistor to the grounded p-type substrate, and from the n-well (held at VDD) to the p-type drain of the PMOS transistor. This leakage is relatively small.

Subthreshold leakage is the small source-to-drain current that flows even when the transistor is held in the “off” state. In older technologies, this current was negligible. However, with lower power supply voltages and lower threshold voltages, “off” gate voltages are getting close to “on” threshold voltages. Subthreshold leakage current increases exponentially as the gate voltage approaches the threshold voltage.

Gate leakage is the result of using an extremely thin insulating layer between the gate conductor and the MOS transistor channel. Gate oxides are becoming so thin that only a dozen or fewer layers of insulating atoms separate the gate from the source and drain. Under

these conditions, quantum-effect tunneling of electrons through the gate oxide can occur. This results in significant leakage from the gate to the source or drain.

Leakage currents occur whenever power is applied to the transistor, irrespective of the clock speed or switching activity. Leakage cannot be reduced by slowing or stopping the clock. However, it can be reduced or eliminated by lowering the supply voltage or by switching off the power to the transistors completely.

Power Reduction Methods

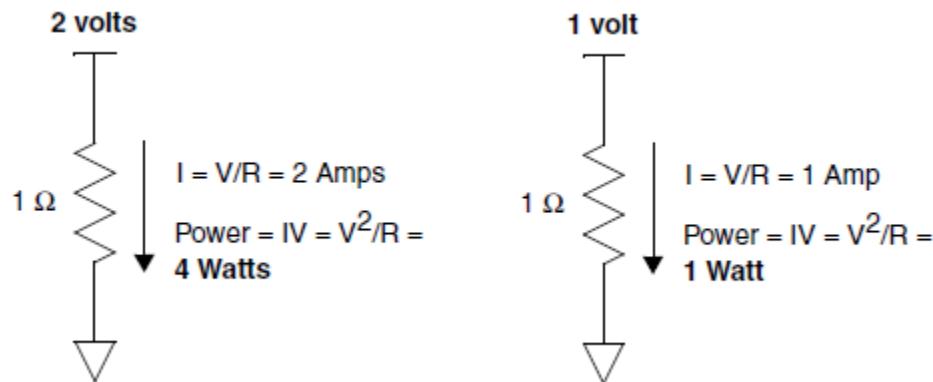
There are multiple RTL and gate-level design strategies for reducing power. The following topics describe the power reduction methods:

- Supply Voltage Reduction
- Clock Gating
- Multiple-V_t Library Cells
- Multi-Voltage Design
- Power Gating
- Dynamic Voltage and Frequency Scaling

Supply Voltage Reduction

The most basic way to reduce power is to reduce the supply voltage. Power usage is proportional to the square of the supply voltage, as shown in example in [Figure 1-5](#). A 50 percent reduction in the supply voltage results in a 50 percent reduction in current and a 75 percent reduction in power. A similar degree of power reduction can be achieved in CMOS circuits for both dynamic and static power.

Figure 1-5 Power Versus Supply Voltage Example

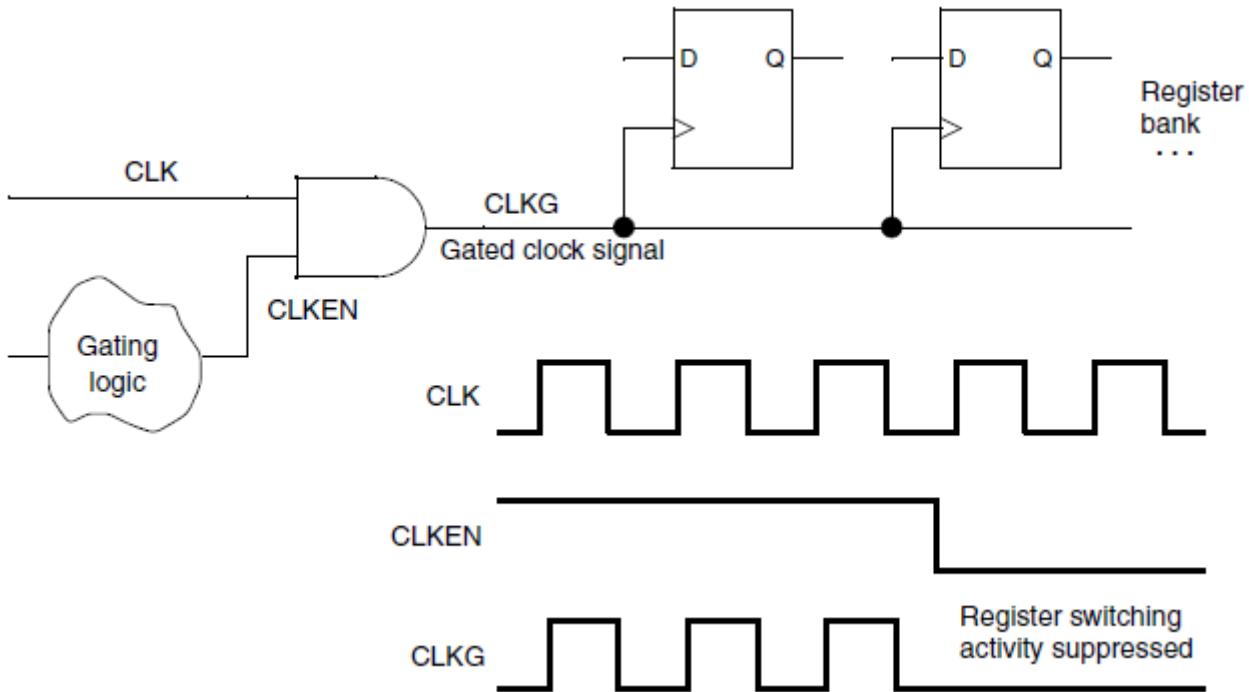


Successive generations of CMOS technologies have used lower and lower supply voltages to reduce power. Starting at 5 volts for compatibility with bipolar TTL circuits in the 1980s, supply voltages have fallen to about 1 volt for advanced technologies. Each lowering of the supply voltage reduces per-gate power consumption, but also lowers the switching speed. In addition, the transistor threshold voltage must be lowered, causing more problems with noise immunity, crowbar currents, and subthreshold leakage. The lower voltage swing makes it more difficult to interface the chip with external devices that require larger voltage swings.

Clock Gating

Clock gating is a dynamic power reduction method in which the clock signals are stopped for selected register banks during times when the stored logic values are not changing. A sample implementation of clock gating is shown in [Figure 1-6](#).

Figure 1-6 Clock Gating Example



Clock gating is particularly useful for registers that need to maintain the same logic values over many clock cycles. Shutting off the clocks eliminates unnecessary switching activity that would otherwise occur to reload the registers on each clock cycle. The main challenges of clock gating are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

Clock gating is a well-established power-saving technique that has been used for years. It is relatively simple to implement because it only requires a change in the netlist. No additional power supplies or power infrastructure changes are required.

Multiple-Vt Library Cells

Some CMOS technologies support the fabrication of transistors with different threshold voltages (V_t values). In that case, the cell library can offer two or more different cells to implement each logic function,

each using a different transistor threshold voltage. For example, the library can offer two inverter cells: one using low-V_t transistors and another using high-V_t transistors.

A low-V_t cell has higher speed, but higher subthreshold leakage current. A high-V_t cell has low leakage current, but less speed.

Multi-Voltage Design

Different parts of a chip might have different speed requirements. For example, the CPU and RAM blocks might need to be faster than a peripheral block. As mentioned earlier, a lower supply voltage reduces power consumption but also reduces speed. To get maximum speed and lower power at the same time, the CPU and RAM can operate with a higher supply voltage while the peripheral block operates with a lower voltage, as shown in [Figure 1-7](#).

Figure 1-7 Multi-voltage Chip Design

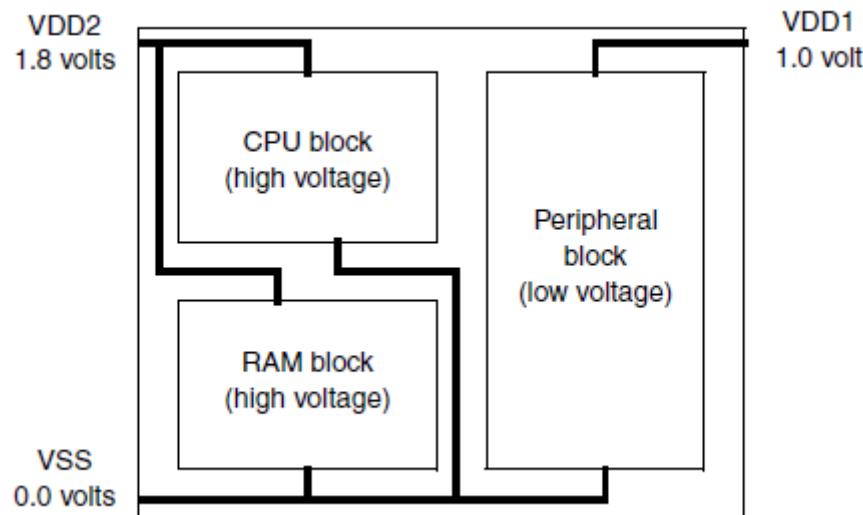
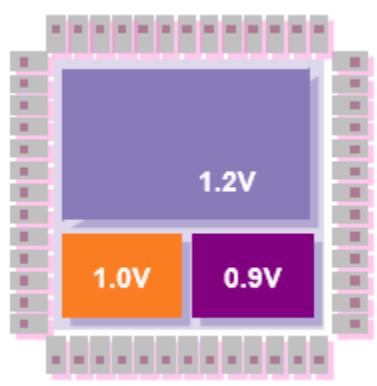


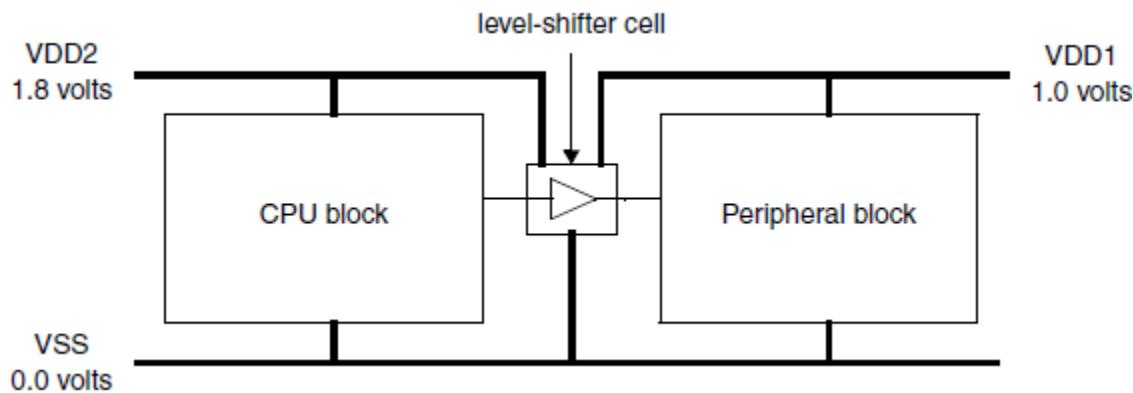
Figure 1-8 Multi-voltage Chip



Providing two or more supply voltages on a single chip introduces some complexities and costs. Additional device pins must be available to supply the chip voltages, and the power grid must distribute each of the voltage supplies separately to the appropriate blocks.

Where a logic signal leaves one power domain and enters another, if the voltages are significantly different, a level-shifter cell is necessary to generate a signal with the proper voltage swing. In the example shown in [Figure 1-9](#), a level shifter converts a signal with 1.8-volt swing to a signal with a 1.0-volt swing. A level-shifter cell itself requires two power supplies that match the input and output supply voltages.

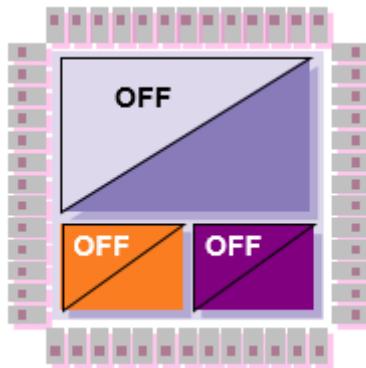
Figure 1-9 Level Shifter



Power Gating

Power gating is a power-saving technique in which portions of the chip are shut down completely during periods of inactivity, as illustrated in [Figure 1-10](#).

Figure 1-10 Power Gating

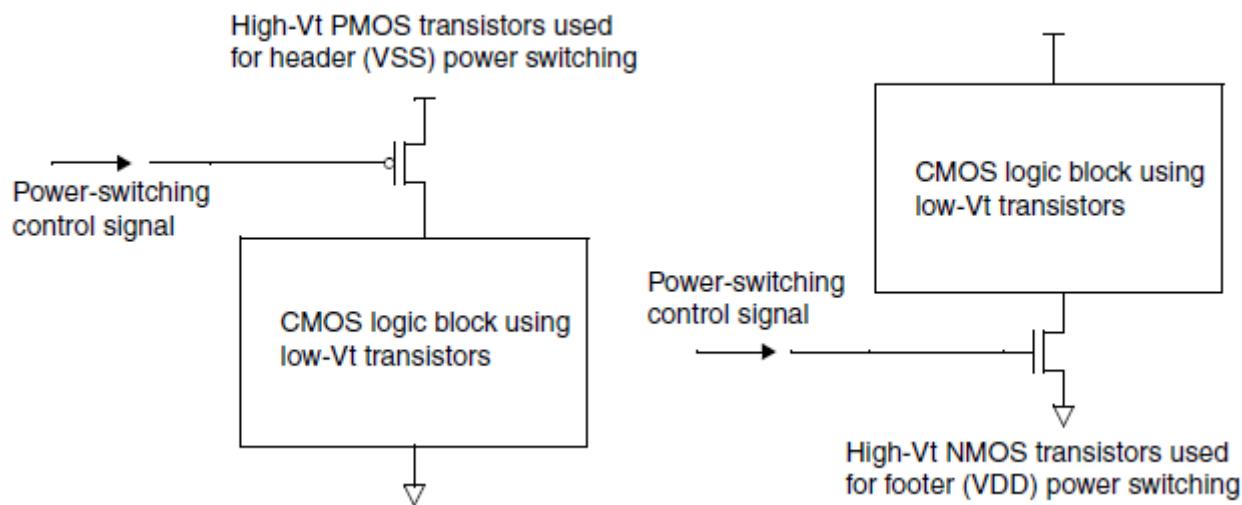


For example, in a cell phone chip, the block that performs voice processing can be shut down when the phone is in standby mode. When the user places a call or receives an outside call, the voice processing block must “wake up” from its powered-down state.

Power gating has the potential to reduce overall power consumption substantially because it lowers leakage power as well as switching power. It also introduces some additional challenges, including the need for a power controller, a power-switching network, isolation cells, and retention registers.

High-V_t transistors from a Multiple-Threshold CMOS (MTCMOS) technology are used for the power switches because they minimize leakage and their switching speed is not critical. PMOS header switches can be placed between V_{DD} and the block power supply pins, or NMOS footer switches can be placed between V_{SS} and the block ground pins, as illustrated in [Figure 1-11](#).

Figure 1-11 Power-Switching Network Transistors

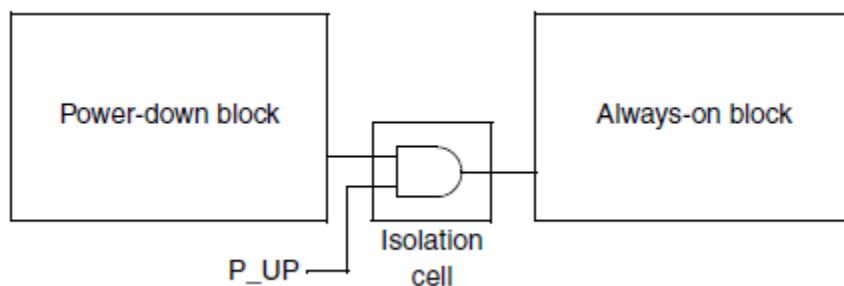


The switching strategy shown in [Figure 1-11](#) is called a coarse-grain strategy because the power gating is applied to the whole block. Multiple transistors in parallel drive a common supply net for the block. In a fine-grain strategy, each library cell has its own power switch, allowing fine-grain control over which cells are powered down. The fine-grain approach has better potential for power savings, but requires significantly more area.

Any use of power gating requires isolation cells where signals leave a powered-down block and enter a block that is always on (or currently powered up). An isolation cell provides a known, constant logic value to an always-on block when the power-down block has no power, thereby preventing unknown or intermediate values that could cause crowbar currents.

One simple implementation of an isolation cell is shown in [Figure 1-12](#). When the block on the left is powered up, the signal P_UP is high and the output signal passes through the isolation cell unchanged (except for a gate delay). When the block on the left is powered down, P_UP is low, holding the signal constant at logic 0 going into the always-on block. Other types of isolation cells can hold a logic 1 rather than 0, or can hold the signal value latched at the time of the power-down event. Isolation cells must themselves have power during block power-down periods.

Figure 1-12 Isolation Cell



The power gating can be combined with multi-voltage operation. Different blocks can be designed to operate at different voltages and also to be separately powered down when they are not needed. In that case, the interface cells between different blocks must perform both level shifting and isolation functions, depending on whether the two blocks are operating at different voltages or one is shut down. A

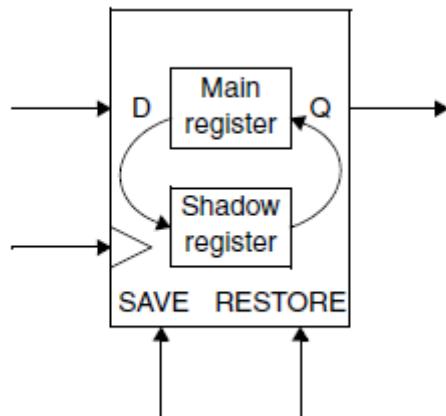
cell that performs both functions is called an enable level shifter. This cell must have two separate power supplies, just like any other level shifter.

When a block is powered down and then powered back up, it is often desirable for the block to be restored to the state it was in before the power-down event. There are several strategies for doing this. For example, the block register contents can be copied to RAM outside of the block before power-down, and then copied back after power-up.

Another strategy is to use retention registers in the power-down block. A retention register can retain data during power-down by saving the data into a shadow register (also known as the bubble register) before power-down. Upon power-up, it restores the data from the shadow register to the main register. The shadow register has an always-on power supply, but it is constructed with high-V_t transistors to minimize leakage during the power-down period. The main register is built with fast but leaky low-V_t transistors.

One type of retention register implementation is shown in [Figure 1-13](#). The SAVE signal saves the register data into the shadow register before power-down and the RESTORE signal restores the data after power-up. Instead of using separate, edge-sensitive SAVE and RESTORE signals, a retention register could use a single level-sensitive control signal.

Figure 1-13 Retention Register



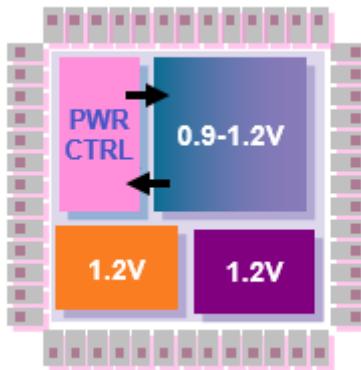
A retention register occupies a larger area than an ordinary register, and it requires an always-on power supply connection for the shadow register in addition to the power-down supply used by the rest of the device. However, restoring the data to the registers after power-up is fast and simple compared with other strategies.

Dynamic Voltage and Frequency Scaling

The principle of multi-voltage operation can be extended to allow the voltage to be changed during operation of the chip to match the current workload.

For example, a math processor chip in a laptop computer might operate at a lower voltage and lower clock frequency during simple spreadsheet computations, thereby saving power; and then at a higher voltage and higher clock frequency during 3-D image rendering when the highest performance is needed. The changing of supply voltage and operating frequency during operation to meet workload requirements is called dynamic voltage and frequency scaling. [Figure 1-14](#) illustrates the dynamic voltage and frequency scaling.

Figure 1-14 Dynamic Voltage and Frequency Scaling



The chip and voltage supply can be designed to use a number of established levels, or even a continuous range. Dynamic voltage scaling requires a multilevel power supply and a logic block to determine the best voltage level to use for a given task. Design, implementation, verification, and testing of the device can be challenging because of the ranges and combinations of voltage levels and operating frequencies that must be analyzed and accommodated.

Dynamic voltage scaling can be combined with power gating technology so that each block in the design can operate at multiple voltage levels for different performance requirements, or shut off completely when not needed at all.

Specifying Power Intent of the Design Using UPF

The IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF), consists of a set of commands used to specify the design intent for multi-voltage electronic systems.

Using UPF commands, you can specify the supply network, power switches, isolation, retention, and other aspects of power management for a chip design. A single set of low power design specification commands can be used throughout the design, analysis, verification, and implementation flow.

VCS NLP supports the industry-standard UPF 1.0 and 2.0 subset that are used to capture low power design requirements. Detailed information about the IEEE 1801 standard can be found in the IEEE 1801 specification itself, available as a download from the IEEE Standards Association at <http://standards.ieee.org>.

2

Low Power Design Concepts

This chapter describes the power elements that are used to specify the design intent for low power designs. It contains the following sections:

- [Power Architectural Elements](#)
- [Power Distribution Elements](#)
- [Power State Tables](#)

Power Architectural Elements

This section contains the following topics:

- [Library Requirements for Low Power Designs](#)
- [Power Domains](#)

Library Requirements for Low Power Designs

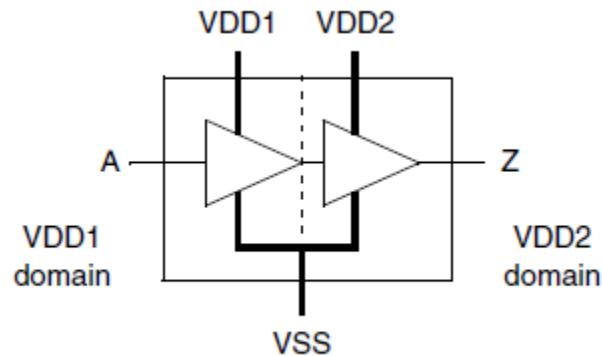
In order to use power-saving strategies such as clock gating, multi-voltage, multiple-V_t library cells, or power gating, the technology libraries used must conform to the Liberty syntax. The library should contain logic cells such as clock-gating cells, level-shifters, isolation cells, retention registers, always-on buffers, and inverters that support these strategies. The following topics describe the types of cells that support low power designs:

- [Level-Shifter Cells](#)
- [Isolation Cells](#)
- [Power Switch Cells](#)
- [Always-On Logic Cells](#)
- [Retention Register Cells](#)

Level-Shifter Cells

A level shifter is required where each signal crosses from one power domain to another. The level shifter operates as a buffer with one supply voltage at the input and a different supply voltage at the output. Thus, a level shifter converts a logic signal from one voltage swing to another, with a goal of having the smallest possible delay from input to output. See [Figure 2-1](#).

Figure 2-1 Level Shifter



Level-shifter cells are of three types:

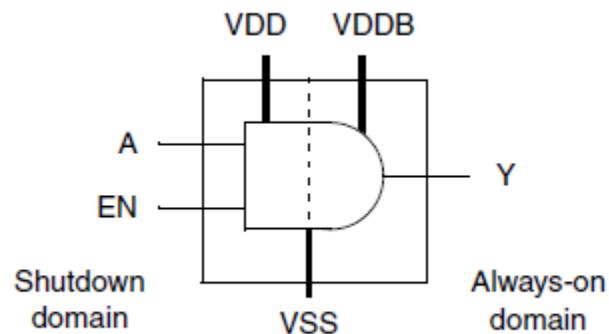
- Level shifters that convert from High voltage to Low voltage (H2L)
- Level shifters that convert from Low voltage to High voltage (L2H)
- Level shifter that can convert H2L and L2H

The library description of a level-shifter cell must have information about the type of conversion performed (high-to-low, low-to-high, or both), the supported voltage levels, and the identities of the respective power pins that must be connected to each power supply.

Isolation Cells

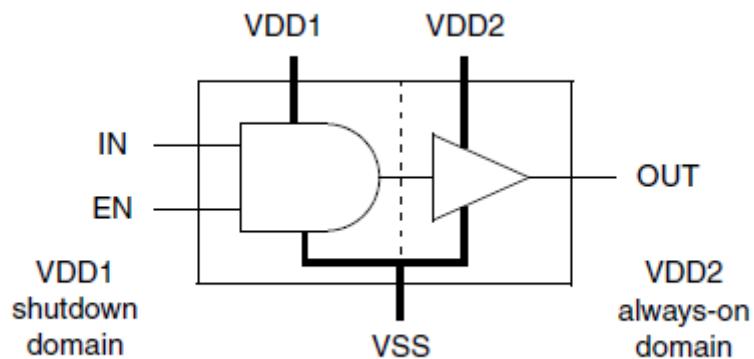
In a design with power gating, an isolation cell is required where each logic signal crosses from a power domain that can be powered down to a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a constant output signal during times that the input side is powered down. An enable input controls the operating mode of the cell. See [Figure 2-2](#).

Figure 2-2 Isolation Cell



A cell that can perform both level-shifting and isolation functions is called an enable level-shifter cell. This type of cell is used where a signal crosses from one power domain to another, where the two voltage levels are different and the first domain can be powered down. See [Figure 2-3](#).

Figure 2-3 Enable Level Shifter

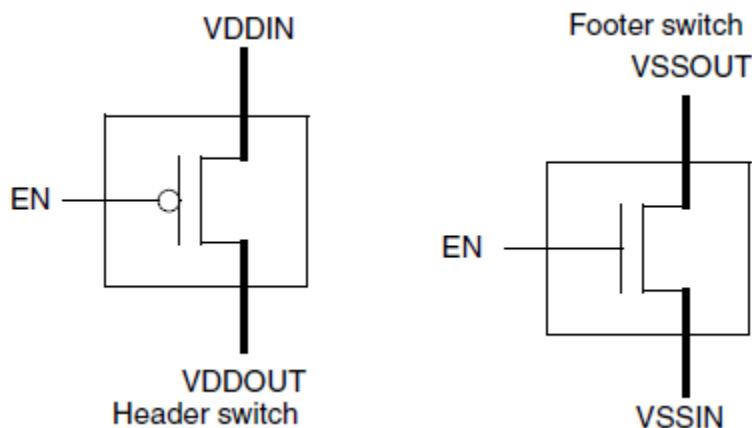


Power Switch Cells

In a design with power gating, either header or footer type power switch cells are required to supply power for cells that can be powered down. A header type power switch connects the power rail

to the power supply pins of the cells in the power-down domain. A footer type power switch connects the ground rail to the ground supply pins of the cells in the power-down domain. An input logic signal to the power switch controls the connection or disconnection state of the switch. See [Figure 2-4](#).

Figure 2-4 Power-Switch Cells



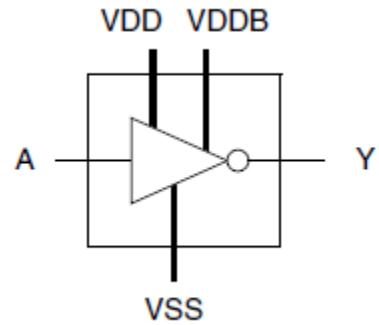
The power-switch cell can optionally have an output “acknowledge” signal indicating the current status of the switch.

Always-On Logic Cells

When dealing with shutdown domains, there can be some situations in which certain cells in the shutdown portion need to continuously stay active, such as for implementing retention registers, isolation cells, retention control paths, and isolation enable paths. For example, if a save signal or restore signal passing through a shutdown voltage area needs buffering, an always-on buffer cell must be used.

This type of logic is called always-on logic, which is built with always-on library cells. Compared to an ordinary cell, a functionally equivalent always-on cell has a backup power supply that operates continuously, even during the shutdown mode. See [Figure 2-5](#).

Figure 2-5 Always-On Logic Cell

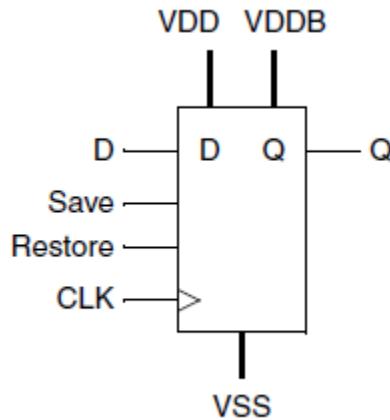


Retention Register Cells

In a design with power gating, there are multiple ways to save register states before power-down and restore them upon power-up in the power-down domain. One method is to use retention registers, which are registers that can maintain their state during power-down by means of a low-leakage register network and an always-on power supply.

The library description of a retention register specifies the power pins and the input signals that control the saving and restoring of data. It also specifies which power pins are normal and can be powered down and which ones are the always-on pins used to maintain the data during power-down. See [Figure 2-6](#).

Figure 2-6 Retention Register



Power Domains

Power domain is a group of elements in the design that share a common power supply distribution network. Each power domain has a scope and an extent. The scope is the hierarchical level at which the domain is defined and is an ancestor of the elements belonging to the power domain, whereas the extent is the actual set of elements belonging to the power domain.

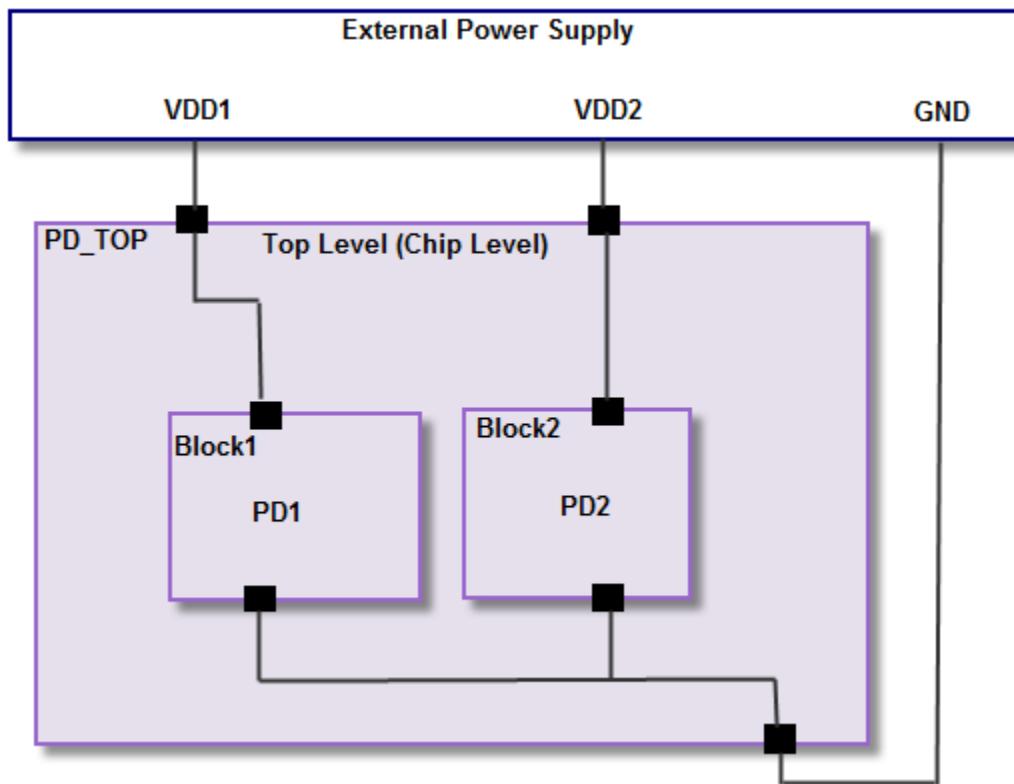
A power domain must have one primary supply net and one primary ground net, and might optionally have additional supply nets, supply ports, and power switches.

A power domain has the following characteristics:

- Name
- Level of hierarchy or scope where the power domain is defined or created
- The set of design elements that comprise the power domain

- Associated set of supply nets that are allowed to be used within the power domain
- Primary power supply and ground nets
- Synthesis strategies for isolation, level-shifters, always-on cells, and retention registers

Figure 2-7 Power Domains



In [Figure 2-7](#), the top-level chip occupies the top-level power domain, **PD_TOP**. The domain **PD_TOP** is defined to have three supply ports: **VDD1**, **VDD2**, and **GND**.

In addition to the top-level power domain, **PD_TOP**, there are two more power domains defined, called **PD1** and **PD2**, created at the levels of two hierarchical blocks, **Block1** and **Block2**, respectively.

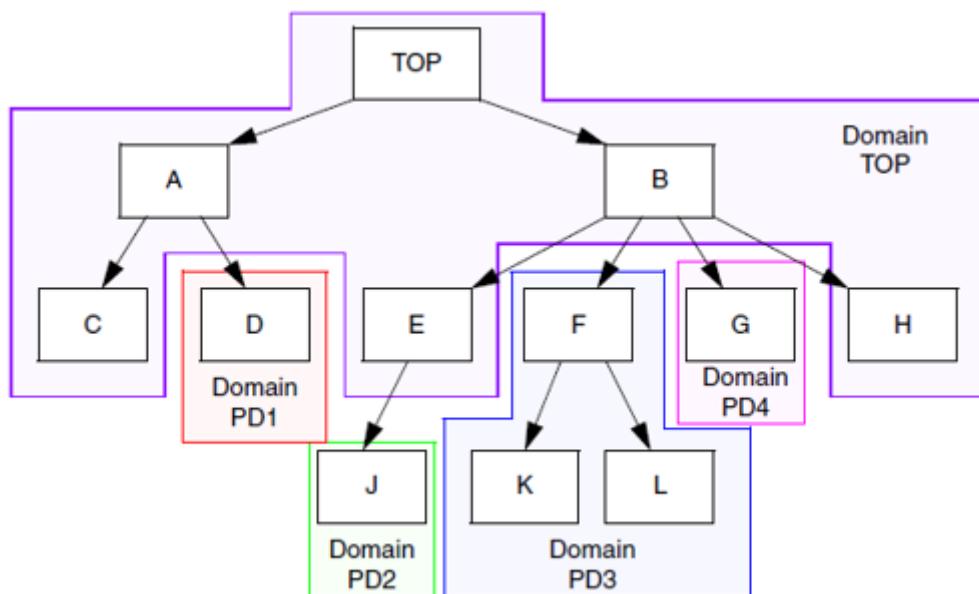
Each block has supply ports (shown as black squares in the diagram) to allow supply nets to cross from the top level down into the block level.

The following commands define the power domains of a design and assign hierarchical blocks to the domains:

```
create_power_domain TOP  
create_power_domain PD1 -elements {A/D}  
create_power_domain PD2 -elements {B/E/J}  
create_power_domain PD3 -elements {B/F}  
create_power_domain PD4 -elements {B/G}
```

The hierarchy and power domain membership of this example are illustrated in [Figure 2-8](#).

Figure 2-8 Power Domains and Hierarchy Example



In this example, the first `create_power_domain` command assigns the TOP design to the power domain called TOP, which causes all the lower-level blocks to also belong to the same power domain. The subsequent `create_power_domain` commands

assign some of the lower-level blocks to different power domains. Lower-level blocks not assigned to other power domains remain in the TOP power domain.

Power Distribution Elements

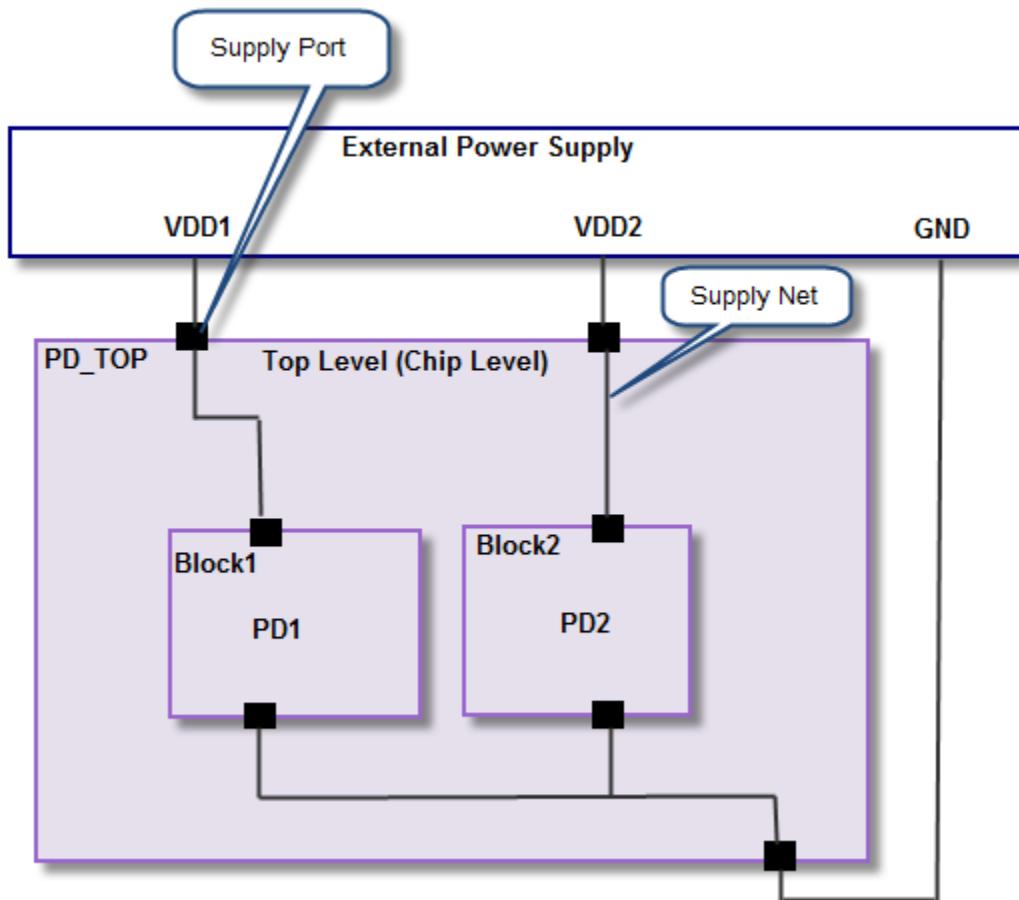
The following topics describe the power elements that allow you to specify the power intent of a design:

- [Supply Ports](#)
 - [Supply Nets](#)
 - [Power Switch](#)
 - [Supply Sets](#)
 - [Supply Set Handles](#)
-

Supply Ports

A supply port is a power supply connection point between two adjacent levels of the design hierarchy, between the parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

Figure 2-9 Supply Ports



In Figure 2-9, the black squares along the border of the power domain represent the supply ports of that domain.

To create the power supply and ground ports, use the `create_supply_port` command.

The name of the supply port should be a simple (non-hierarchical) name and unique at the level of hierarchy it is defined. Unless the `-domain` option is specified, the port is created in the current scope or level of hierarchy and all power domains in the current scope can use the created port. By default, the direction of the port is in (or input port).

For example, to create the supply ports VDD1, VDD2, and VDD3, and GND at the top level of design hierarchy or power domain PD_TOP, use the command as follows:

```
create_supply_port VDD1  
create_supply_port VDD2  
create_supply_port VDD3  
create_supply_port GND
```

Supply Nets

A supply net is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net connects supply ports. A supply net that spans more than one power domain is said to be “reused” in multiple domains. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port. In [Figure 2-9](#), each block has supply ports to allow supply nets to cross from the top level down into the block level.

To create a supply net, use the `create_supply_net` command.

The supply net is created in the same scope or logic hierarchy as the specified power domain. When you use the `-reuse` option, the specified supply net is not created, instead an existing supply net with the specified name is reused.

```
create_supply_net GND_NET -domain PD1  
create_supply_net GND_NET -domain PD2 -reuse
```

When a supply net is created, it is not considered a primary power supply or ground net. To make a specific power supply or ground net of a power domain, the primary supply or ground net, use the `set_domain_supply_net` command.

The `connect_supply_net` command connects the supply net to the specified supply ports or pins. The connection can be within the same level of hierarchy or to ports or pins down the hierarchy.

The following example shows the use of the `connect_supply_net` command to connect supply nets to various supply ports at different levels of hierarchy or power domains.

```
connect_supply_net VDD_SW -ports VDD  
connect_supply_net VDD_SW -ports I1/I2/VDD
```

Power Switch

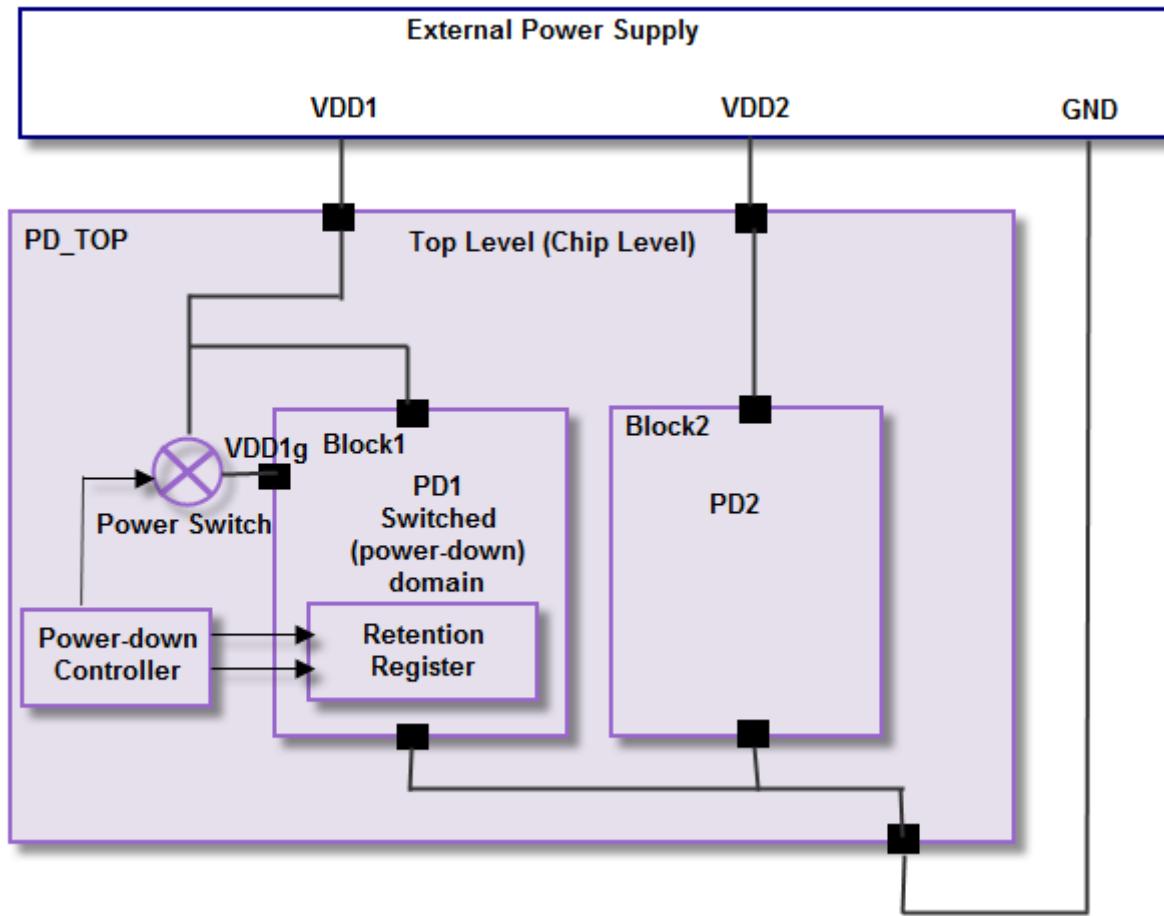
Power switch is a transistor or transistor array that can switch the power on and off for a portion of the chip, either at the VDD power supply (header switch) or at GND (footer switch), thereby cutting off power to portions of the chip during periods of inactivity.

Conceptually, it is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals.

In [Figure 2-10](#), PD1 is a power domain that has two supplies. One is a switchable supply called `VDD1g`. The other supply is an always-on supply from `VDD1`. The always-on power supply maintains the retention registers of the domain while `VDD1g` is powered down.

The power switch shuts off and turns on the power net `VDD1g`, either by connecting or disconnecting `VDD1` and `VDD1g`. A power controller logic block at the top level generates the control signal for the switch. It also generates the save and restore signals for the retention registers in domain PD1.

Figure 2-10 Power Switch



The `create_power_switch` command creates a virtual instance of power switch in the scope of the specified power domain. This command lets the tool know that a generic power switch resides in the design at a specific scope or level of hierarchy. The off state of the power switch output is used in the power state table.

Supply Sets

A supply set is an abstract collection of supply nets, that consists of two supply functions, power and ground. A supply set is domain-independent, which means that the power and ground in the supply

set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use the supply sets to define power intent at the RTL level. This allows you to synthesize a design even before you know the names of the actual supply nets.

You can use the `create_supply_set` command to create a supply set.

Supply Set Handles

A supply set handle is an abstract supply set implicitly created for a power domain. A newly created power domain has supply set handles for the primary supply set of a domain, a default isolation supply set, and a default retention supply set.

These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can physically implemented, its supply set handles must be refined, or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

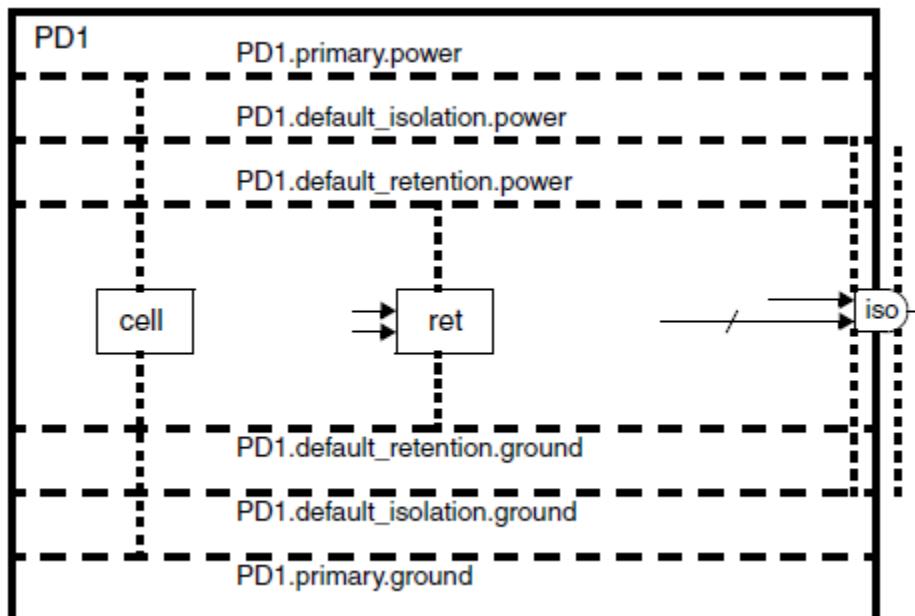
When you create new power domain with the `create_power_domain` command, the following supply set handles are implicitly created:

- `primary` – The primary supply set of the power domain.

- `default_isolation` – The default isolation supply set for any isolation strategy applied to this power domain, which is used when the isolation power or ground is not otherwise specified in the strategy.
- `default_retention` – The default retention supply set for any retention strategy applied to this power domain, which is used when the retention power or ground is not otherwise specified in the strategy.

For example, when you use the `create_power_domain` command to create a power domain called `PD1`, the command implicitly creates the supply set handles `PD1.primary`, `PD1.default_isolation`, and `PD1.default_retention`, providing the implicit power supply nets shown in [Figure 2-11](#).

Figure 2-11 Implicit Supply Set Handles



Power State Tables

A power state table (PST) lists the allowed combinations of voltage values and states of the power switches for all power domains in the design. It defines the legal combination of states that can exist simultaneously during the operation of the design.

A power state table is a set of power states of a design in which each power state is represented as an assignment of power states to individual power nets.

The `create_pst` command creates a new power state table and assigns a name to the table. The command lists the supply ports or supply nets in a particular order. The `add_port_state` command defines the names of the possible states for each supply port. The `add_pst_state` command lists the allowed combinations of states in the design.

Figure 2-12 Power State Table

	VDD	VDD_SW_S	VDD_SW
s0	HV	LV	LV
s1	HV	OFF	LV

3

Using UPF Commands to Specify Power Intent

This chapter describes the UPF commands supported by VCS NLP. These commands are used to specify the power intent for low power designs. This chapter contains the following sections:

- [UPF Commands Supported by VCS NLP](#)
- [Using add_power_state Command](#)
- [Using associate_supply_set Command](#)
- [Using add_state_transition Command](#)
- [Using bind_checker Command](#)
- [Using create_power_switch Command](#)
- [Using create_supply_net Command](#)

- Using map_isolation_cell Command
- Using query_cell_mapped Command
- Querying Power Domain Information of an Instance
- Using set_design_top Command
- Using set_equivalent Command
- Using set_isolation Command
- Using set_repeater Command
- Using set_related_supply_net Command
- Using set_port_attributes Command
- Defining Retention Strategies
- Including Current Scope in the Power Domain
- Supply Equivalence Checks by VCS NLP
- Using upf_version Command
- Using upf_tool Variable
- Refining Power Domain Elements
- Support for Negative Voltages
- Using synopsys_program_name Variable
- Handling Conflicting Port State Names in the add_port_state Command
- Setting Supply Port Default Value
- Driving Supply Expression Using Logic Expression

- Support for Auto-Completion of UPF Commands
- Support for Feedthrough Paths Inside Macro Blocks
- Generic Handles Support
- Bias Support for Low Power Designs
- Controlling Assertions
- Querying Power Domains to Find Always-On and Collapsible Power Domains
- Querying Supply Information of a Cell Instance
- Searching the Logic Hierarchy for the Supply Ports
- Supply Equivalence Checks by VCS NLP
- Interpretation of '/' in the UPF Commands
- UPF Structural Components
- UPF Script Examples

UPF Commands Supported by VCS NLP

The following sections describe the IEEE 1801 UPF commands supported by VCS NLP. These descriptions are intended to provide an overview of how the commands work in VCS NLP. For more information about using the UPF commands, see the IEEE 1801 (UPF) specification.

- [Basic Power Network Commands](#)
- [Level Shifter Commands](#)
- [Isolation Commands](#)
- [Retention Commands](#)
- [Power State Table Commands](#)
- [Logic Editing Commands](#)
- [Query Commands](#)
- [Simulation/Verification Extension](#)
- [Utility Commands](#)
- [Extending Switch](#)
- [Merging Power Domains](#)
- [Other Supported Power Intent Commands](#)

Basic Power Network Commands

The basic power commands define the power domains of the design and the supply ports, supply nets, and power switches of each domain. Following are the basic power network commands supported by VCS NLP:

- `associate_supply_set`
- `create_power_domain`
- `create_power_switch`
- `create_supply_port`
- `create_supply_net`
- `connect_supply_net`
- `create_supply_set`
- `map_power_switch`
- `set_domain_supply_net`

`associate_supply_set`

The `associate_supply_set` command associates a supply set handle to another supply set. The `supply_set_name` specifies the name of the supply set that must be associated with the handle specified with the `-handle` option.

Syntax

```
associate_supply_set supply_set_name  
[ -handle supply_set_handle ]
```

For more information on the `associate_supply_set` command, see “[Using associate_supply_set Command](#)” .

`create_power_domain`

The `create_power_domain` command defines a power supply distribution network at the current scope (hierarchical level) or at the scope of a specified hierarchical instance. A power domain must have one primary supply net and one primary ground net, and can optionally have additional supply nets, supply ports, and power switches.

Syntax

```
create_power_domain domain_name
    [-elements element_list]
    [-include_scope]
    [-available_supplies supply_set_sef_list]
    [-supply {supply_set_handle [supply_set_ref]}*]
    [-scope instance_name]
    [-update]
    [-define_func_type {supply_function {pg_type_list}}]*1
```

* Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

`create_power_switch`

The `create_power_switch` command creates an instance of a power switch in the scope of a power domain. The switch has at least one input supply port and one output supply port.

Syntax

```
create_power_switch switch_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}} ]*
[-supply_set supply_set_name]
[-on_partial_state {state_name input_supply_port
{boolean_function}} ]*
[-ack_port {port_name net_name [{boolean_function}]} ]*
[-ack_delay {port_name delay}]*
[-error_state {state_name {boolean_function}} ]*
[-domain domain_name]

```

* Indicates that the item in curly braces can be repeated in the command.

For more information, see “[Using create_power_switch Command](#)” .

create_supply_port

The `create_supply_port` command creates a supply port at the current scope or at the scope of a power domain specified with the `-domain` option. A supply port is a power supply connection point between the current scope and the next higher-level (parent) scope. A supply port allows a supply net to cross from one hierarchical level to another.

Syntax

```

create_supply_port port_name
  [-domain domain_name]
  [-direction <in | out | inout>]

```

create_supply_net

The `create_supply_net` command creates a supply net to supply power or ground to a power domain.

Syntax

```
create_supply_net net_name  
    [-domain domain_name] [-reuse]  
    [-resolve <unresolved | one_hot | parallel  
    | resolution_function_name>]
```

For more information on the `resolution_function_name` argument, see “[Specifying User-Defined Supply Net Resolution As a Verilog Function or Task in the Package](#)”.

For more information on the `create_supply_net` command, see “[Using `create_supply_net` Command](#)”.

connect_supply_net

The `connect_supply_net` command specifies an explicit connection of a supply net to a list of supply ports, thereby overriding any implicit connections that might otherwise apply. The command specifies the name of an existing supply net and lists the supply ports to be connected to the supply net.

Syntax

```
connect_supply_net net_name  
    [-ports list]  
    [-pg_type {pg_type_list element_list}]1  
    [-vct vct_name]  
    [-pins list]1  
    [-cells list]1  
    [-domain domain_name]1  
    [-rail_connection rail_type]1
```

- * Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

create_supply_set

The `create_supply_set` command creates a supply set at the current level of logic hierarchy. A supply set is a collection of supply nets. A supply set consists of the following functions:

- Power
- Ground
- pwell
- nwell

Syntax

```
create_supply_set set_name
    [-function {func_name [net_name]}] *
    [-update]
```

- * Indicates that the item in curly braces can be repeated in the command.

map_power_switch

The `map_power_switch` command can be used to explicitly specify which power switch cell must be used to implement a specified switch instance. The command must specify the name of an existing switch previously created with the

`create_power_switch` command, the name of the power domain containing the switch, and the list of library cells that can be used to implement the switch.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
map_power_switch {switch_name}
    -domain domain_name
    -lib_cells {list}
    [-port_map {{mapped_model_port
        switch_port_or_supply_net_ref}*}]
```

* Indicates that the item in curly braces can be repeated in the command.

set_domain_supply_net

The `set_domain_supply_net` command specifies the primary power net and primary ground net for an existing power domain. Every power domain must have these supply nets defined. They are the default power nets connected to the logic elements (or inferred cells) of the power domain.

Syntax

```
set_domain_supply_net domain_name
    -primary_power_net supply_net_name
    -primary_ground_net supply_net_name
```

Level Shifter Commands

The level shifter commands let you specify the strategy for inserting level shifters between power domains operating at different voltages. Following are the level shifter commands supported by VCS NLP:

- [set_level_shifter](#)
- [map_level_shifter_cell](#)
- [name_format](#)

set_level_shifter

The `set_level_shifter` command allows you to set a strategy for inserting level shifters during implementation. If a level shifter strategy is not specified on a particular power domain, the default level shifter strategy applies to all elements in the power domain, using the default strategy settings.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
set_level_shifter level_shifter_name
    -domain domain_name
    [-elements element_list]
    [-exclude_elements exclude_list]
    [-source <source_domain_name | source_supply_ref>]
    [-sink <sink_domain_name | sink_supply_ref>]
    [-applies_to <inputs | outputs | both>]
    [-rule <low_to_high | high_to_low | both>]
    [-threshold value | list]
    [-no_shift]
    [-force_shift]
    [-location <self | parent | automatic>]
```

```
[-input_supply_set supply_set_name]
[-output_supply_set supply_set_name]
[-internal_supply_set supply_set_name]
[-name_prefix string]
[-name_suffix string]
[-instance {{instance_name port_name}*} ]
[-transitive <TRUE | FALSE>]
[-update]
```

The `set_level_shifter` command supports path-based filtering with the `-source` and `-sink` options. Here, `-source` and `-sink` should be the name of a supply set or power domain. When a domain name is used, it represents the primary supply of that domain.

map_level_shifter_cell

The `map_level_shifter_cell` command maps a particular level-shifter strategy to a library cell or range of library cells. All level-shifter cells belonging to the specified strategy are mapped to one of the library cells specified in the `-lib_cells` list.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
map_level_shifter_cell level_shifter_strategy
  -domain domain_name
  -lib_cells list
  [-elements element_list]
```

name_format

The `name_format` command defines the format for constructing names of implicitly created objects.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
name_format
  [-isolation_prefix string]
  [-isolation_suffix string]
  [-level_shift_prefix string]
  [-level_shift_suffix string]
  [-implicit_supply_suffix string]
  [-implicit_logic_prefix string]
  [-implicit_logic_suffix string]
```

Note:

According to the UPF, the isolated net/port must be suffixed with `_UPF_ISO` by default. In case of back-to-back isolation on the same net/port, two isolation cells are inserted, and hence the suffixes are `_UPF_ISO_0` and `_UPF_ISO`.

Isolation Commands

The isolation commands specify the strategy for inserting isolation cells at the outputs of switched (power-down) domains. Following are the isolation commands supported by VCS NLP:

- [set_isolation](#)
- [set_isolation_control](#)
- [map_isolation_cell](#)

set_isolation

The `set_isolation` command specifies the isolation strategy for a power domain and the elements in that domain on which the strategy is applied. An isolation strategy includes specification of the enable signal net, the clamp value, and the location (inputs, outputs, or both).

Syntax

```
set_isolation isolation_name
    -domain ref_domain_name
    [-elements element_list]
    [-exclude_elements exclude_list]
    -source source_supply_ref
    -sink sink_supply_ref
    [-diff_supply_only <TRUE | FALSE>]
    [-applies_to <inputs | outputs | both>]
    [-applies_to_clamp <0 | 1 | any | Z | latch | value>]1
    [-applies_to_sink_off_clamp <0 | 1 | any | Z | latch | value>]1
    [-applies_to_source_off_clamp <0 | 1 | any | Z | latch | value>]1
    [-isolation_power_net net_name]
    [-isolation_ground_net net_name]
    [-no_isolation]
    [-force_isolation]1
    [-location <automatic | self | fanout | fanin1 | faninout1 | parent | sibling1>]
    [-clamp_value {<0 | 1 | Z | latch>*}]
    [-isolation_signal signal_list <>]
    [-isolation_sense <high | low | {<high | low>*}>]]
    [-isolation_supply_set supply_set_list]
    [-isolation_sense {<high | low>*}]
    [-name_prefix string]1
    [-name_suffix string]1
    [-sink_off_clamp {<0 | 1 | any | Z | latch | value> [simstate_list]}]1
    [-source_off_clamp {<0 | 1 | any | Z | latch | value>}
```

```
[simstate_list}]}1  
[-instance {{instance_name port_name}*}]*1  
[-transitive <TRUE | FALSE>]1  
[-update]
```

* Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

For more information on the `set_isolation` command, see “[Using set_isolation Command](#)” .

set_isolation_control

The `set_isolation_control` command allows the specification of the isolation control signal and the logical sense of that signal. The command identifies an existing isolation strategy and specifies the isolation control signal for that strategy.

Syntax

```
set_isolation_control isolation_name  
-domain domain_name  
-isolation_signal signal_name  
[-isolation_sense <high | low>]  
[-location <self | parent| sibling1 | fanout| automatic1>]
```

¹This command argument is parsed and ignored by VCS NLP.

map_isolation_cell

The `map_isolation_cell` command maps a particular isolation strategy to a library cell or range of library cells. All isolation cells belonging to the specified strategy are mapped or remapped to one of the library cells specified in the `-lib_cells` list.

Syntax

```
map_isolation_cell isolation_name
    -domain domain_name
    [-elements element_list]1
    [-lib_cells lib_cells_list]1
    [-lib_cell_type lib_cell_type]
    [-lib_model_name model_name {-port {port_name
    net_name}}]*]
```

* Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

For more information on the `map_isolation_cell` command, see [“Using map_isolation_cell Command”](#).

Retention Commands

The retention commands specify the strategy for inserting retention cells inside switched (power-down) domains. Following are the retention commands supported by VCS NLP:

- [set_retention](#)
- [set_retention_control](#)
- [set_retention_elements](#)
- [set_retention](#)

set_retention

The `set_retention` command specifies which registers in the domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality. Only the registers implied in the elements are given retention capabilities.

Syntax

```
set_retention retention_name
    -domain domain_name
    [-elements element_list]
    [-retention_power_net net_name]
    [-retention_ground_net net_name]
    [-retention_supply_set ret_supply_set]
    [-no_retention]
    [-save_signal {logic_net <high | low | posedge | negedge>}
     -restore_signal {logic_net <high | low | posedge |
     negedge>}]
    [-save_condition {{boolean_function}}]
    [-restore_condition {{boolean_function}}]
    [-retention_condition {{boolean_function}}]
    [-use_retention_as_primary]
    [-parameters {<RET_SUP_COR | NO_RET_SUP_COR | SAV_RES_COR
     | NO_SAV_RES_COR> *} ]
    [-instance {{instance_name [signal_name]} *} ]1
    [-transitive <TRUE | FALSE>]
    [-update]
```

*Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

For more information on the `set_retention` command, see “[Using set_retention command](#)”.

Note:

This command supports all UPF LRM options except the options related to supply set.

set_retention_control

The `set_retention_control` command allows the specification of the retention control signal and the logical sense of that signal. The command identifies an existing retention strategy and specifies the save and restore signals and logical senses of those signals for that strategy.

Note:

The `set_retention_control` command is deprecated as per UPF 2.0 LRM (IEEE Std 1801-2013).

Syntax

```
set_retention_control retention_name
  -domain domain_name
  -save_signal {{net_name <high | low | posedge | negedge>}}
  -restore_signal {{net_name <high | low | posedge | negedge>}}
  [-assert_r_mutex {{net_name <high | low | posedge | negedge>}}]* 
  [-assert_s_mutex {{net_name <high | low | posedge | negedge>}}]* 
  [-assert_rs_mutex {{net_name <high | low | posedge | negedge>}}]*
```

* Indicates that the item in curly braces can be repeated in the command.

Note:

The `-assert_r_mutex`, `-assert_rs_mutex`, and `-assert_s_mutex` arguments are not supported in VCS.

For more information on the `set_retention_control` command, see “[State Retention Support](#)”.

set_retention_elements

The `set_retention_elements` command defines a list of state elements whose collective state shall be maintained coherently if retention is applied to any of these elements in the list.

Syntax

```
set_retention_elements retention_list_name  
[-elements element_list]
```

For more information on the `-elements` argument, see “[Creating Retention List](#)”.

map_retention_cell

The `map_retention_cell` command provides a mechanism for constraining the implementation choices for retention registers. The command must specify the name of an existing retention strategy and power domain.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
map_retention_cell retention_name_list  
-domain domain_name  
[-elements element_list]  
[-exclude_elements exclude_list]  
[-lib_cells lib_cell_list]  
[-lib_cell_type lib_cell_type]  
[-lib_model_name name -port_map {{port_name net_ref} *}]
```

- * Indicates that the item in curly braces can be repeated in the command.
-

Power State Table Commands

A Power State Table (PST) defines the legal combinations of states that can exist at the same time during operation of a design. The state of a power domain consists of the set of on-off states and voltage levels of its supply ports. The power state table is used for synthesis, analysis, and optimization. Following are the PST commands supported by VCS NLP:

- `create_pst`
- `add_port_state`
- `add_power_state`
- `create_power_state_group`
- `add_pst_state`

`create_pst`

The `create_pst` command provides a mechanism for constraining the implementation choices for retention registers. The command must specify the name of an existing retention strategy and power domain.

Syntax

```
create_pst table_name
           -supplies supply_list
```

add_port_state

The `add_port_state` command adds state information to a supply port. The command specifies the name of the supply port and the possible states of the port. Each state is specified as a state name and the voltage level for that state.

Syntax

```
add_port_state port_name  
  {-state {name nom | min nom max | off}}*
```

* Indicates that the item in curly braces can be repeated in the command.

Note:

You cannot use `add_port_state` on an RTL port tied to a supply net in the UPF. You can only use this command on the supply ports explicitly created in the UPF using `create_supply_port`.

add_power_state

The `add_power_state` command sets the power states on the nets of a supply set so that they can be used in the power state table.

Syntax

```
add_power_state object_name  
  [-supply | -domain | -group]  
  [-state {state_name  
    [-logic_expr {boolean_expression}]  
    [-supply_expr {boolean_expression}]  
    [-power_expr {power_expression}]  
    [-simstate simstate]  
    [-legal | -illegal]}]*  
  [-update]  
  [-simstate simstate]
```

* Indicates that the item in curly braces can be repeated in the command.

For more information on the `add_power_state` command, see “[Using add_power_state Command](#)” .

create_power_state_group

The `create_power_state_group` command defines a group name that can be used in the `add_power_state` command. The group `group_name` is defined in the current scope.

Syntax

```
create_power_state_group group_name
```

For more information on the `create_power_state_group` command, see “[Creating Power State Group Using create_power_state_group Command](#)” .

add_pst_state

The `add_pst_state` command defines the states of each of the supply nets for one possible state of the design.

Syntax

```
add_pst_state state_name
    -pst_table_name
    -state supply_states
```

Logic Editing Commands

The logic editing commands create logic nets and logic ports and make connections between these nets and ports, irrespective of the power network. Following are the logic editing commands supported by VCS NLP:

- `create_logic_net`
- `create_logic_port`
- `connect_logic_net`

create_logic_net

The `create_logic_net` command creates a new logic net in the active scope.

Syntax

```
create_logic_net net_name
```

create_logic_port

The `create_logic_port` command creates a new logic port in the active scope and specifies the port direction (`in` for input, `out` for output, or `inout` for bidirectional).

Syntax

```
create_logic_port port_name  
[-direction <in | out | inout1>]
```

¹This command argument is parsed and ignored by VCS NLP.

connect_logic_net

The `connect_logic_net` command connects a specified logic net to one or more specified logic ports.

Syntax

```
connect_logic_net net_name  
    -ports port_list  
    [-reconnect]
```

Query Commands

Each query commands returns a list of objects in the design that match the criteria you specify in the command.

Note:

Query commands are not supported in the UPF. You can specify query commands in a configuration file and pass it to the `vcs` command line using the `-lpa_bind <filename>` option.

Following are the query commands supported by VCS NLP:

- [find_objects](#)
- [query_cell_instances](#)
- [query_cell_mapped](#)
- [query_power_domain_element](#)
- [query_net_ports](#)
- [query_port_net](#)
- [query_pst](#)

- `query_pst_state`
- `query_power_switch`
- `query_upf`
- `query_design_attributes`
- `query_hdl2upf_vct`
- `query_isolation`
- `query_power_domain`
- `query_pg_info_cell`
- `query_retention`
- `query_retention_control`
- `query_supply_net`

find_objects

The `find_objects` command finds instances, nets, or ports defined in the logic hierarchy in a specified scope and returns a list of object names that match a given search pattern.

Syntax

```
find_objects scope
  -pattern search_pattern
    [-object_type <model | inst | port | supply_port | net
     | process1>]
    [-direction <in | out | inout>]
    [-transitive [<TRUE | FALSE>]]
    [-regexp | -exact]
    [-ignore_case]
    [-non_leaf | -leaf_only]
```

¹This command argument is parsed and ignored by VCS NLP.

Note:

- Search is case-sensitive for Verilog objects and case-insensitive for VHDL objects.
- Hierarchical separators are not supported in the search pattern with `-pattern`. Search is limited to a non-hierarchical name in the given scope of search.

For more information on the `find_objects` command, see [“Searching the Logic Hierarchy for the Supply Ports”](#).

query_cell_instances

The `query_cell_instances` command returns a list of instance names for all instances of a given reference cell in the current scope of the design. You can optionally restrict the query to a given power domain.

Syntax

```
query_cell_instances cell_name  
[-domain domain_name]
```

query_cell_mapped

The `query_cell_mapped` command returns the reference cell name of a given cell instance.

Syntax

```
query_cell_mapped instance_name
```

For more information on the `query_cell_mapped` command, see [“Using query_cell_mapped Command”](#).

query_power_domain_element

The `query_power_domain_element` command returns the hierarchical power domain name corresponding to the given instance.

Syntax

```
query_power_domain_element design_element
```

For more information on the `query_power_domain_element` command, see “[Querying Power Domain Information of an Instance](#)”

query_net_ports

The `query_net_ports` command returns a list of the ports logically connected to a specified net.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
query_net_ports net_name
  [-transitive <TRUE | FALSE>]
  [-leaf]
```

query_port_net

The `query_port_net` command returns the name of the net logically connected to a specified port, if any such net exists.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
query_port_net port_name  
[-conn low | high]
```

query_pst

The `query_pst` command returns information about previously defined power state tables in the active scope. The command `query_pst *` returns a list of the power state table names.

Syntax

```
query_pst table_name  
[-detailed]
```

query_pst_state

The `query_pst_state` command returns information about the states that have been previously defined for a specified power state table.

Syntax

```
query_pst_state state_name  
-pst table_name  
[-detailed]
```

query_power_switch

The `query_power_switch` command returns information about previously defined power switches. The command `query_power_switch *` returns a list of the power switch names.

Syntax

```
query_power_switch switch_name  
[-detailed]
```

query_upf

The `query_upf` command searches for instances, nets, supply nets, ports, and supply ports in and below the scope or within the extent of a `domain_name`. This command works on the logic hierarchy and can be executed post-UPF annotation.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
query_upf <domain_name | scope>
  -pattern search_pattern
  [-object_type <inst | port | supply_port | net | supply_net
  | supply_set>]
  [-inst_type <level_shifter | isolation_cell | switch_cell
  | retention_cell | all>] |
  [-direction <in | out | inout>]
  [-transitive [<TRUE | FALSE>]]
  [-regexp | -exact]
  [-ignore_case]
  [-non_leaf | -leaf_only]
```

query_design_attributes

The `query_design_attributes` command queries attribute information for a specified `element_name` or `model_name`.

Syntax

```
query_design_attributes
  <-element element_name | -model model_name>
  [-detailed]
```

query_hdl2upf_vct

The `query_hdl2upf_vct` command can list and query any previously defined value conversion table (VCT).

Syntax

```
query_hdl2upf_vct vct_name  
      [-detailed]
```

query_isolation

The `query_isolation` command can list the previously defined isolation strategies for the specified power domain `domain_name`. All elements returned are referenced to the current scope.

Syntax

```
query_isolation isolation_name  
      -domain ref_domain_name  
      [-detailed]
```

query_power_domain

The `query_power_domain` command queries the parameters of a power domain.

Syntax

```
query_power_domain domain_name  
      [-detailed]
```

query_pg_info_cell

The `query_pg_info_cell` command allows you to query the supply information of a hierarchical cell instance.

Syntax

```
query_pg_info_cell -instance <instance_hier_path>
```

For more information on the `query_pg_info_cell` command, see “[Querying Supply Information of a Cell Instance](#)” section.

query_retention

The `query_retention` command lists the previously defined retention strategies for the specified power domain `domain_name`. All elements returned are referenced to the current scope.

Syntax

```
query_retention retention_name
    -domain domain_name
    [-detailed]
```

query_retention_control

The `query_retention_control` command queries the retention control information for a retention strategy.

Syntax

```
query_retention_control retention_name
    -domain domain_name
    [-detailed]
```

query_supply_net

The `query_supply_net` command returns the information about a previously created supply net.

Syntax

```
query_supply_net net_name  
  [-domain domain_name]  
  [-detailed]
```

Simulation/Verification Extension

The simulation and verification extension commands support low-power checking by simulation and verification tools. Following are the simulation and verification extension commands supported by VCS NLP:

- [set_design_top](#)
- [set_partial_on_translation](#)

set_design_top

The `set_design_top` command specifies the root of the design.

Syntax

```
set_design_top root
```

For more information on the `set_design_top` command, see [“Using set_design_top Command”](#).

set_partial_on_translation

The `set_partial_on_translation` command specifies the translation of the PARTIAL_ON state to either FULL_ON or OFF for purposes of evaluating the power state of supply sets and power domains.

Syntax

```
set_partial_on_translation  
[FULL_ON | OFF]
```

Utility Commands

These are the commands to load and execute UPF commands from a file, to write a set of UPF commands to a file, and to set the hierarchical scope for subsequent UPF commands. Following are the utility commands supported by VCS NLP:

- [load_upf](#)
- [set_scope](#)
- [set_design_attributes](#)
- [set_port_attributes](#)

load_upf

The `load_upf` command executes the UPF commands in a specified file.

Syntax

```
load_upf  
    load_upf upf_file_name  
    [-scope instance_name]
```

set_scope

The `set_scope` command specifies the hierarchical scope of subsequent commands, including UPF commands.

Syntax

```
set_scope [instance]
```

set_design_attributes

The `set_design_attributes` command sets the attributes of one or more cells.

Syntax

```
set_design_attributes
  [-models model_list | -elements element_list]
  {-attribute name value}*  
* Indicates that the item in curly braces can be repeated in the command.
```

The `-models` option specifies a list of design models on which the attributes are set. The `-elements` option specifies a collection of cells or supply sets on which the attributes are set. In the absence of the `-models` or `-elements` option, the attribute is set on the current top-level design.

The `-attribute` option specifies the name and value of the attributes to be set on the cells specified with the `-elements` option.

set_port_attributes

The `set_port_attributes` command specifies a collection of ports where the attributes must be set for the source or sink, for the interface of the power domains, when used with the `set_isolation` command.

Syntax

```
set_port_attributes
  [-model name]
  [-elements element_list]
  [-exclude_elements element_exclude_list]1
  [-ports port_list]1
  [-exclude_ports port_exclude_list]1
  [-applies_to <inputs | outputs | both>]
  [-attribute {name value}]*
  [-clamp_value <0 | 1 | any | Z | latch | value1>]
  [-sink_off_clamp <0 | 1 | any | Z | latch | value>]1
  [-source_off_clamp <0 | 1 | any | Z | latch | value>]1
  [-driver_supply supply_set_ref]
  [-receiver_supply supply_set_ref]
  [-pg_type pg_type_value]
  [-related_power_port supply_port_name]
  [-related_ground_port supply_port_name]
  [-related_bias_ports supply_port_name_list]
  [-feedthrough]
  [-unconnected]
```

* Indicates that the item in square braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

For more information on the `set_port_attributes` command, see “[Using set_port_attributes Command](#)” .

Extending Switch

The following UPF command allows you to extend a switch:

set_power_switch

The `set_power_switch` command extends a switch by adding the input supply port(s), output supply port(s), and states to the switch. The supply ports are connected to the specified nets.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
set_power_switch switch_name
    {-output_supply_port {port_name [supply_net_name]}}
    {-input_supply_port {port_name [supply_net_name]} } }*
    {-control_port {port_name}}}
    {-on_state {state_name input_supply_port
    {boolean_function}} }*
    [-supply_set supply_set_name]
    [-on_partial_state {state_name input_supply_port
    {boolean_function}} ]*
    [-off_state {state_name {boolean_function}} ]*
    [-error_state {state_name {boolean_function}} ]*
```

* Indicates that the item in curly braces can be repeated in the command

Merging Power Domains

The following UPF command allows you to merge power domains:

merge_power_domains

The `merge_power_domains` command merges two or more existing power domains into a single, new power domain. The merged domains cannot be referenced separately.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
merge_power_domains new_domain_name
    -power_domains list
    [-scope instance_name]
    [-all_equivalent]
```

Other Supported Power Intent Commands

Following are the other UPF commands supported by VCS NLP to specify power intent:

- bind_checker
- create_composite_domain
- create_hdl2upf_vct
- create_upf2hdl_vct
- describe_state_transition
- add_state_transition
- load_simstate_behavior
- load_upf_protected
- set_design_top
- set_equivalent
- set_repeater
- set_simstate_behavior

- [upf_version](#)

bind_checker

The `bind_checker` command inserts checker modules into a design without modifying the design code or introducing functional changes. The mechanism for binding the checkers to design elements relies on the SystemVerilog `bind` directive. The `bind` directive causes one module to be instantiated within another, without having to explicitly alter the code of either. This facilitates the complete separation between the design implementation and any associated verification code.

Syntax

```
bind_checker instance_name
  -module checker_name
  [-elements element_list]
  [-ports {{port_name net_name}*}]
  [-parameters {{param_name param_value}*}]
```

* Indicates that the item in curly braces can be repeated in the command.

For more information on the `bind_checker` command, see “[Using bind_checker Command](#)” .

create_composite_domain

The `create_composite_domain` command defines a composite power domain composed of one or more sub-domains. A composite power domain is a simple container for a set of power domains. Unlike a power domain, a composite domain has no corresponding physical region on the silicon.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
create_composite_domain composite_domain_name
    [-subdomains subdomain_list]
    [-supply {supply_set_handle [supply_set_ref]}]*
    [-update]
```

* Indicates that the item in curly braces can be repeated in the command

create_hdl2upf_vct

The `create_hdl2upf_vct` command defines a VCT from an HDL logic type to the state type of the supply net value when that value is propagated from HDL port to a UPF supply net.

Syntax

```
create_hdl2upf_vct vct_name
    -hdl_type {<vhdl | sv> [typename]}
    -table {{from_value to_value}*}
```

create_upf2hdl_vct

The `create_upf2hdl_vct` command defines a VCT that can be used to convert the UPF `supply_net_type.state` values into the HDL logic values when the values are propagated from a UPF supply net to a logic port defined in HDL model.

Syntax

```
create_upf2hdl_vct vct_name
    -hdl_type {<vhdl | sv> [typename]}
    -table {{from_value to_value}*}
```

- * Asterisk indicates that the item in curly braces can be repeated in the command.

describe_state_transition

The `describe_state_transition` command specifies the legality of a transition from one named power state of an object to another. The occurrence of an unnamed state during the transition from one state to another is ignored. `-from` and `-to` specify many-to-many transitions. The `-paired` argument specifies one or more one-to-one transitions.

Syntax

```
describe_state_transition transition_name
-object object_name
[-from from_list -to to_list]
[-paired {{from_state to_state}*}]
[-through through_list]
[-legal | -illegal]
```

- * Indicates that the item in curly braces can be repeated in the command

For more information, see “[Constraining Low Power Coverage Using the `describe_state_transition` Command](#)” .

add_state_transition

The `add_state_transition` command defines the named transitions between power states of an object and used to specify the legality of the transitions.

Syntax

```
add_state_transition
[-supply | -domain] object_name
```

```
[ -transition {transition_name  
[-from from_list -to to_list]  
[-paired {{from_state to_state}*}}]  
[-legal | -illegal]  
[-through through_list]  
} ]*
```

* Indicates that the item in curly braces can be repeated in the command

For more information, see [“Using add_state_transition Command”](#).

load_simstate_behavior

The `load_simstate_behavior` command loads a UPF file that only contains `set_simstate_behavior` commands and applies these to the models in the library `lib_name`.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
load_simstate_behavior lib_name  
-file file_list
```

load_upf_protected

The `load_upf_protected` command loads a UPF file in a protected environment that prevents corruption of existing variables.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
load_upf_protected upf_file_name  
[-hide_globals]
```

```
[-scope scope_name]
[-version upf_version]
[-params param_list]
```

set_design_top

The `set_design_top` command specifies the root of the design.

Syntax

```
set_design_top root
```

set_equivalent

The `set_equivalent` command declares that the specified supplies are electrically or functionally equivalent. This command returns an empty string on success and an error on failure.

Syntax

```
set_equivalent
[-function_only]
[-nets supply_net_name_list]
[-sets supply_set_name_list]]
```

For more information on the `set_equivalent` command, see [“Using set_equivalent Command”](#).

set_repeater

The `set_repeater` command defines a strategy for inserting repeater cells (buffers) for ports on the interface of a power domain. Repeater cells are placed within the domain, driven by input ports of the domain, and driving the output ports of the domain.

Following is the syntax of the `set_repeater` command:

```
set_repeater strategy_name
  -domain domain_name
  [-elements element_list]
  [-exclude_elements exclude_list]
  [-applies_to <inputs | outputs | both>]
  -repeater_supply supply_set_ref
  [-name_prefix string] [-name_suffix string]
  [-update]
```

For more information on the `set_repeater` command, see “[Using set_repeater Command](#)”.

set_simstate_behavior

The `set_simstate_behavior` command specifies the simstate behavior for models or instances. If `ENABLE` is specified, the simstate simulation semantics are applied for every supply set automatically connected to an instance of the model.

Syntax

```
set_simstate_behavior <ENABLE | DISABLE>
  [-lib name]1
  [-models list]
  [-elements element_list]
  [-exclude_elements exclude_list]
```

¹This command argument is parsed and ignored by VCS NLP.

upf_version

The `upf_version` command returns a string value representing the UPF version currently being used by the tool reading the UPF file.

Note:

This command is parsed and ignored by VCS NLP.

Syntax

```
upf_version [string]
```

For more information on the `upf_version` command, see “[Using upf_version Command](#)” .

Using add_power_state Command

The `add_power_state` command defines power states of a supply set. This section describes the simulation semantics of the `add_power_state` command.

The following sections describe the `add_power_state` support in VCS NLP:

- [Use Model](#)
- [Supply Expressions and Logic Expressions](#)
- [Power State Evaluation of Supply Set in Presence of -supply_expr/-logic_expr](#)
- [Defining Power State Tables Using the add_power_state Command](#)
- [Limitations of add_power_state Support](#)

Use Model

Following is the syntax of the `add_power_state` command:

```
add_power_state object_name  
[-supply | -domain | -group]  
[-state {state_name}
```

```

[-logic_expr {boolean_expression}]
[-supply_expr {boolean_expression}]
[-power_expr {power_expression}]
[-simstate simstate]
[-legal | -illegal}] }*
[-update]
[-simstate simstate]

```

- * Indicates that the item in curly braces can be repeated in the command.

Where, `object_name` is the name of a supply set. You can use the `add_power_state` command to specify:

- Single or multiple states of a supply set
- Combination of supply expression and logic expression to define the state
- The `simstate` (NORMAL, CORRUPT, CORRUPT_ON_ACTIVITY) for any power domain driven by a supply set

Note:

- You can use `-simstate` either inside or outside the `-state` option.
- You can use a single `add_power_state` command to define multiple states with the same `simstate`. For example,

```

add_power_state SS_PD3
  -state ON1 {-supply_expr {power == `{FULL_ON, 1.2}} } \
  -state ON2 {-supply_expr {power == `{FULL_ON, 1.1}} } \
  -simstate NORMAL

```

Supply Expressions and Logic Expressions

The supply expression (-supply_expr) contains the supply set function (power, ground, nwell, pwell). The expression should contain the state and voltage value or a range of voltage values of a supply set function. The expression can be combination of multiple functions of a supply set (see [Example 3-1](#)).

Example 3-1 Supply Expressions

```
add_power_state SS_TOP -state HV {-supply_expr {power ==  
`{FULL_ON, 1.08} } }
```

The logic expression (-logic_expr) is a regular SystemVerilog Boolean expression. You must use logic signals from the design to create logic expressions (see [Example 3-2](#)).

Example 3-2 Logic Expressions

```
add_power_state TOP -state HV {-logic_expr {pd_top} -  
simstate NORMAL }
```

A particular supply set state can contain a supply expression, logic expression, or both.

Power State Evaluation of Supply Set in Presence of -supply_expr/-logic_expr

In alignment with IEEE 1801-2013 (Section 9.3.2 Power state determination), the power state of a supply set is determined as follows:

1. If a power state has a supply expression, but no logic expression, then the power state is active if the supply expression is True

2. Else if a power state has a logic expression, but no supply expression, then the power state is active if the logic expression is True
3. Else if a power state has both a logic expression and a supply expression, then:
 - If the logic expression is True, the power state is active
 - If the supply expression is False, issue an error message

As indicated above, `-logic_expr` gets highest priority when both `-logic_expr` and `-supply_expr` are defined for a power state.

Defining Power State Tables Using the `add_power_state` Command

You can use the `add_power_state` command to perform the following:

- Define power state tables (PSTs) using the following two options:
`-supply` and `-group`
For more information, see “[Use Model](#)” and “[Defining PSTs Using add_power_state Options](#)” sections.
- Create power state group using the `create_power_state_group` command. For more information, see “[Creating Power State Group Using create_power_state_group Command](#)” section.
- Compose states hierarchically using the `-group` option. For more information, see “[Composing States Hierarchically](#)” section.

Use Model

You can use the `-supply` and `-group` options of the `add_power_state` command to define PSTs. Following is the syntax of the `add_power_state` command:

```
add_power_state [-supply | -group] object_name  
[-simstate simstate]  
{-state state_name { [-supply_expr {boolean_function}]  
[-logic_expr {boolean_function}]  
[-simstate simstate] }}
```

Where,

`-supply` — Allows you to specify the name of the supply set for which a state has to be created. This option is not mandatory. Even in the absence of this option, the given `object_name` is matched with an existing supply set.

When `-supply` is used:

- `object_name` must be the name of a supply set or a supply set handle
- The `-state` option can be used multiple times
- The following operators are supported for `-supply_expr`:
`==` and `&&` (this implies that both power and ground functions can be specified in the same Boolean expression)
- The `-supply_expr boolean_expression` references functions of supply sets or supply set handles

`-group` — Allows you to specify the name of the already created group (at the given scope). For more information on group, see “[Creating Power State Group Using `create_power_state_group` Command](#)”.

When `-group` is used:

- The `-state` option can be used multiple times
- The `-logic_expr boolean_expression` references supply set states, group states, or PST states. The Boolean expression can also be a mixture of these three types of states.
- Logic nets/ports/pins are not allowed in the Boolean expression. Specifying logic nets/ports/pins results in an error.
- The `-supply_expr` option is not supported (as per IEEE Std 1801-2009 LRM).
- The `-simstate` option (both within `-state` and outside `-state`) is not supported (as per IEEE Std 1801-2009 LRM).

Note:

You cannot use both `-supply` and `-group` options with the `add_power_state` statement. You can only use one of these options.

Creating Power State Group Using `create_power_state_group` Command

The `create_power_state_group` command defines a name for a group of related power states of the `add_power_state` command. Following is the syntax of the `create_power_state_group` command:

```
create_power_state_group group_name
```

Where, `group_name` is the simple name of the group that is defined in the current scope. The group name is used with the `-group` option of the `add_power_state` command.

Power states of a group may be defined in terms of power states of supply sets, power domains, composite domains, instances, and other groups. Power states of two or more different groups may refer to power states of the same object.

Defining PSTs Using `add_power_state` Options

To define PSTs using `add_power_state`, the following options are used:

- `-supply`
- `-group`

Consider that the following PST is to be built:

STATE\SUPPLY_SET	SS1	SS2
RUN	ON	ON
SLEEP	ON	OFF

Defining the PST is a three-step process:

1. Define the supply set states (ON and OFF listed in the above table)
2. Create a new group (also known as PST name)
3. Define the PST states (RUN and SLEEP listed in the above table)

Defining Supply Set States Using the add_power_state Command

Follow the guidelines listed below to define the supply set states:

1. Use `-supply` and specify the name of the supply set for which a state has to be created.
2. Use `-state` to specify the name of the supply set state.
3. Use `-supply_expr` to specify the Boolean expression for the supply set state.

The exact usage of `add_power_state` to define the supply set states is listed below:

```
# Creating power state ON for SS1
add_power_state -supply SS1 \
    -state ON {-supply_expr {(power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
```



```
# Creating power states ON and OFF for SS2
add_power_state -supply SS2 \
    -state ON {-supply_expr {(power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
```



```
add_power_state -supply SS2 \
    -state OFF {-supply_expr {(power == {OFF}) && (ground ==
{FULL_ON 0})}}
```

Creating a Group Using the create_power_state_group Command

Create a new group (PST name) using the `create_power_state_group` command. This command takes only one argument which is the group name. The group is created at the given scope.

```
create_power_state_group GROUP1
```

Defining PST States Using the add_power_state Command

Follow the guidelines listed below to define the PST states:

1. Use `-group` and specify the name of the already created group (at the given scope).
2. Use `-state` to specify the name of the PST state.
3. Use `-logic_expr` to specify the Boolean expression for the PST state.

The exact usage of `add_power_state` to define the PST states is listed below:

```
add_power_state -group GROUP1
    -state RUN {-logic_expr {SS1 == ON && SS2 == ON}}
    -state SLEEP {-logic_expr {SS1 == ON && SS2 == OFF}}
```

Composing States Hierarchically

This section lists an example on how states present in multiple (for example, scope-level) groups (PSTs) can be composed hierarchically to create a top-level group. The `-group` option is used to achieve this. This avoids the exponential explosion of PST size.

Consider the following two scope-level groups (PSTs):

Table 3-1 Group Defined at SCOPE1

STATE\SUPPLY_SET	SS1	SS2
RUN	ON	ON
SLEEP	ON	OFF

Table 3-2 Group Defined at SCOPE2

STATE\SUPPLY_SET	SS1	SS2
RUN	ON	ON
IDLE	ON	OFF

For example, if your top-level group has the following states:

- When SCOPE1 is in RUN, SCOPE2 will be in RUN
- When SCOPE1 is in SLEEP, SCOPE2 will be in RUN
- When SCOPE1 is in SLEEP, SCOPE2 will be in IDLE

With the -group option, this intent can be easily captured with the following command:

```
create_power_state_group TOP_SCOPE_GROUP
add_power_state -group TOP_SCOPE_GROUP
  -state RUN_RUN {-logic_expr {SCOPE1/GROUP1 == RUN &&
SCOPE2/GROUP1 == RUN}}
  -state SLEEP_RUN {-logic_expr {SCOPE1/GROUP1 == SLEEP &&
SCOPE2/GROUP1 == RUN}}
  -state SLEEP_IDLE {-logic_expr {SCOPE1/GROUP1 == SLEEP &&
SCOPE2/GROUP1 == IDLE}}
```

Examples

This section lists the examples that show the usages that are syntactically and semantically correct and the usages that result in an error.

Example-1

```
create_supply_set SS1
create_power_domain PD_TOP
create_power_state_group SS1
```

The `create_power_state_group` command results in an error as there is another UPF object (supply set) with the name SS1.

Example-2

```
set_scope /
create_supply_set SS1
create_supply_set SS2
add_power_state -supply SS1 \
    -state ON {-supply_expr {(power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
add_power_state -supply SS2 \
    -state ON {-supply_expr {(power == {FULL_ON 0.9}) &&
(ground == {FULL_ON 0})}}

create_power_state_group GROUP1
add_power_state -group GROUP1
    -state RUN {-logic_expr {SS1 == ON && SS2 == ON} }

set_scope MID_INST
create_supply_set SS1
create_supply_set SS2
add_power_state -supply SS1 \
    -state ON {-supply_expr {(power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
add_power_state -supply SS2 \
    -state ON {-supply_expr {(power == {FULL_ON 0.9}) &&
(ground == {FULL_ON 0})}}

add_power_state -group GROUP1
    -state RUN {-logic_expr {SS1 == ON && SS2 == ON} }
```

The last `add_power_state` command results in an error because this command is referring to a group by name GROUP1 that is not created at the current scope MID_INST.

Example-3

```
set_scope /
create_supply_set SS1
```

```

create_supply_set SS2
add_power_state -supply SS1 \
    -state ON {-supply_expr { (power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
add_power_state -supply SS2 \
    -state ON {-supply_expr { (power == {FULL_ON 0.9}) &&
(ground == {FULL_ON 0})}}

create_power_state_group GROUP1
add_power_state -supply SS1 -group GROUP1
    -state RUN {-logic_expr {SS1 == ON && SS2 == ON}}

```

The last `add_power_state` command results in an error because the options `-supply` and `-group` are used together. The syntax does not allow these two options to be used together.

Example-4

```

set_scope /
create_supply_set SS1
add_power_state -supply SS1 \
    -state {ON -supply_expr { (power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}

```

The `add_power_state` command has a curly brace before the state name. The curly brace should appear after the state name. This command results in an error because of the syntax error.

Example-5

```

set_scope /
create_supply_set SS1
add_power_state -supply SS1 \
    -state ON {-supply_expr { (power == {FULL_ON 0.8}) &&
(ground == {FULL_ON 0})}}
add_power_state -supply SS1 \
    -state ON {-logic_expr {port1 && port2}}

```

Here, all the `add_power_state` commands run successfully.

Interaction Between `create_pst/add_pst_state` and `add_power_state` Syntax

This section describes the interaction that is allowed between the old syntax (defining PSTs using the `create_pst` and `add_pst_state` commands) and new syntax (defining PSTs using `-supply` and `-group` options) of the `add_power_state` command for defining PSTs.

The interaction style involves using old syntax at lower level and new syntax at higher level. Two cases are possible here.

- Case-1: Create supply set states using the old syntax, and create PST using the new syntax
- Case-2: Create PST using the old syntax, and create top-level PST using the new syntax

Case-1: Create supply set states using old syntax, and create PST using new syntax

Following is an example to illustrate this case.

The supply set states are created using the old syntax of `add_power_state` as follows:

```
add_power_state SS1 -state P_ON {-supply_expr {power ==  
`{FULL_ON, 1.08}}}  
add_power_state SS1 -state G_ON {-supply_expr {ground ==  
`{FULL_ON, 0}}}  
  
add_power_state SS2 -state P_ON {-supply_expr {power ==  
`{FULL_ON, 0.08}}}  
add_power_state SS2 -state G_ON {-supply_expr {ground ==  
`{FULL_ON, 0}}}
```

```
add_power_state SS2 -state P_OFF {-supply_expr {power ==
`{OFF}}}}
```

For example, if you want to create the following PST using the above supply set states.

STATE\SUPPLY_SET	SS1.power	SS1.ground	SS2.power	SS2.ground
RUN	P_ON	G_ON	P_ON	G_ON
SLEEP	P_ON	G_ON	P_OFF	G_ON

This can be achieved using the `-group` option as follows:

```
create_power_state_group MY_GROUP
add_power_state -group MY_GROUP \
    -state RUN {-logic_expr {SS1 == P_ON && SS1 == G_ON && SS2 == P_ON && SS2 == G_ON}}
    -state SLEEP {-logic_expr {SS1 == P_ON && SS1 == G_ON && SS2 == P_OFF && SS2 == G_ON}}
```

Case-2: Create PST using old syntax, and create top-level PST using new syntax

Following is an example to illustrate this case.

Consider the following scope-level PSTs created using the old syntax:

Table 3-3 PST Defined at SCOPE1

STATE\SUPPLY_SET	SS1.power	SS1.ground	SS2.power	SS2.ground
RUN	P_ON	G_ON	P_ON	G_ON
SLEEP	P_ON	G_ON	P_OFF	G_ON

Table 3-4 PST Defined at SCOPE2

STATE\SUPPLY_SET	SS1.power	SS1.ground	SS2.power	SS2.ground
RUN	P_ON	G_ON	P_ON	G_ON
SLEEP	P_ON	G_ON	P_OFF	G_ON

The commands used to create the above PSTs (for example, PST defined at SCOPE1) are as follows:

```

add_power_state SS1 -state P_ON {-supply_expr {power ==
`{FULL_ON, 1.08}}}
add_power_state SS1 -state G_ON {-supply_expr {ground ==
`{FULL_ON, 0}}}

add_power_state SS2 -state P_ON {-supply_expr {power ==
`{FULL_ON, 0.08}}}
add_power_state SS2 -state G_ON {-supply_expr {ground ==
`{FULL_ON, 0}}}
add_power_state SS2 -state P_OFF {-supply_expr {power ==
`{OFF}}}

create_pst MY_PST -supplies {SS1.power SS1.ground SS2.power
SS2.ground}
add_pst_state RUN -pst MY_PST -state {P_ON G_ON P_ON G_ON}
add_pst_state SLEEP -pst MY_PST -state {P_ON G_ON P_OFF G_ON}

```

If you want to create the following top-level PST by combining the states of the two scope-level PSTs:

STATE\PST	SCOPE1/MY_PST	SCOPE2/MY_PST
RUN_RUN	RUN	RUN
SLEEP_RUN	SLEEP	RUN
SLEEP_SLEEP	SLEEP	SLEEP

This can be achieved using the `-group` option in the following way:

```

create_power_state_group TOP_SCOPE_GROUP
add_power_state -group TOP_SCOPE_GROUP

```

```
-state RUN_RUN {-logic_expr {SCOPE1/MY_PST == RUN &&
SCOPE2/MY_PST == RUN}}
-state SLEEP_RUN {-logic_expr {SCOPE1/MY_PST == SLEEP &&
SCOPE2/MY_PST == RUN}}
-state SLEEP_SLEEP {-logic_expr {SCOPE1/MY_PST == SLEEP &&
SCOPE2/MY_PST == SLEEP}}
```

Note:

The scope-level PSTs can be created either by using `add_power_state` or by using `add_port_state`. The main thing to note here is how scope-level PSTs can be used to create a top-level PST using the `-group` option.

Limitations

Following is the limitation of this feature:

- The `-legal/-illegal` options are ignored when specified with `-group`

Defining Power States for a Power Domain

VCS NLP allows you to define power states for a power domain using the `-domain` option of the `add_power_state` command.

The following sections describe this feature in detail:

- [Use Model](#)
- [Referring Power Domain States in Group Logic Expression](#)
- [Usage Examples](#)
- [Error Scenarios](#)

Use Model

Following is the use model of `add_power_state -domain`:

```
add_power_state
[-domain] object_name
[-update]
[-state {state_name [-logic_expr {boolean_expression}]}
[-illegal]}]*
```

To define power state of a power domain:

- The `object_name` must be an existing power domain name. Power domain name must be simple name.
- The `-logic_expr` option must be used to specify the power state definition. The objects allowed in the logic expression are supply sets, PSTs, groups, and other power domains.

Example:

```
create_supply_set SS1
create_supply_set SS2
create_power_domain PD1
add_power_state -domain PD1 -state ON {-logic_expr {SS1==ON1
&& SS2== ON2}}
```

Key Points to Note

- The `-domain` option is optional. The type of the object determines the power state being defined. If the object is a power domain, then the power state defined by that command is a power domain state.
- Except the first power state, you must specify all the additional power states with the `-update` option.

- The `-supply_expr` and `-simstate` options of the `add_power_state` command are not supported with the `-domain` option.
- A legal power state cannot refer to an illegal object state in its logic expression.
- Short hand notation of supply set handles is allowed in the logic expression of a power domain state. That is, supply set handles (`primary/default_retention/default_isolation`) can be directly used in logic expression without prefixing them with the power domain name. For example, consider the following UPF:

```
add_power_state -domain PDM -state PDM_ON1
{-logic_expr {primary==ON && default_retention==ON} }
```

Here, supply set handles `primary` and `default_retention` are referred directly in the logic expression without prefixing them with power domain name. VCS NLP interprets them as `PDM.primary` and `PDM.default_retention`.

The following table shows the VCS NLP behavior for the implicit and explicit supply sets in the current scope.

Implicit supply set	Explicit supply set	Command result
Not found	Not found	Error (object not found)
Not found	Found	Use explicit supply set present in the current scope
Found	Not found	Use implicit supply set
Found	Found	Error (conflict)

Referring Power Domain States in Group Logic Expression

You can refer power domain states in the logic expression of add_power_state -group. The following example depicts the usage:

```
set_scope mid
create_power_domain PDM
add_power_state -domain PDM -state PDM_ON1 \
    {-logic_expr {PDM.primary==ON &&
PDM.default_retention==ON}}
add_power_state -domain PDM -state PDM_ON2 \
    {-logic_expr {PDM.primary==OFF &&
PDM.default_retention==ON}} -update
set_scope/

create_power_domain PDT
add_power_state -domain PDT -state PDT_ON1 \
    {-logic_expr {PDT.primary==ON &&
PDT.default_retention==ON}}

create_power_state_group GROUP
add_power_state -group GROUP -state GROUP_ON \
    {-logic_expr {PDT == PDT_ON1 && mid/PDM == PDM_ON1}}
```

Here, top power domain PDT has one state of operation PDT_ON1. The power domain PDM of mid has two states of operation PDM_ON1 and PDM_ON2. Therefore, system PST can have the following two states of operations:

- Top in PDT_ON1 state and mid in PDM_ON1 state
- Top in PDT_ON1 state and mid in PDM_ON2 state

To restrict the system states to one, GROUP is used. The GROUP_ON group state restricts the system PST to the following state:

- Top in PDT_ON1 state and mid in PDM_ON1 state

Usage Examples

Example-1: The -domain option is optional

```
create_power_domain PDT
add_power_state PDT \
    -state PDT_ON1 {-logic_expr {primary == ON1 &&
default_retention == ON1}}
```

Here, the power state PDT_ON1 is interpreted as power state of the power domain PDT.

Example-2: Objects allowed in logic expression are supply sets, PSTs, Groups, and other power domains

```
create_power_domain PDT
set_scope mid
create_power_domain PDM
add_power_state -domain PDM \
    -state PDM_ON1 {-logic_expr {primary == ON1 &&
default_retention == ON1}}
set_scope /
add_power_state PDT \
    -state PDT_ON1 {-logic_expr {primary == ON1 && mid/PDM
== PDM_ON1}}
```

Error Scenarios

VCS NLP issues an error message for the following usages of add_power_state -domain:

Scenario-1: Unsupported options with the -domain option

```
create_power_domain PDT
add_power_state -domain PDT \
    -state PDT_ON1 {-supply_expr {...} -simstate}
```

Here, the `-supply_expr` and `-simstate` options cannot be specified with `-domain`.

Scenario-2: Power domain name must be simple

```
set_scope mid
create_power_domain PDM
set_scope /
add_power_state -domain mid/PDM \
    -state PDM_ON1 {-logic_expr {primary == ON1 &&
default_retention == ON1}}
```

You cannot use complex power domain names like `mid/PDM`.

Scenario-3: First power domain state cannot be specified with the `-update` option

```
create_power_domain PDT
add_power_state -domain PDT \
    -state PDT_ON1 {-logic_expr {primary==ON &&
default_retention==ON}} -update
```

You cannot specify the first power domain state with the `-update` option.

Scenario-4: Additional power states must be specified with the `-update` option

```
create_power_domain PDT
add_power_state -domain PDT \
    -state PDT_ON1 {-logic_expr {primary==ON1 &&
default_retention==ON}}
add_power_state -domain PDT \
    -state PDT_ON2 {-logic_expr {primary==ON2 &&
default_retention==ON}}
```

You cannot specify additional domain states without the `-update` option.

Scenario-5: A legal power state cannot refer to an illegal object state in its logic expression

```
create_power_domain PDT
create_power_state_group GRP
add_power_state -group GRP \
    -state GRP_ON1 {-logic_expr {PDT.primary == ON1} -illegal}

add_power_state -domain PDT \
    -state PDT_ON1 {-logic_expr {GRP == GRP_ON1}}
```

A legal domain state PDT_ON1 cannot refer to an illegal state GRP_ON1 of GRP.

Limitations of add_power_state Support

Following are the limitations of add_power_state support:

- Only one state can be defined per one add_power_state command
- Only one supply function is allowed in each supply expression
- -legal/-illegal options are not supported
- Interval function with -logic_expr is not supported
- CORRUPT_STATE_ON_CHANGE, CORRUPT_STATE_ON_ACTIVITY, and NOT_NORMAL simstates are not supported

Using associate_supply_set Command

The `associate_supply_set` command associates a supply set with a domain or strategy. The supply set includes the specific nets for each of the supplies (power, ground, and so on).

The following sections describe the `associate_supply_set` command:

- [Support for the UPF 2.1 associate_supply_set Command](#)
- [Support for the UPF 3.0 associate_supply_set Command](#)

Support for the UPF 2.1 associate_supply_set Command

Following is the syntax of the UPF 2.1 `associate_supply_set` command:

```
associate_supply_set supply_set_name  
      [ -handle supply_set_handle ]
```

Where,

- `supply_set_name` is a rooted name of a supply set to associate with a supply set handle
- `supply_set_handle` is the supply set handle with which the supply set is associated

When an implicit supply set (supply handle) is associated with another supply set, they are resolved to same real supply net Power/Ground pair, hence they have identical switching/corruption semantics. When two supply sets are associated, they are treated as virtually connected.

The following error checking applies to `associate_supply_set` command:

- Associating an implicit supply set (supply handle) either directly using `associate_supply_set` or some other command like `set_domain_supply_net` can be done only once
- It is an error to make circular associations
- You cannot use the `associate_supply_set` command with explicit supply sets in the `-handle` option
- Implicit supply set (supply handle) specified with the `-handle` option must be at or below the scope of supply set with which it is associated
- Associating supply handle of a domain to a supply set makes the supply set available in the domain to which the supply handle belongs to

Support for the UPF 3.0 `associate_supply_set` Command

The UPF 2.1 `associate_supply_set` command does not allow you to associate two explicit supply sets.

VCS NLP supports the UPF 3.0 `associate_supply_set` command syntax. [Table 3-5](#) describes the UPF 3.0 and UPF 2.1 syntax of the `associate_supply_set` command.

Table 3-5 UPF 3.0 associate_supply_set syntax

UPF 3.0 Syntax	UPF 2.1 Syntax
associate_supply_set supply_set_name_list <code>[-handle supply_set_handle]</code> Where, supply_set_name_list A list of rooted names of supply sets. -handle supply_set_handle The rooted name of a supply set of a power domain, power switch, or strategy.	associate_supply_set supply_set_name -handle supply_set_handle supply_set_name The rooted name of a supply set to associate with a supply set handle. -handle supply_set_handle The rooted name of a supply set of a power domain, power switch, or strategy.

The UPF 3.0 syntax allows you to specify a list of supply sets or supply set handles. These supply sets or supply set handles can be hierarchical names.

With this support, VCS NLP does not perform the following semantic checks related to UPF 2.1:

1. Multiple supply sets defined in an ancestor scope are associated with the supply set handle defined for a power domain.
2. Multiple supply sets defined in the scope of a power domain or a descendant scope are associated with the supply set handle defined for a power domain.

Following is the sample VCS NLP error message for 1 and 2:

```
Error-[UPF_MSSET_SET] Master Supply Set already set
test_associate_ss_handle_with_multi_ss.upf, 73
Master Supply Set for Supply Set Handle 'testbench/TOP/
PD4.primary' is already set to 'testbench/TOP/PD4_prim'.
Supply Set association with 'testbench/TOP/PD2_prim' will
be ignored.
Please verify the association of Supply Set Handle
```

3. The associations of supply sets with supply set handles form a loop of associations. Following is the sample VCS NLP error message:

```
Error-[UPF_LOAOS] Loop of associations of SupplySets  
test_associate_ss_nlp_loop.upf, 43  
The association of handle 'tb/top/A.primary' to supply  
set handle/reference 'tb/top/C.primary' forms a loop of  
associations.  
Please ensure that the root of the handle is not the same  
as the root of the supply set handle/reference
```

The UPF 2.1 syntax is internally interpreted as the UPF 3.0 syntax, and the above semantic checks are not done. For example, the following command:

```
associate_supply_set SS1 -handle PD1.primary
```

is equivalent to

```
associate_supply_set { SS1 PD1.primary }
```

Supply Set Association

Association of supply sets means associating or connecting their corresponding functions. Consider the following command:

```
associate_supply_set {SS1 SS2 mid/SSMid}
```

For each function of SS1, SS2, and mid/SSMid, SS1.function and SS2.function are the same supply nets or connected supply nets that are connected to mid/SSMid.function.

The following sections describe the association of supply sets:

- [Same Scope Association](#)

- [Association Across Scopes](#)

Same Scope Association

If two supply sets being associated are in the same scope, then this indicates that the corresponding functions of each supply set are:

- The same physical supply nets in the current scope
Or,
- The two supply nets that are connected to each other (in a different scope).

Association Across Scopes

If two supply sets being associated are in different scopes (one scope is an ancestor of another), then the association of two supply sets is equivalent to connecting (using the `connect_supply_net` command) each of the corresponding functions explicitly through an intermediate supply port that is created for each function.

Checks for Supply Set Association

VCS NLP performs the following checks for supply set association and updates to the supply set functions:

- [Update Supply Set Functions with Supply Nets](#)
- [Common Ancestor Requirement for associate_supply_set](#)

In this section, all references to supply net mean supply net or supply net handle. Supply net handles are supply nets that are created implicitly as a result of creation of the supply set or supply set handles. For example, SS.power, PD.primary.power.

Update Supply Set Functions with Supply Nets

Supply set functions are updated with supply nets using the `create_supply_set` command.

At a given scope, every group of associated supply set functions can be associated with only one supply net created using the `create_supply_net` command. VCS NLP issues an error message if the set of associated supply set functions in a scope is updated with multiple supply nets.

Examples

Example-1: Associated supply set functions are updated to two different supply nets in the same scope

VCS NLP issues an error message for the following UPF since the associated supply set functions `SS1.power` and `SS2.power` are updated to two different supply nets `V1` and `V2` respectively in the same scope.

```
associate_supply_set {SS1 SS2}
create_supply_set SS1 -update -function {power V1}
create_supply_set SS2 -update -function {power V2}
```

Following is the sample VCS NLP error message:

```
Error-[UPF_SSET_ASSOC_INVALID] Associated Supply Set
functions with different supply nets
assoc_two_ss_with_diff_net.upf, 18
```

Associated Supply Set functions 'tb/top/TOP.power' and 'tb/top/A.power' are updated to two different supply nets 'tb/top/VDD' and 'tb/top/VDDA' respectively in the same scope, at assoc_two_ss_with_diff_net.upf,19.

In a scope, supply net should be specified for only one of the associated supply set functions.

Example-2: Associated supply set functions are updated to two different supply nets in the same scope. Supply nets are connected in a different scope

If Example-1 is modified such that V1 and V2 are connected in a different scope, VCS NLP still issues the

UPF_SSET_ASSOC_INVALID error message (mentioned in the above example) as the associated supply set functions SS1.power and SS2.power are updated to two different supply nets V1 and V2 respectively in the same scope.

```
associate_supply_set {SS1 SS2}
set_scope U1
create_supply_port V1
create_supply_port V2
create_supply_net VDD
connect_supply_net VDD -ports {V1 V2} # V1 and V2 are
connected
set_scope ..
connect_supply_net V1 -ports U1/V1
connect_supply_net V2 -ports U1/V2
create_supply_set SS1 -update -function {power V1}
create_supply_set SS2 -update -function {power V2}
```

Example-3: Associated supply set functions are updated to two different supply net handles in the same scope.

The following UPF is valid as the associated supply set functions SS1.power and SS2.power are updated to different supply net handles (and not different supply nets).

```

create_supply_set SS1
create_supply_set SS2
create_supply_set SSTop
create_power_domain PDTop
associate_supply_set {SS1 SS2}
create_supply_set SS1 -update -function {power
PDTop.primary.power}
create_supply_set SS2 -update -function {power SSTop.power}

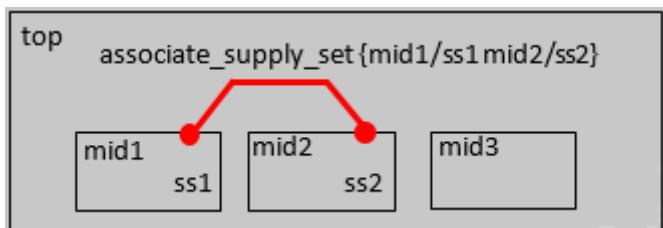
```

The supply net handles are added to the same group of associated supply set functions. The group contains SS1.power, SS2.power, PDTop.primary.power, and SSTop.power. After this point, only PDTop.primary.power and SSTop.power can be further refined to supply nets.

Common Ancestor Requirement for associate_supply_set

For every `associate_supply_set` command there must be a supply set that is created in a common ancestor scope of all the supply sets that are being associated. VCS NLP issues an error message if in the group of supply sets that are being associated, there is no supply set that is created in a common ancestor scope of all the supply sets.

In order to associate two supply sets in the sibling scope, they must be connected with a supply defined in their common ancestor scope. Consider the following figure:



In the above figure, in order to associate mid1/ss1 and mid2/ss2, a top level supply must be created in their common ancestor scope top, which makes supply mid1/ss1 (mid2/ss2) available in all mid1, mid2, top, and mid3.

To prevent the spreading of this unintentional supply availability, a group of supply sets can be associated only when the group contains a supply in their common ancestor scope.

Examples

Example-1

VCS NLP issues a warning message for the following command because in the group of supply sets being associated there is no supply set that is created in a common ancestor scope.

```
associate_supply_set {u1/SS1 u2/SS2 u3/SS3}
```

Following is the sample VCS NLP warning message:

```
Warning-[UPF_COMPAT_SSET_COMMON_HIER] No common ancestor in
supply set association
./no_common_ancestor_in_assoc.upf, 89
No common ancestor found in supply set association.
No common ancestor found in supply set association, this may
not be supported by other tools.
```

Example-2

For the following command, VCS NLP issues the UPF_COMPAT_SSET_COMMON_HIER warning message (mentioned in the above example) because in the group of supply sets being associated there is no supply set that is created in a common ancestor scope of all the supply sets.

```
associate_supply_set {u3/u2/SS2 u3/SS3 u1/SS1}
```

Example-3

The following command is valid because there is a common ancestor scope `u3` where the supply set `u3/SS3` exists and is a part of the group of supply sets that is being associated.

```
associate_supply_set {u3/u2/SS2 u3/SS3 u3/u1/SS1}
```

Using `add_state_transition` Command

The `add_state_transition` command defines the named transitions between power states of an object and allows you to specify the legality of the transitions. The `add_state_transition` command is defined in UPF 3.0 and it replaces the `describe_state_transition` command which is defined in UPF 2.0. However, to maintain backward compatibility, both `describe_state_transition` and `add_state_transition` commands are supported with VCS NLP.

Following is the syntax of the `add_state_transition` command:

```
add_state_transition
[-supply | -domain] object_name
[-transition {transition_name
[-from from_list -to to_list]
[-paired {{from_state to_state}*}}]
[-legal | -illegal]
[-through through_list]
}]*
```

Example:

```
add_state_transition -domain PDA
-transition {turn_on -from OFF_MODE -to NORMAL_MODE}
-transition {suspend -from NORMAL_MODE -to SLEEP_MODE}
-transition {resume -from SLEEP_MODE -to NORMAL_MODE}
-transition {turn_off -from NORMAL_MODE -to OFF_MODE}
add_state_transition -domain PDA -update
-transition {error1 -from OFF_MODE -to SLEEP_MODE -illegal}
```

The semantics and error conditions of the `add_state_transition` command is similar to the `describe_state_transition` command. For information on the `describe_state_transition` command, see “[Constraining Low Power Coverage Using the `describe_state_transition` Command](#)” section.

Note:

VCS NLP converts the `add_state_transition` command as `describe_state_transition` for each transition. That is, the `add_state_transition` command does not support `-group`, `-complete`, and `-update` options, and reports all the error messages of `describe_state_transition`.

The `-supply` and `-domain` options specify the kind of object to which the `add_state_transition` command applies. If none of these options are specified, then the object can be any of the following types:

PowerDomain, SupplySet, PowerStateGroup, PowerStateTable, SupplyPort, SupplyNet, or PowerSwitch.

Note:

- When object is a `PowerStateTable`, acceptable values in the `from/to` state are the states added on the PST using the `create_pst` command.
- When object is a `SupplyPort` or `SupplyNet`, acceptable values in the `from/to` state are the port states.
- When object is a `PowerSwitch`, acceptable values in the `from/to` state are the states defined on the power switch.
- The `-through` option is supported in the same way as `describe_state_transition`.

Limitations

- The `-group`, `-complete`, `-update`, `-model` and `-instance` options are not supported. VCS NLP issues an error message for these options.
- Wildcards are not supported.

Using bind_checker Command

The `bind_checker` command inserts checker modules into a design without modifying the design code or introducing functional changes. The mechanism for binding the checkers to design elements relies on the SystemVerilog `bind` directive. The `bind` directive causes one module to be instantiated within another, without having to explicitly alter the code of either. This facilitates the complete separation between the design implementation and any associated verification code.

Following is the syntax of the `bind_checker` command:

Syntax

```
bind_checker instance_name
  -module checker_name
  [-elements element_list/UPF object*]
  [-ports {{port_name net_name/upf_name*} {}}]
  [-parameters {{parameter_name constant_expression/
    upf_name*} {}}]
  [-name_prefix string]
  [-name_suffix string]
```

Arguments

`instance_name`

The name used to instance the checker module in each design element.

`-module checker_name`

The name of a checker module containing the verification code. The verification code itself shall be written in SystemVerilog, but it can be bound to either a SystemVerilog or VHDL instance.

`-elements element_list`

The list of design elements or UPF objects like Power Domain, Isolation Strategy, Retention Strategy, Power Switch.

`-ports {{port_name net_name} {}}`

The association of signals to the checker's ports.

`port_name` is a port defined on the interface of `checker_name` and `net_name` is a name of a net relative to the scope where `checker_name` is being instantiated.

Signals in the target instance are bound by position to inputs in the bound checker module through the portlist. Thus, the bound module has access to all signals in the scope of the target instance by simply adding them to the port list, which facilitates sampling of the arbitrary design signals.

-parameters {{ parameter_name constant_expression/
upf_name } }

Allows parameter passing to the checker module.

-name_prefix prefix

In case the checker is bound to strategy, then name_prefix option suggests the prefix to be used for uniquely naming the checker instance.

-name_suffix suffix

In case the checker is bound to a strategy, then name_prefix option suggests the suffix to be used for uniquely naming the checker instance.

* UPF Name Alias

As part of the extensions to the UPF bind_checker command, the UPF names must be prefixed with '@' for distinction.

For example, consider the following custom bind file. Here, the checker to each power domain is binded using the `-elements` option, which uses named aliases (`@$PD`) .

```
proc primary_power_off {} {
    foreach PD [query_power_domain *] {
        .....
        bind_checker primary_power_off_msg \
            -module primary_power_off \
            -elements [list @$PD] \
            -parameters $params_list \
            -ports $ports_list
    }
}
primary_power_off
```

Since you are binding the checker to a power domain (extension of `bind_checker`), UPF name alias ‘@’ must be used with the `-elements` option.

- If the ports list passed with the `-ports` option contains any of the UPF variables (returned by the UPF query command), then the target for the binding specified with `-elements` must be any UPF object like Power Domain, Power Switch, Isolation Strategy, and Retention Strategy.
- Cross-binding is supported between power domain, isolation and retention strategies only. For example, elements returned by `query_isolation` can be bound to a power domain.
- Binding of PST supplies returned by the `query_pst` command as ports is not supported.

Using `create_power_switch` Command

The `create_power_switch` command (see “[create_power_switch](#)” for syntax) creates an instance of a power switch in the scope of a power domain. The switch has at least one input supply port and one output supply port. When the switch is on, it connects the input supply port (or one of multiple input supply ports) to the output supply port. When the switch is off, the output supply port is shut down and has no power.

The command can optionally specify the characteristics of the acknowledge port or ports. The ports specified in the command are the connections points of the power switch.

The following sections describe how to use the `create_power_switch` command:

- [Handling `create_power_switch -ack_port` Connections](#)
- [Using Supply Set of the Power Switch for Acknowledge Port Corruption](#)

Handling `create_power_switch -ack_port` Connections

The following points describe how the connections are handled between the `-ack_port` option of the `create_power_switch` command and its corresponding design nets and ports.

- For `-ack_port` of type variable whose resolution function is not defined, the connection is established if the design node does not contain any driver. VCS NLP issues an error message if the design node contains a driver.

- For `-ack_port` of type wire whose resolution function is defined, the connection is established if the design node does not contain any driver. VCS NLP issues a warning message and adds the acknowledge port connection as a multiple driver, if the design node contains a driver.

Using Supply Set of the Power Switch for Acknowledge Port Corruption

VCS NLP supports `-supply_set` option of the `create_power_switch` command.

The `-supply_set` option allows you to specify related supply set for both the control and acknowledge ports of the power switch. If any of the supply function of this related supply set goes OFF, then the acknowledge port is corrupted.

For backward compatibility, enable the `SKIP_ACK_PORT_CORRUPTION` variable in the Backward Compatibility configuration file and recompile. By default, this variable is disabled.

Using `create_supply_net` Command

The `create_supply_net` command (see “[create_supply_net](#)” for syntax) creates a supply net to supply power or ground to a power domain. You must specify the name for the new supply net and the name of the existing power domain in which the supply net will be used. The supply net is specified as a simple (not hierarchical) name.

The following sections describe how to use the `create_supply_net` command:

- Support for `-resolve parallel` Option
- Specifying User-Defined Supply Net Resolution As a Verilog Function or Task in the Package
- Resolution Mechanism for Supply Net With “strong” and “weak” Resolutions
- Design Namespace Conflict Resolution for the Creation of Supply Ports/Nets

Support for `-resolve parallel` Option

VCS NLP supports the `-resolve parallel` option with the `create_supply_net` command.

Note:

If a `create_power_switch` command in the UPF and power switch instantiated in the design drive the same supply net, then the `create_supply_net` command for that supply net must have `-resolve parallel`.

Limitation

- VCS NLP does not check whether the root supply for the parallel switches is same.

Specifying User-Defined Supply Net Resolution As a Verilog Function or Task in the Package

VCS NLP supports custom resolution of supply nets. You can specify your own supply net resolution as a Verilog function or task in your package and use it in the UPF as follows:

```
create_supply_net net_name -resolve  
package_name::resolution_name
```

VCS NLP resolves the Verilog task or function and uses it to build the resolution gate of supply nets. This feature helps you to write your own supply net resolution function based on your supply network use model.

Note:

Resolution function must be commutative and associative.

Following is the example of UPF and resolution function:

Example 3-3 UPF Example Code

```
set_design_top top  
set_design_attributes -elements {.} -attribute enable_bias  
TRUE  
  
create_power_domain TOP -include_scope  
create_power_domain CORE1 -elements {c1}  
  
create_supply_net combined_vdd -resolve my_pkg::my_res  
create_supply_net supply_net1
```

Example 3-4 Resolution Function Example Code

```
package my_pkg;  
import UPF::*;


```

```

function state partOnPriority (state a, state b);
    if ((a == UNDETERMINED) || (b == UNDETERMINED))
        return UNDETERMINED;
    else if ((a == PARTIAL_ON) || (b == PARTIAL_ON))
        return PARTIAL_ON;
    else if ((a == FULL_ON) || (b == FULL_ON))
        return FULL_ON;
    else
        return OFF;
endfunction

task my_res(input supply_net_type a, input supply_net_type
b, output supply_net_type out);
    out = '{ partOnPriority(a.state, b.state), (a.voltage >
b.voltage) ? a.voltage : b.voltage }';
endtask
endpackage

module tb ;
import UPF::*;
import my_pkg::*;

endmodule

```

Limitations

- You can only write resolution function inside a package. Module, program-block, and classes are not supported.
- The resolution function must be in the form of 2-input supply nets and 1-output supply net.

Resolution Mechanism for Supply Net With “strong” and “weak” Resolutions

This document describes the resolution mechanism for supply net with “strong” and “weak” resolutions in the following sections:

- [Resolution Mechanism for Supply Net With “strong” Resolution](#)
- [Resolution Mechanism for Supply Net With “weak” Resolution](#)

Resolution Mechanism for Supply Net With “strong” Resolution

You can use the following resolution method in the `create_supply_net` command if the supply net is driven by multiple supply drivers:

`-resolve strong`

Outputs of multiple supply drivers, each having a unique source, may be connected to the same supply net.

Syntax example:

```
create_supply_net local_vdd_3  
-resolve strong
```

A supply net with `strong` resolution shall be resolved to `FULL_ON` only when at least one of the supply driver is `FULL_ON` and no other supply driver is `UNDETERMINED`. If the supply net is resolved to `FULL_ON`, the voltage value of the supply net shall be the maximum value of the conflicting voltages.

Table 3-6 describes the semantics of the supply net state and voltage resolution for `-resolve strong`:

Table 3-6 Resolution Mechanism for -resolve strong

Supply Source1 (sw1)	Supply Voltage (v1)	Supply Source2 (sw2)	Supply Voltage (v2)	Resolved Supply Net State	Resolved Supply Net Voltage
FULL_ON	1.5	FULL_ON	2.5	FULL_ON	2.5 (maximum of SW1 and SW2 voltage values)
FULL_ON	1.5	OFF	0	FULL_ON	1.5 (voltage value of the root supply driver)
OFF	0	FULL_ON	2.5	FULL_ON	2.5
OFF	0	OFF	0	OFF	0
PARTIAL_ON	0	FULL_ON	2.5	FULL_ON	2.5
FULL_ON	1.5	PARTIAL_ON	0	FULL_ON	1.5
PARTIAL_ON	0	OFF	0	PARTIAL_ON	0
OFF	0	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	PARTIAL_ON	0	PARTIAL_ON	0
UNDETERMINED	-	FULL_ON	2.5	UNDETERMINED	0
FULL_ON	1.5	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	OFF	0	UNDETERMINED	0
OFF	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	PARTIAL_ON	0	UNDETERMINED	0
PARTIAL_ON	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	UNDETERMINED	-	UNDETERMINED	0

Resolution Mechanism for Supply Net With “weak” Resolution

You can use the following resolution method in the `create_supply_net` command if the supply net is driven by multiple supply drivers:

`-resolve weak`

Outputs of multiple supply drivers, each having a unique source, may be connected to the same supply net.

Syntax example:

```
create_supply_net local_vdd_3  
-resolve weak
```

A supply net with `weak` resolution shall be resolved to `FULL_ON` only when at least one of the supply drivers is `FULL_ON`. If there are multiple drivers in `FULL_ON` state, the voltage value of the supply net shall be the minimum value of conflicting voltages.

[Table 3-7](#) describes the semantics of the supply net state and voltage resolution for `-resolve weak`:

Table 3-7 Resolution Mechanism for -resolve weak

Supply Source1 (sw1)	Supply Voltage (v1)	Supply Source2 (sw2)	Supply Voltage (v2)	Resolved Supply Net State	Resolved Supply Net Voltage
FULL_ON	1.5	FULL_ON	2.5	FULL_ON	1.5 (minimum of SW1 and SW2 voltage values)
FULL_ON	1.5	OFF	0	FULL_ON	1.5
OFF	0	FULL_ON	2.5	FULL_ON	2.5
OFF	0	OFF	0	OFF	0
PARTIAL_ON	0	FULL_ON	2.5	FULL_ON	0
FULL_ON	1.5	PARTIAL_ON	0	FULL_ON	0
PARTIAL_ON	0	OFF	0	PARTIAL_ON	0
OFF	0	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	PARTIAL_ON	0	PARTIAL_ON	0
UNDETERMINED	-	FULL_ON	2.5	FULL_ON	2.5
FULL_ON	1.5	UNDETERMINED	-	FULL_ON	1.5
UNDETERMINED	-	OFF	0	UNDETERMINED	0
OFF	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	UNDETERMINED	-	PARTIAL_ON	0
UNDETERMINED	-	UNDETERMINED	-	UNDETERMINED	0

Design Namespace Conflict Resolution for the Creation of Supply Ports/Nets

Creating a supply port or net with the same name as in the design for the same scope is not allowed as per *IEEE Std 1801*. In the following example, `vdd` is created in both RTL and UPF. VCS issues the [UPF_ULOAD] UPF object already defined error message for this case.

RTL	UPF
<pre>module top (vdd, in1, in2); output vdd; ... endmodule</pre>	<pre>set_design_top top create_power_domain PD1 create_supply_net vdd create_supply_port vdd -domain PD1- direction out connect_supply_net vdd -ports vdd</pre>

If the supply ports or nets created in the UPF share the same namespace with the ports or nets that are present in the design, you can use the `-power=rtlpg` option at compile time to enable VCS NLP to automatically match the supply nets or ports created in UPF with the nets or ports in RTL within the same scope.

Using map_isolation_cell Command

The `map_isolation_cell` command allows you to map a particular isolation strategy to the user-defined library cell or behavioral model, and instructs VCS NLP to use this isolation cell for clamping. You must specify the `set_isolation` and `set_isolation_control` commands. The `set_isolation_control` command is used to specify the location of the isolation strategy.

Following is the use model of the `map_isolation_cell` command:

```
set_isolation isolation_name
  -domain ref_domain_name
  [-applies_to <inputs | outputs | both>]
  [-isolation_power_net net_name]
  [-isolation_ground_net net_name]
  [-elements element_list]
  [-clamp_value {< 0 | 1 | z | latch >}]
  [-location <self | parent | automatic>]
map_isolation_cell isolation_name
  -domain domain_name
  [-elements element_list]1
  [-lib_cells lib_cells_list]1
  [-lib_cell_type lib_cell_type]
  [-lib_model_name model_name {-port {port_name
    net_name}}*]
```

* Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

Key Points to Note

- If `-lib_model_name` is specified:
 - VCS NLP uses `model_name` for isolation modeling
 - Logic and supply ports are connected as specified by the `-port` options
 - All supply connections must be specified with `-port`
 - Ports of type inout are not supported for supply connections
 - Neither implicit nor automatic supply net connections apply

- No corruption semantics apply to the model specified using –
`lib_model_name`
- VCS NLP generates an error if:
 - `domain_name` does not indicate a previously created power domain
 - `isolation_name` is not specified
 - `-lib_model_name` is not specified

Isolation Cell Modeling Example

Each `set_isolation` command contains one corresponding `map_isolation_cell` command. For example, consider the following user-defined isolation cell test case:

Example 3-5 User-defined Isolation Cell With Multiple Supplies and Enables

```
module Complex_ISO_Mod_Zero(input D, output Z, input power1,
    input power2, input backup, input ground1, input ground2,
    input EN1, input EN2, input EN3, input EN4 );

    wire temp;

    assign Z = ({power1,power2, backup, ground1, ground2} ===
5'b11100) ? temp : 1'bx;

    assign temp = (EN1 | EN2 | EN3 | EN4 | D) ;

endmodule
```

UPF Command

```
set_isolation ISO1 -domain PD1 -isolation_signal {iso_ctrl1
iso_ctrl2 iso_ctrl3 iso_ctrl4} -clamp_value 0 -applies_to
outputs -location parent -isolation_supply_set {TOP_prim
PD1_prim PD2_prim}

map_isolation_cell ISO1 -domain PD1 \
```

```

-lib_model_name Complex_ISO_Mod_Zero \
-port {EN1 isolation_signal[0]} \
-port {EN2 isolation_signal[1]} \
-port {EN3 isolation_signal[2]} \
-port {EN4 isolation_signal[3]} \
-port {power1 isolation_supply_set[0].power} \
-port {power2 isolation_supply_set[1].power} \
-port {backup isolation_supply_set[2].power} \
-port {ground1 VSS} \
-port {ground2 VSS}

```

Note:

- You must use the `-port` option to connect ports like enable, power, and ground.
- The `-port` option need not specify model input and output data port connection. Except these two data ports, all other ports must be connected using `-port`.
- It is assumed that the cell is the power aware model having PG pins in module interface, and manages its corruption semantics.

Using Value Conversion Table (VCT) for Model Supply Connection

- The default VCT for isolation model power port connection is `UPF2SV_LOGIC`. You can use the following syntax to change this VCT:

```

set_design_attributes -models <model_name>
-attribute SNPS_default_power_upf2sv_vct
<vct_name>

```

- The default VCT for isolation model ground port connection is `UPF_GNDZERO2SV_LOGIC`. You can use the following syntax to change this VCT:

```
set_design_attributes -models <model_name>
-attribute SNPS_default_ground_upf2sv_vct
<vct_name>
```

Note:

In cases where isolation supplies are not defined as part of the set_isolation command, and default_isolation supply set is picked up from the power domain, the usage of <Domain>.default_isolation.<power/ground> or <Explicit Supply Set>.power along with the -ports option for connecting power and ground pins of the isolation cell is supported.

```
create_supply_set PD2_iso
..
create_power_domain PD2 -supply {primary PD2_prim}
-supply {default_isolation PD2_iso}
..
set_isolation iso_pd1      -domain PD2      -clamp_value 1
-source PD1_prim
set_isolation_control iso_pd1  -domain PD2
-isolation_signal iso_enable  -isolation_sense high
-location parent

map_isolation_cell iso_pd1 -domain PD2 \
-lib_model_name ISO_Mod_Zero \
-port {enable iso_enable} \
-port {power PD2_iso.power} \
-port {ground PD2_iso.ground}
```

Or,

```
map_isolation_cell iso_pd1 -domain PD2 \
-lib_model_name ISO_Mod_Zero \
-port {enable iso_enable} \
-port {power PD2.default_isolation.power} \
-port {ground PD2.default_isolation.ground}
```

Also, generic name `isolation_supply_set` can be used.

```
map_isolation_cell iso_pd1 -domain PD2 \
    -lib_model_name ISO_Mod_Zero \
    -port {enable iso_enable} \
    -port {power isolation_supply_set.power} \
    -port {ground isolation_supply_set.ground}
```

Limitations

Following are the limitations of the `map_isolation_cell` command:

- Non power-aware models are not supported.
- In case of mixed designs, the `map_isolation_cell` command is supported only for the isolation cells inserted in Verilog.
- The `-elements` option is not supported with the `map_isolation_cell` command.
- Auto corruption is not supported.

Using `query_cell_mapped` Command

The `query_cell_mapped` command identifies the cell that is used for the named instance. If the specified instance module has a db cell mapped to it, then this command returns the cell name in the `<library>/<cell>` format, else it returns null string.

Following is the syntax of the `query_cell_mapped` command:

```
query_cell_mapped instance_name
```

Where, `instance_name` is the name of the instance.

Example:

```
query_cell_mapped top/a/my_inst
```

Limitations

Following are the limitations of the `query_cell_mapped` command:

- This command supports only Verilog instances.
- Wildcards are not supported.

Querying Power Domain Information of an Instance

VCS NLP allows you to query the power domain information of a given instance using the `query_power_domain_element` UPF command. This command returns the hierarchical power domain name corresponding to the given instance. Following is the syntax:

```
query_power_domain_element design_element
```

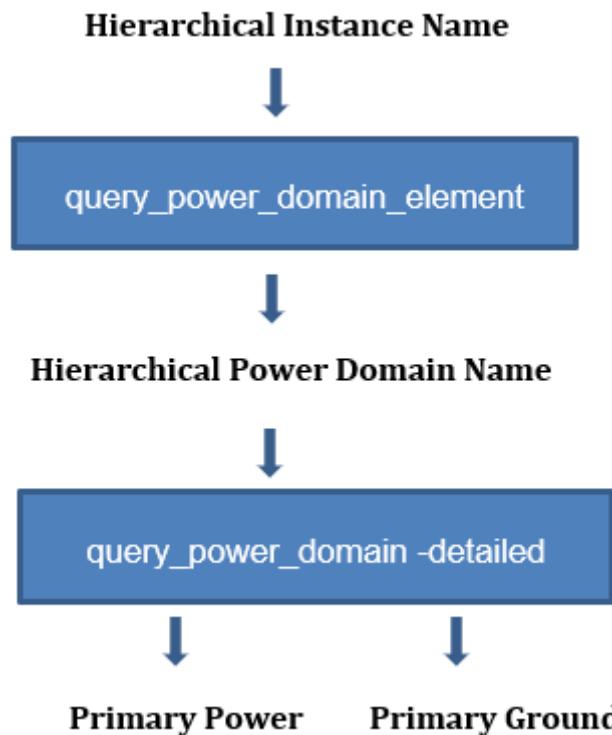
Where, `design_element` is the hierarchical element name for which the power domain information is to be returned.

The specified `design_element` cannot start with `..` or `/`, that is, the hierarchical instance name shall be referenced from the current scope, and reside in the current scope or below it.

You must specify the `query_power_domain_element` command in the Tcl file provided with the `-lpa_bind` option.

Once you have the power domain, you can get to the power and ground net handles using the `query_power_domain -detailed` command. The `primary_power_net` and `primary_ground_net` arguments can be used to get the power and ground net handles respectively.

Figure 3-1 Querying Power Domain Information of an Instance



Use Model

Following is the use model:

```
set PD [query_power_domain_element <hier_instance_name from  
current scope> ]  
array set pd_detail [query_power_domain $PD -detailed]  
set PPN [list PPN $pd_detail(primary_power_net)]  
set PGN [list PGN $pd_detail(primary_ground_net)]
```

Once you have the power and ground information for the given instance in PPN and PGN, you can use it for further processing.

Expected Output for Specific Scenarios

The expected output of `query_power_domain_element` for a few specific scenarios is as follows:

- This command returns nearest parent power domain name for the following scenarios:
 - If a power domain is not defined for a design instance
 - The hierarchy is inside the DB mapped instance
 - Specified `design_element` is other than an instance
- This command returns NULL for the following scenarios:
 - The design element specified is above power top
 - The design element specified does not exist

VCS NLP issues the following warning message for these two scenarios:

```
Warning-[UPF_QUERY_PDE_UNRESOLVED_PATH]
query_power_domain_element command failed.
<FILE>, <LINE>
<DESIGN_ELEMENT> under current scope <SCOPE> could not
be resolved to any valid design element.
Please specify a valid design element.
```

- The master module of the instance is marked as blackbox (power blackbox) and a power domain is not explicitly defined on it
- The instance specified is inside a black box (power blackbox)

VCS NLP issues the following warning message for these two scenarios:

```
Warning-[UPF_QUERY_PDE_BLACKBOX]
query_power_domain_element command failed.
<FILE>,<LINE>
Hierarchical design element name <DESIGN_ELEMENT> under
current scope <SCOPE> is either a blackbox instance that
is not explicitly partitioned into a power domain or an
instance inside blackbox.
```

Example

Consider the following files:

Example 3-6 test.v

```
module tb();
    reg in ;
    wire out ;

    import UPF::*;

    initial begin
        supply_on("VDD_top",1.2);
        supply_on("VSS_top",0.0);
        supply_on("VDD_1",1.3);
        supply_on("VSS_1",0.0);
        in = 1;

        #20 ; $finish ;
    end
    top top_unit(in,out);

endmodule
module top(input in , output out);
    test U0(in,out);
endmodule
module test(input in , output out);
    assign out = ~in ;
```

```
endmodule
```

Example 3-7 Design: checker.sv

```
module chk_module(ppn,pgn);
  input ppn,pgn;
  import UPF::*;
  supply_net_type ppn,pgn;
  parameter string PPN = "";
  parameter string PGN = "";
  parameter string PD_NAME = "";

  always @ (ppn,pgn) #0.1 begin
    $display("BIND_CHECK : [%t] PD_NAME = %s , ppn =
    %s,{%s,%d} , pgn=%s,{%s,%d}" , $time , PD_NAME , PPN ,
    ppn.state.name, ppn.voltage , PGN , pgn.state.name,
    pgn.voltage);
  end
endmodule
```

Example 3-8 Tcl File: test.tcl

```
proc domain_check {} {
  foreach instance [find_objects . -transitive true -pattern
  * -object_type inst] {

    set PD [query_power_domain_element $instance]
    puts "Q_bind : instance=$instance , PD=$PD"

    array set pd_detail [query_power_domain $PD -detailed]

    set ppn [list ppn $pd_detail(primary_power_net)]
    set pgn [list pgn $pd_detail(primary_ground_net)]
    set PPN [list PPN $pd_detail(primary_power_net)]
    set PGN [list PGN $pd_detail(primary_ground_net)]
    set PD_NAME [list PD_NAME $pd_detail(domain_name)]

    set ports_list {}
    lappend ports_list $ppn $pgn
    set params_list {}
    lappend params_list $PD_NAME $PPN $PGN
    bind_checker BIND_CHECK \
```

```

-module chk_module \
-elements @$PD \
-parameters $params_list \
-ports $ports_list
array unset pd_detail
}
}
domain_check

```

Example 3-9 UPF File: test.upf

```

set_design_top top
create_supply_net VDD_top
create_supply_port VDD_top
connect_supply_net VDD_top -ports VDD_top
create_supply_net VSS_top
create_supply_port VSS_top
connect_supply_net VSS_top -ports VSS_top

create_supply_set SS_top -function {power VDD_top} -function
{ground VSS_top}
create_power_domain PD_top      -supply {primary SS_top}  -
include_scope

create_supply_port VDD_1
create_supply_net VDD_1
connect_supply_net VDD_1 -ports {VDD_1}
create_supply_port VSS_1
create_supply_net VSS_1
connect_supply_net VSS_1 -ports {VSS_1}
create_supply_set SS_1 -function {power VDD_1} -function
{ground VSS_1}
create_power_domain PD_1 -elements { U0 } -supply {primary
SS_1}

```

1. Execute the following compile command:

```
% vcs test.v checker.sv -upf test.upf -lpa_bind
test.tcl -sverilog
```

Compile Output:

```
Q_bind : instance=U0 , PD=tb/top_unit/PD_1
```

In this output, you can see that the `query_power_domain_element` command returns the hierarchical power domain of instance U0.

2. Execute the following simulation command:

```
% simv
```

Simulation Output:

```
BIND_CHECK : [ 0] PD_NAME = tb/top_unit/  
PD_1 , ppn = tb/top_unit/VDD_1,{FULL_ON, 1300000} ,  
pgn=tb/top_unit/VSS_1,{FULL_ON, 0}
```

Where, `ppn` and `pgn` show the power and ground information of power domain `PD_1`.

Limitations

Following are the limitations of this feature:

- This feature is not supported for `power_top`
- Wildcards are not supported

Using `set_design_top` Command

VCS NLP allows you to specify either the module name or instance name as an argument to the `set_design_top` command. For any string passed with the `set_design_top` command, VCS NLP first checks for the existence of the module name.

If the specified string is a module, VCS NLP finds the first instance of that module and sets it as design top. If the specified string is not a module, VCS NLP checks whether it is an instance or not. If it is an instance, VCS NLP sets it as design top. If the specified string is not found in the design, then VCS NLP generates an error message.

Using `set_equivalent` Command

The `set_equivalent` command declares that the specified supplies are electrically or functionally equivalent. This command returns an empty string on success and an error on failure.

Following is the syntax of the `set_equivalent` command:

```
set_equivalent
  [-function_only]
  [-nets supply_net_name_list]
  [-sets supply_set_name_list]
```

Where,

`-function_only`

Specifies that the supplies are functionally equivalent rather than electrically equivalent.

If the `-function_only` option is specified, then the supplies are declared to be functionally equivalent only. If `-function_only` is not specified, the supplies are declared to be electrically equivalent, which implies that they are also functionally equivalent.

`-nets`

Defines equivalence for a list of supply port, supply net, and/or macro internal PG pin names. When supply ports are listed, equivalence applies to the supply nets connected to the port.

`-sets`

Defines equivalence for a list of supply sets and/or supply set handles.

Note:

You cannot use both `-nets` and `-sets` options with the `set_equivalent` statement. You can only use one of these options.

Example

```
set_equivalent -nets { vss1 vss2 ground }
set_equivalent -function_only -nets { vdd_wall vdd_battery }
set_equivalent -function_only -sets {sys/aon_ss mem/
PD.primary}
```

Limitations

- Macros are not supported for electrical equivalence
- Error message is not issued for a supply source without supply pad or macro port

- For -function_only, no assertion is issued during the simulation when functionally equivalent nets go to different states
-

Using set_isolation Command

The `set_isolation` command allows you to specify an isolation strategy for ports on the interface of a power domain. The isolation strategy for a power domain is applied at the domain boundary, as required to ensure correct electrical and logical functionality when domains are in different power states.

The following sections describe the `set_isolation` command:

- [Use Model](#)
 - [Examples](#)
 - [Precedence Details](#)
 - [Defining Isolation Strategies](#)
 - [Error Checking for Isolation Strategy with -location parent on Top-level Ports](#)
-

Use Model

Following is the syntax of the `set_isolation` command:

```
set_isolation isolation_name
  -domain ref_domain_name
  [-elements element_list]
  [-exclude_elements exclude_list]
  -source source_supply_ref
  -sink sink_supply_ref
```

```

[-diff_supply_only <TRUE | FALSE>]
[-applies_to <inputs | outputs | both>]
[-applies_to_clamp <0 | 1 | any | Z | latch | value>]1
[-applies_to_sink_off_clamp <0 | 1 | any | Z | latch | value>]1
[-applies_to_source_off_clamp <0 | 1 | any | Z | latch | value>]1
[-isolation_power_net net_name]
[-isolation_ground_net net_name]
[-no_isolation]
[-force_isolation]1
[-location <automatic | self | fanout | fanin1 | faninout1 | parent | sibling1>]
[-clamp_value {<0 | 1 | Z | latch>}*]
[-isolation_signal signal_list <>]
[-isolation_sense <high | low | {<high | low>}*>]]
[-isolation_supply_set supply_set_list]
[-isolation_sense {<high | low>}*]
[-name_prefix string]1
[-name_suffix string]1
[-sink_off_clamp {<0 | 1 | any | Z | latch | value> [simstate_list]}]1
[-source_off_clamp {<0 | 1 | any | Z | latch | value> [simstate_list]}]1
[-instance {{instance_name port_name}*}]1
[-transitive <TRUE | FALSE>]1
[-update]

```

* Indicates that the item in curly braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

The `set_isolation` command specifies the ports on `ref_domain_name` to isolate using a specified strategy. The `-source/-sink/-diff_supply_only` options are filters similar to `-applies_to`. These options are used to filter the set of elements for a given `set_isolation` command invocation.

For any port in the `prefilter_element_list` (`elements` specified in the `-elements` list), the following filtering functions are applied:

- `-source` filters the ports receiving a net that is driven by logic powered by the supply set
- `-sink` filters the ports driving a net that fans-out to logic powered by the supply set
- When both `-source` and `-sink` are specified, a port is included if it has a source and a sink as specified
- With `-diff_supply_only`, no isolation is introduced into the path from the driver to the receiver for an isolation strategy defined for a port on the interface of `ref_domain_name`, where the driver is powered by the same supply as a receiver of the port
- `-applies_to` filters the ports within a domain for which the isolation strategy with specified mode is defined
- When one or more of the above filters are used together, then `effective_element_list` contains only those elements that satisfy all the filters. If an element in `effective_element_list` is not on the interface of `ref_domain_name`, then it is not isolated.

The `-source` and `-sink` options require supply sets. The `-diff_supply_only` option can be used without supply sets as the difference is evaluated on a supply net basis.

If `-applies_to` is not specified, and if at least one filter (`-source/-sink/-diff_supply_only`) or `-elements` is specified, the default value of `-applies_to` is taken as both. This is as per the IEEE Standard 1801.

If a filter (-source/-sink/-diff_supply_only) or -elements is not specified, the default value of -applies_to is taken as outputs. This behavior exists in order to be backward compatible with UPF 1.0. However, this is not compatible with the latest IEEE Standard 1801.

To get the IEEE Standard 1801 compliant behavior, use the -power=default_applies_to_dont_filter compile-time option. This option sets the default filter as both. Therefore, the strategy that has no filters and -elements applies to both input and output ports.

Examples

Basic Source–Sink Policy

The basic usage of -source/-sink is illustrated here. The -source and -sink options use different supply sets. Power domain specified with -domain is considered as a reference domain for the isolation strategy. The location (self/parent/automatic) of isolation cell is being evaluated with the reference to the domain name mentioned with -domain.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply_ref> \
    -sink   <sink_supply_ref> \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

Source–Sink with Elements

With -source/-sink/-elements usage, only those pins/ports (mentioned in the -elements option) whose source/sink criteria is satisfied, are isolated.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply_ref> \
    -sink <sink_supply_ref> \
    -elements {elements_list} \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

Isolation Policy with Source only

The `set_isolation` command can be used only with `-source <source_supply_ref>` defined. In this case, all ports of reference domain that are driven by the supply set specified in the `-source` option should be isolated, irrespective of knowledge of sink supply set.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply_ref> \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

Source only with -elements

If `-source` and `-elements` options are used together in the `set_isolation` command, isolation policy applies only to the ports that satisfy both the filters.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply_ref> \
    -elements {elements_list} \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

Isolation Policy with Sink only

The `set_isolation` command can be used only with `-sink <sink_supply_ref>` defined. In this case, all ports of reference domain for which the sink (receiver logic) is supplied by the supply set specified in the `-sink` switch should be isolated, irrespective of knowledge of source supply set.

```
set_isolation <isolation_name> -domain <domain_name> \
    -sink <sink_supply_ref> \
    -location <self/parent/automatic> \
    -clamp_value <1/0/z/latch>
```

Sink only with -elements

If `-sink` and `-elements` options are used together in the `set_isolation` command, isolation policy is applied to the ports that satisfy both the filters.

```
set_isolation <isolation_name> -domain <domain_name> \
    -sink <sink_supply_ref> \
    -elements {elements_list} \
    -location <self/parent/automatic> \
    -clamp_value <1/0/z/latch>
```

Source–Sink with -no_isolation

No isolation is applied on the elements mentioned in the `-elements` option when `-no_isolation` is used in the `set_isolation` command. Generally, this policy is used in conjunction with other isolation policy having same `reference_domain` and `-source/-sink` filters. Specific elements listed in `-elements` are not isolated.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply> -sink<sink_supply> \
    -no_isolation \
    -elements {elements_list} \

```

```
-location <self/parent/automatic> \
-clamp_value <1/0/Z/latch>
```

-diff_supply_only TRUE

The `-diff_supply_only` option determines the isolation behavior between driver and receiver supply sets. It applies isolation if the supplies of driver and receiver are different. Default value of `diff_supply_only` is FALSE.

```
set_isolation <isolation_name> -domain <domain_name> \
    -diff_supply_only TRUE \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

VCS NLP also supports conservative `diff_supply_only`. For more information, see “[Support for Conservative diff_supply_only Isolation](#)” .

Source–Sink with -diff_supply_only

As per the IEEE 1801 standard, an error message is issued if the `-diff_supply_only`, `-source`, or `-sink` arguments are used together. VCS NLP supports using `-source`, `-sink`, and `-diff_supply_only` options together. In this case, isolation policy must be applied to the ports from source to sink domain, if both source and sink supplies are different.

```
set_isolation <isolation_name> -domain <domain_name> \
    -source <source_supply> -sink <sink_supply> \
    -diff_supply_only TRUE \
    -location <self/parent/automatic> \
    -clamp_value <1/0/Z/latch>
```

-diff_supply_only with -no_isolation

No isolation is applied on the elements mentioned in the `-elements` option when `-no_isolation` is used in the `set_isolation` command. Generally, this policy is used in conjunction with other isolation policy having same `reference_domain` and `-diff_supply_only TRUE` filters. Specific elements listed in `-elements` are not isolated. Other elements fulfilling the `-diff_supply_only TRUE` condition are isolated.

```
set_isolation <isolation_name> -domain <domain_name> \
    -diff_supply_only TRUE  \\
    -location <self/parent/automatic> \
    -no_isolation \
    -elements {elements_list} \
    -clamp_value <1/0/z/latch>
```

Precedence Details

Precedence among various options is listed below. Strategies that are applied directly on pins/ports have highest precedence, followed by the strategies defined for the whole domain (with no `-elements`). The general rule is that specific strategies have higher precedence over general strategies. The `-source/-sink/-diff_supply_only` rule has higher precedence than the `-applies_to` rule. The effective precedence is as follows (precedence decreases from top to bottom):

- Domain level strategy matching `-source && -sink && -diff_supply_only && -no_isolation`
- Domain level strategy matching `-source && -sink && -diff_supply_only`

- Domain level strategy matching `-source && -sink && -no_isolation`
- Domain level strategy matching `-source && -sink`
- Domain level strategy matching `-sink && -no_isolation`
- Domain level strategy matching `-source && -no_isolation`
- Domain level strategy matching `-sink`
- Domain level strategy matching `-source`
- Domain level strategy matching `-diff_supply_only && -no_isolation`
- Domain level strategy matching `-diff_supply_only`

Note:

- If multiple isolation strategies refer to a single port, a warning message is issued to indicate that multiple isolation strategies are applied. It also indicates the policy that is applied to the port out of multiple strategies.
- Top domain boundary ports are not considered as source or sink. If you want to use top domain boundary port as source or sink, you must use the `set_port_attributes` command.
- By default, Verilog continuous assignment statements are considered as feedthrough in VCS NLP. Therefore, continuous assignment statements or buffers (Verilog) in the design are not treated as driver or load.
- If an isolation policy refers to any specific member of SystemVerilog union, then it applies to all the members of the union.

- VCS NLP supports using `-diff_supply_only`, `-source`, or `-sink` together on the command line.
-

Defining Isolation Strategies

Isolation strategy can be defined using both `set_isolation` and `set_isolation_control` commands or using the `set_isolation` command only (as per the IEEE 1801-2013).

VCS NLP supports the `-isolation_signal`, `-isolation_sense`, and `-location` options in the `set_isolation` command. Following is the syntax:

```
set_isolation isolation_name
...
[-location <self | parent | automatic |>]
[-isolation_signal signal_list [-isolation_sense {<high | low>}]]
```

For every isolation strategy created, you can specify the above options with the `set_isolation` command. There is no need to define `set_isolation_control` for such isolation strategy.

If you use any of the `-isolation_signal`, `-isolation_sense`, and `-location` options in the `set_isolation` command, then the `set_isolation_control` command is ignored.

Error Checking for Isolation Strategy with -location parent on Top-level Ports

When an isolation strategy applies to a top-level port, the isolation strategy can be implemented only in the “self” location that is visible in the design scope.

VCS NLP generates an error message when an isolation strategy with location “parent” is specified on the top-level ports of the design.

Example

```
set_design_top tb/dut
create_power_domain PD_TOP -include_scope
..
set_isolation PD_ISO -domain PD_TOP
set_isolation_control PD_ISO -domain PD_TOP -
isolation_signal iso_sig -location parent
```

Error Message

```
Error-[UPF_IOPTP] Isolation on the parent of power top
TOP.upf, 48
Isolation policy 'PD_ISO' of power domain 'PD_TOP' is
specified on power top instance 'tb/dut'. Isolation policy
with location 'parent' cannot be specified on the parent of
power top instance.
Please consider specifying the policy with location 'self'.
```

Using set_repeater Command

The `set_repeater` command defines a strategy for inserting repeater cells (buffers) for ports on the interface of a power domain. Repeater cells are placed within the domain, driven by input ports of the domain, and driving the output ports of the domain.

VCS NLP supports the `set_repeater` command as per the *IEEE 1801-2015 UPF LRM*. This command allows you to insert buffers inside the specified domain. This helps you to insert a buffer with a specific supply on the required side of the domain boundary pins.

Following is the syntax of the `set_repeater` command:

```
set_repeater strategy_name
```

```
-domain domain_name
[-elements element_list]
[-exclude_elements exclude_list]
[-applies_to <inputs | outputs | both>]
-repeater_supply supply_set_ref
[-name_prefix string] [-name_suffix string]
[-update]
```

-domain

Specifies the domain name for which the repeater strategy applies. Both simple and rooted names are supported.

-elements

Explicitly identifies a set of candidate ports to which this strategy potentially applies. The `element_list` may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the `element_list`, it is equivalent to specifying all the ports of the instance in the `element_list`.

If `-elements` is not specified in the base command or any update, every port on the interface of the domain is included in the `aggregate_element_list`.

By default, only upper boundary of the specified domain is considered while computing `aggregate_element_list`. Lower boundary elements are considered only when the `lower_domain_boundary` design attribute is set for the design.

The `set_repeater` command will not apply to the ports of direction `inout`. These ports are not included in the `aggregate_element_list`.

-exclude_elements

Explicitly identifies a set of ports to which this strategy does not apply. The `exclude_list` may contain rooted names of instances or ports in the specified domain.

If an instance name is specified in the `exclude_list`, it is equivalent to specifying all the ports of the instance in the `exclude_list`. Any `exclude_lists` specified on the base command or any updates of the base command are combined into the `aggregate_exclude_list`.

The `exclude_list` can specify instances or ports that have not already been explicitly or implicitly specified through an explicit or implicit `element_list`.

-applies_to

VCS NLP follows the following rules while filtering the elements based on -applies_to:

- This option is interpreted relative to the domain for which the strategy is being defined (also known as reference domain).
- Filtering of HighConn elements is done only when lower domain boundary feature is enabled.
- For ports/pins whose LowConn is on the interface of the reference domain, the port/pin is selected when the direction of the port/pin matches the direction specified by -applies_to.
- For ports/pins whose HighConn is on the interface of the reference domain, the port/pin is selected when the inverse of the direction of the port matches the direction specified by -applies_to.

- In the absence of `-applies_to`, be it domain level or element level strategy, no default is assumed, hence no filtering is done.
 - When `-elements` has pins/ports, strategy is applied to all the specified pin/port elements.
 - When `-elements` has instances, strategy is applicable to all the pins of the instance elements.
 - When `-elements` is not specified, strategy is applicable to all the domain boundary pins/ports.

-repeater_supply

Specifies the supply set that powers the inserted buffer. This option supports both simple and rooted name of a supply set. You must use the `-repeater_supply` option with the `set_repeater` command. VCS NLP issues an error message if `-repeater_supply` is not specified.

-name_prefix and -name_suffix

The `-name_prefix` option specifies the substring to place at the beginning of any generated name implementing the `set_repeater` strategy.

The `-name_suffix` option specifies the substring to place at the end of any generated name implementing the `set_repeater` strategy.

If these options are not specified, a default suffix of `_UPF_RPTR` is used and there will not be any default prefix.

-update

Allows you to update the `-elements` and `-exclude_elements` options. This option adds information to the base command executed in the same scope.

Effective Element List and Precedence Rules

The `effective_element_list` for the `set_repeater` command consists of all the port names located inside or outside the `aggregate_element_list` and the port names that satisfy all of the filters specified in the `set_repeater` command. If a port in the `effective_element_list` is not on the interface of the specified domain, it is not buffered.

If a given domain boundary pin is referenced in the `effective_element_list` of more than one repeater strategy of a given domain, following precedence rules are used to resolve the applicable repeater strategy (listed in the order from highest to lowest):

- Pin or port-level strategy (pin/port specified in the `-elements` option)
- Cell-level strategy (cell specified in the `-elements` option)
- Domain-level strategy (the `-elements` option is not used)

The `-applies_to` option does not play any role in deciding precedence of repeater strategies.

Example-1

The following UPF sets repeater strategy R1 on upper boundary ports of root instance inst1 of the domain PD1, and sets repeater strategy R2 on only some specific pins of root instance inst2 by dropping inst2/inout from the element list.

```
create_power_domain PD1 -elements {inst1 inst2}
set_repeater R1 -domain PD1 -elements {inst1} -
repeater_supply SS1
set_repeater R2 -domain PD1 -elements {inst2/in1 inst2/out1
inst2/inout} -repeater_supply SS1
```

Example-2

In the following UPF, repeater strategy R1 takes precedence over R2 for output pins of PD1_inst, since R1 is an element based strategy. For input pins of PD1_inst, R2 applies.

```
create_power_domain PD1 -elements {PD1_inst}
set_repeater R1 -domain PD1 -elements {PD1_inst} -
repeater_supply SS1 -applies_to outputs
set_repeater R2 -domain PD1 -repeater_supply SS2
```

Example-3

In the following UPF, repeater strategy R2 errors out since it is of the same precedence as R1.

```
create_power_domain PD1 -elements {PD1_inst}
set_repeater R1 -domain PD1 -repeater_supply SS1 -applies_to
outputs
set_repeater R2 -domain PD1 -repeater_supply SS2
```

Supported Insertion Semantics

Insertion of repeater cells happen before inserting isolation and level-shifter cells. Therefore, repeater cells impact the evaluation of source/sink filters of isolation and level-shifter strategies.

You cannot place a cell between a port and a pad physically. Therefore, repeater cells are not inserted if you specify repeater strategy on the port which is connected to a pad pin of a pad cell.

Naming of Repeater

The repeater cells that are inserted as a result of `set_repeater` strategy are named using the prefix and suffix specified by the `-name_prefix` and `-name_suffix` options of the command. If these options are not specified, a default suffix of `_UPF_RPTR` is used to name the repeater.

The repeater cells are named as per the following naming convention:

```
[<name_prefix>]snps_<power_domain_name>__<repeater_strategy_name>_snps_<pin_name>[<_inst_index>] [<name_suffix>]
```

The following table shows the example naming of repeater inserted for a pin `p1` that belongs to the power domain `PD` and repeater strategy `R1`:

Prefix	Suffix	Repeater name
""	""	snps_PD__R1_snps_p1
PFX_	_SFX	PFX_snps_PD__R1_snps_p1_SFX
Not specified	_SFX	snps_PD__R1_snps_p1_SFX
PFX_	Not specified	PFX_snps_PD__R1_snps_p1_UPF_RPTR
Not specified	Not specified	snps_PD__R1_snps_p1_UPF_RPTR
PFX	SFX	PFXsnps_PD__R1_snps_p1SFX

Interaction with set_port_attributes -repeater_supply

You can use both the `set_port_attribute -repeater_supply` and `set_repeater` commands in the same UPF. However, you must specify these commands on different ports/pins of a feedthrough path as shown in the following UPF:

```
set_port_attributes -repeater_supply {SS1} -ports  
{U1/input1}
```

```
set_repeater -domain U1_PD -elements {U1} -  
applies_to outputs -repeater_supply {SS2}
```

VCS NLP issues an error message if you apply both `set_port_attribute -repeater_supply` and `set_repeater` commands on the same domain boundary port or hierarchical pin.

Error Scenarios

VCS NLP issues error message in the following cases:

- When `-repeater_supply` option is not specified.
- When an attempt is made to set the repeater strategy on the bidirectional ports/pins of the domain.
- When multiple repeater strategies apply on the same object.
- When the same element has both repeater strategy and `set_port_attributes -repeater_supply` or `set_port_attributes -attribute repeater_power_net|repeater_ground_net`

Limitations

Following are the limitations of this feature:

- The `set_repeater` command does not support the `-source`, `-sink`, `-use_equivalence`, `-applies_to_boundary`, and `-instance` options.
- You cannot apply multiple repeater strategies on the same object.
- Back-to-back isolation and repeater in the same scope is not supported.
- Repeater insertion on pad pins is not supported.
- Precedence rules are not considered for naming of repeaters if multiple `set_repeater` strategies are specified.

Using `set_related_supply_net` Command

The `set_related_supply_net` (SRSN) is a non-UPF command. You can use this command to associate an external supply net to the design ports.

This command is mainly used in hierarchical UPF flow to specify always-on paths. It works on power domain instance boundary ports, PG cell pins, and the design top boundary ports. The design top is the top specified using the `-power_top` compile-time option or the `set_design_top` UPF command.

Following is the use model of the `set_related_supply_net` command:

```
set_related_supply_net [-object_list objects]
[-ground ground_net_name]
[-power power_net_name]
```

You must use either `-power` or `-ground`. If `-power` or `-ground` is not specified, corresponding supply is treated as ON.

If you do not specify `-object_list`, all the ports of design top specified by `set_design_top` or `-power_top` are taken as the default object list. `objects` is the hierarchical reference to the logic ports relative to the current scope.

Example

The following example demonstrates the usage of the `set_related_supply_net` command:

```
create_power_domain TOP
create_power_domain A -elements {instA1}

create_supply_port VDD
create_supply_port VDDA
create_supply_port VSS

create_supply_net VDD -domain TOP
create_supply_net VDDA -domain A
create_supply_net VSS -domain TOP
create_supply_net VSS -domain A -reuse

connect_supply_net VDD -ports VDD
connect_supply_net VDDA -ports VDDA

set_domain_supply_net TOP -primary_power_net VDD -
primary_ground_net VSS
set_domain_supply_net A -primary_power_net VDDA -
primary_ground_net VSS

set_related_supply_net -object_list {instA1/out1} -power VDD -
ground VSS
```

In the above example UPF, consider that the power domain A is ON. If VDD is switched off, then `out1` of `instA1` corrupts according to the above specified `set_related_supply_net` command. However, other ports remain ON as the power domain A of `instA1` is ON.

Limitations

Following are the limitations of the SRSN command:

- SRSN on real array with more than one unpacked dimension is not supported.
- SRSN on Wreal `inout` port is not supported.

Using `set_port_attributes` Command

The `set_port_attributes` (SPA) command allows you to specify the boundary conditions of the design under test. During block level verification, this command allows you to specify the power information about the logic driving the primary input ports and logic driven by the primary output ports of the design.

The following sections describe the `set_port_attributes` support in VCS NLP:

- [Use Model](#)
- [Comparison with `set_related_supply_net` \(SRSN\)](#)
- [`set_port_attributes` on Non-Boundary Ports](#)
- [Ignoring SRSN/SPA Buffers Specified on Intermediate Boundary Ports](#)

- Using the `-repeater_supply` Option
 - Support for Unconnected Ports Inside Macro Blocks
 - Inserting SPA and SRSN Buffers on HighConn
-

Use Model

Following is the syntax of the `set_port_attributes` command:

```
set_port_attributes
  [-model name]
  [-elements element_list]
  [-exclude_elements element_exclude_list]1
  [-ports port_list]
  [-exclude_ports port_exclude_list]1
  [-applies_to <inputs | outputs | both>]
  [-attribute {name value}]*
  [-clamp_value <0 | 1 | any | Z | latch | value1>]
  [-sink_off_clamp <0 | 1 | any | Z | latch | value>]1
  [-source_off_clamp <0 | 1 | any | Z | latch | value>]1
  [-driver_supply supply_set_ref]
  [-receiver_supply supply_set_ref]
  [-pg_type pg_type_value]
  [-related_power_port supply_port_name]
  [-related_ground_port supply_port_name]
  [-related_bias_ports supply_port_name_list]
  [-feedthrough]
  [-unconnected]
```

* Indicates that the item in square braces can be repeated in the command.

¹This command argument is parsed and ignored by VCS NLP.

You can use the `set_port_attributes` command to refer only the ports specified in the design.

Supported attributes with -attribute are iso_source, iso_sink, and related_supply_default_primary.

Options

- `-driver_supply/-receiver_supply`

When `-driver_supply` is attributed on a port, it specifies the supply of the logic driving the port. The port is corrupted when the driver supply is in a simstate other than NORMAL. Any isolation strategy with the driver supply as `-source` applies on the port as long as all other filtering criteria on strategy are met. That is, VCS NLP uses the `-driver_supply` for strategy association for the port.

Similarly, when `-receiver_supply` is attributed on a port, it specifies the supply of the logic reading the port. The port is corrupted when the receiver supply is in a simstate other than NORMAL. Any isolation strategy with the receiver supply as `-sink` applies on the port as long as all other filtering criteria on strategy are met. That is, VCS NLP uses `-receiver_supply` for strategy association for the port.

Attributes

- `iso_source/iso_sink`

The `iso_sink` attribute indicates the supply set that drives the eventual sink (or receiving logic) of a primary output port. The `iso_source` attribute indicates the supply set that drives the source (or driving logic) of a primary input port.

The supply set specified with this attribute is used by VCS NLP for strategy association. It is not used for determining the corruption on the port. If a port contains both `iso_source` and `-driver_supply`, `iso_source` is used for strategy association and `-driver_supply` is used for corruption. Similarly, for `iso_sink` and `-receiver_supply`, `iso_sink` is used for strategy association and `-receiver_supply` is used for corruption.

- `related_supply_default_primary`

This attribute indicates that the driver/receiver supply for a particular port is same as the primary power of its corresponding power domain. Note that this attribute applies only if there is no `set_port_attributes` or `set_related_supply_net` command on the port.

Limitations

Following are the limitations of the SPA command:

- SPA on real array with more than one unpacked dimension is not supported.
- SPA on Wreal `inout` port is not supported.

Comparison with `set_related_supply_net` (SRSN)

The `set_related_supply_net` command continues to have the same corruption and strategy association semantics as before. However, if there is a `set_port_attributes` (SPA) command on the same port, the `set_port_attributes` command might override the `set_related_supply_net` specification. The priority (decreasing order) in which all these attributes apply are as follows:

- Corruption semantics

```
driver/receiver_supply > set_related_supply_net >
related_supply_default_primary
```

- Strategy association

```
iso_source/iso_sink > driver_supply/receiver_supply >
set_related_supply_net > related_supply_default_primary
```

Note:

- The SPA command can be applied on all ports of an instance using the `-elements` option. If a particular port figures in both `-elements` list and `-ports` list, the attribute with `-ports` list is applied.
- If the SPA command is applied on whole vector and also on individual bits of a vector, then SPA applied on whole vector gets higher priority which is against LRM. As per LRM, precedence of SPA is higher for the elements having more granularity, but this case is an exception.

set_port_attributes on Non-Boundary Ports

VCS NLP applies corruption semantics, but ignores `set_port_attributes` for policy association. Instead, it uses the actual source/sink logic in the design.

Note:

- Applying both `set_port_attributes` and `set_related_supply_net` on the same port is not recommended.
- Wildcards are not supported with the `set_port_attributes` command.

Ignoring SRSN/SPA Buffers Specified on Intermediate Boundary Ports

By default, VCS NLP considers SRSN/SPA buffers specified on all boundary ports.

You can use the `-power=ignore_redundant_constraint` compile time option to ignore SRSN/SPA buffers specified on all intermediate boundary ports.

With this option, SRSN/SPA constraints are honored only on power-top ports, DB cell, and black-box boundary ports.

Using the `-repeater_supply` Option

VCS NLP supports the `-repeater_supply` option with the `set_port_attributes` UPF command which has the semantic to force a repeater at the port having the repeater supply attribute in addition to specifying the supply to be used by that repeater.

The following syntax describes the usage of this option:

```
set_port_attributes
  [-ports {port_list}]
  [{-elements {element_list} [-applies_to <inputs | outputs
    | both>]}]
  [-repeater_supply supply_set_ref]
```

Where, `supply_set_ref` is the supply set used by a repeater driving the port.

You can also use the following attributes to specify ground and power net, instead of specifying supply set.

```
set_port_attributes -attribute repeater_power_net  
<supply_net_power> -ports ...
```

```
set_port_attributes -attribute repeater_ground_net  
<supply_net_ground> -ports ...
```

The output information is displayed in the following files:

File Name	Description
isolation_association.rpt	Displays information for each boundary port of a node.
repeaters.rpt	Lists all the ports which are treated as having repeaters.
xovers.rpt	Displays information on changes to the crossings/source/sink.

Note:

- If the `-power=verbose1` compile-time option is specified, then VCS NLP generates the `boundary_port_partitioning.rpt` report file. This file displays the repeaters implemented and its associated supplies.
- You cannot apply repeater supply on the top domain input ports and macro output ports.

Limitations

Following are the limitations of this feature:

- Repeater supply on interface/modports is not supported
- `set_port_attributes` is not supported for ‘inout’ ports, `set_related_supply_net` can be used instead

Support for Unconnected Ports Inside Macro Blocks

An unconnected port represents a port on the interface of a module or cell that is not connected to either a source or sink supply.

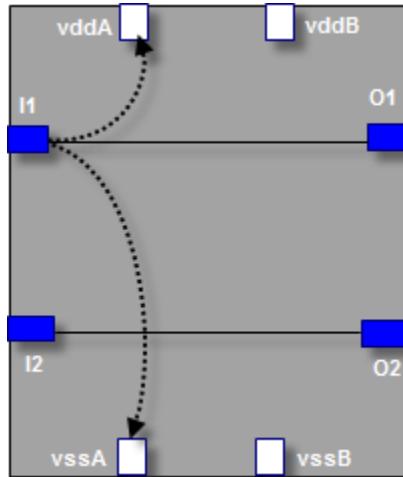
VCS NLP allows you to specify the logic ports of a macro block that are not connected to any supply ports using the `-unconnected` option of the `set_port_attributes` command.

Following is the syntax of the `-unconnected` option:

```
set_port_attributes -model <model> -unconnected -ports  
<port_list>
```

Consider the macro cell in [Figure 3-2](#) with two set of supplies: `vddA / vssA` and `vddB / vssB`.

Figure 3-2 Macro Cell



In [Figure 3-2](#), the input port `I1` drives the logic powered by `vddA / vssA` and connection to port `O1`. The input port `I2` is not connected to any supply port. You can use the following command to attribute port `I2` as unconnected:

```
set_port_attributes -ports {I2} -model cellX -unconnected
```

Key Points to Note

- If `-model` is specified, only ports that belong to the given model can be specified as an argument to the command. Hierarchical names are not supported
- UPF specified unconnected port ignores `related_power_pin`, `related_ground_pin`, and `power_down_function` of that port
- You must use `-unconnected` in combination with the `-ports` option
- VCS NLP issues an error message if `-model` is used in combination with the `-elements` option
- `-model` can refer to the PAD or macro library cells and black box elements
- Multiple instances of a command are allowed for the same `-model` reference. Overlapped instances of referenced port with previous settings are parsed and ignored
- At least one element must be specified in `-ports`
- `set_related_supply_net (SRSN)` overrides the `set_port_attributes -unconnected` setting on a port

Limitation

Following is the limitation:

- Wildcards are not supported with the `-ports` option.

Specifying Clamp Value for Ports

VCS NLP supports specifying clamp value for ports. Following is the syntax for specifying clamp value for ports:

```
set_port_attributes [-clamp_value <0 | 1 | any | Z  
| latch | value>]
```

Where, `-clamp_value` specifies the clamp value to be used for the port with isolation strategy applied to it.

If different clamp values are specified on the same port using the `set_isolation` and `set_port_attributes` commands, then VCS NLP issues the UPF_SPACVM warning message, and considers the clamp value provided by the isolation policy.

Example

Consider the following UPF code:

```
set_isolation iso2 -domain TOP -isolation_supply_set SSTOP  
-clamp_value 1 -elements { dataout1 dataout2 dataout3 }

set_isolation_control iso2 -domain TOP -isolation_signal  
iso_enable -isolation_sense high -location self

set_port_attributes -clamp_value 0 -ports {dataout1} -  
attribute iso_sink SSTOP
```

Here, `set_isolation` and `set_port_attributes` are on the same port `dataout1`. The clamp value from the isolation policy is 1, and from `set_port_attributes` is 0. Therefore, VCS NLP issues the UPF_SPACVM warning message and considers the clamp value from the isolation strategy. Following is the warning message:

```
Warning-[UPF_SPACVM] Clamp Value Mismatch  
sanity_test_port_base.upf, 30
```

Clamp value '1' specified in isolation strategy 'topunit/TOP_U/TOP/isolation/iso2' for design port 'topunit.TOP_U.dataout1' does not match with the value '0' specified with 'set_port_attributes -clamp_value'. Clamp value from isolation strategy is considered. Isolation Strategy: 'sanity_test_port_base.upf, 41'.

Inserting SPA and SRSN Buffers on HighConn

VCS NLP inserts the `set_port_attributes` (with the `-driver_supply` and `-receiver_supply` attributes) and `set_related_supply_net buffers` on HighConn for the regular models and macros. [Table 3-8](#) and [Table 3-9](#) describes the placement of SRSN and SPA buffers.

Table 3-8 Placement of SRSN Buffers

SRSN	Ports	Location
Hierarchical Models	Input, Output, and Inout ports	HighConn
Black box Models	Input, Output, and Inout ports	LowConn
Top Power Domain Boundary	Input, Output, and Inout ports	LowConn

Table 3-9 Placement of SPA Buffers

SPA	Ports	Location
Hierarchical Models	Input ports with <code>-driver_supply</code> Output ports with <code>-receiver_supply</code>	HighConn
Black box Models	Input ports with <code>-receiver_supply</code> Output ports with <code>-driver_supply</code>	LowConn
Top power domain boundary	Input, output	LowConn

Precedence Rules for SRSN and SPA

If SRSN and SPA are specified on the same port, then SPA takes precedence over SRSN. Also, policy that is specified at more granular level takes precedence over the policy specified at less granular level. That is, if the policy P1 is specified on a scope, policy P2 is specified on a port, and policy P3 is specified on a bit of the port, then the precedence order is P3 > P2 > P1.

Defining Retention Strategies

Retention strategies can be defined using the combination of the `set_retention` and `set_retention_control` commands or using the `set_retention` command only (as per the IEEE 1801-2013).

VCS NLP supports the `-save_signal` and `-restore_signal` options in the `set_retention` command. Following is the syntax:

```
set_retention retention_name
...
[-save_signal {logic_net <high | low | posedge | negedge>}
 -restore_signal {logic_net <high | low | posedge | negedge>}]
```

If you use any of the `-save_signal` and `-restore_signal` options in the `set_retention` command, then the `set_retention_control` command is ignored.

Note:

The following mutex options are supported only with the `set_retention_control` command:

```
[-assert_r_mutex {{net_name <high | low | posedge | negedge>}}]*
```

```
[-assert_s_mutex {{net_name <high | low | posedge | negedge>}}]*  
[-assert_rs_mutex {{net_name <high | low | posedge | negedge>}}]*
```

Including Current Scope in the Power Domain

VCS NLP allows you to use the `-elements { . }` option in the `create_power_domain` command to include the current scope in the power domain.

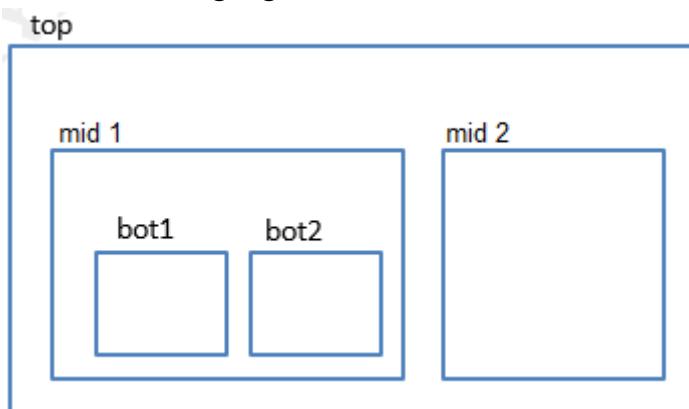
For example, if the current scope is set to instance top, the `create_power_domain PD -elements { . }` command includes the current scope (top) and all of its descendants in the power domain PD.

Note:

VCS NLP issues an error message if you use `-scope { . }` in the `create_power_domain` command.

Examples

Consider the following figure:



Case-1: `create_power_domain PD -include_scope`

Case-2: `create_power_domain PD -elements { . }`

Case-1 and **case-2** are equivalent. They both create power domain on top.

Case-3:

```
set_scope mid1  
create_power_domain PD -elements { . }
```

Case-4:

```
create_power_domain PD -elements {mid1} -scope mid1
```

Case-3 and **case-4** are equivalent. They both create the power domain on mid1.

Case-5:

```
create_power_domain PD_mid -elements mid1  
create_power_domain PD_top -elements { . mid1/  
bot1 }
```

In case-5, the power domain `PD_top` is on both top scope and mid1/bot1.

Supply Equivalence Checks by VCS NLP

The `set_equivalent` command allows you to declare two supply nets (or supply sets) as electrically or functionally equivalent. Electrical equivalence is primarily related to supply ports and nets, whereas functional equivalence is primarily related to supply sets.

VCS NLP performs semantic checks for the following cases:

- Functionally equivalent supply sources with conflicting port states
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources with Conflicting Port States](#)” .
- Functionally equivalent supply sources with missing port states
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources with Missing Port States](#)” .
- Functionally equivalent supply sources having conflicting type
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources Having Conflicting Type](#)” .
- PST entry with conflicting port states
This check is performed for both functionally and electrically equivalent supply sources. For more information, see “[PST Entry with Conflicting Port States](#)” .
- Supply source state does not match with the state of equivalent supply source

This check is performed for the functionally equivalent supply sources. For more information, see “[Supply Source State Does Not Match with the State of Equivalent Supply Source](#)” .

Functionally Equivalent Supply Sources with Conflicting Port States

Functionally equivalent supply sources should not have different voltages defined for same state name.

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test1.upf` file where VDD1 and VDD2 are functionally equivalent, but the state named ON have different voltage values.

Example 3-10 test1.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON 1.0}
add_port_state VDD2 -state {ON 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

VCS NLP issues the UPF_FEWCP warning message for this case. Following is the sample warning message:

```
Warning-[UPF_FEWCP] Functional equivalent supply with
conflicting port states
```

```
Port state 'ON' with voltage value '0.8' defined for Supply Net/Supply Port 'tb/top/VDD2' has a conflicting definition with different voltage value '1' defined for Supply Net/Supply Port 'tb/top/VDD1' at (test1.upf, 23).
```

Functionally Equivalent Supply Sources with Missing Port States

Functionally equivalent supply sources should have the same number of states defined.

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test2.upf` file where VDD1 and VDD2 are functionally equivalent, but have different number of states defined where VDD1 has 1 state while VDD2 has 2 states.

Example 3-11 test2.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON 0.8}
add_port_state VDD2 -state {ON 0.8}
add_port_state VDD2 -state {ON2 1.0}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

In the above case, VCS NLP issues the `UPF_FEWMPS` warning message for missing state `ON2` for supply VDD1.

Following is the sample warning message:

```
Warning-[UPF_FEWMPS] Functional equivalent supply with
missing port states
State 'ON2' missing for Supply Net/Supply Port 'tb/top/VDD1'
has been defined for functional equivalent Supply Net/Supply
Port 'tb/top/VDD2' with voltage value '1' at (test2.upf, 23).
```

Functionally Equivalent Supply Sources Having Conflicting Type

Functionally equivalent supply sources should not have different supply type (that is, power/ground).

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test3.upf` file where `VDD1` and `VSS` are functionally equivalent, but have different supply type (that is, power/ground).

Example 3-12 test3.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VSS
create_supply_net VSS_net
connect_supply_net VSS_net -ports VSS

add_port_state VDD1 -state {ON1 1.0}
add_port_state VSS -state {ON 0.0}

set_equivalent -nets {VDD1_net VSS_net} -function_only

create_power_domain -include_scope -primary_power_net
VDD1_net -primary_ground_net VSS_net
```

In the above case, VCS NLP issues the `UPF_FESCT` error message. Following is the sample error message:

```
Error-[UPF_FESCT] Functional equivalent supply have
conflicting type
electrical_equivalent_nets.upf, 10
Supply Net/Supply Port 'tb/top/VDD1' used as 'power' has
functional equivalent Supply Net/Supply Port 'tb/top/VSS'
used as 'ground'.
```

PST Entry with Conflicting Port States

Functionally or electrically equivalent supply sources should have the same state in a PST state.

This check is applicable for both electrical and functional equivalent supplies. For functional supplies, the check is enabled using the `-power=hier_pst_check` compile-time option.

Consider the following `test4.upf` where VDD1 and VDD2 are functionally equivalent. The PST state have the same supply source, but different port states with different supply voltage.

Example 3-13 test4.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {OFF off}
add_port_state VDD2 -state {OFF off}

add_port_state VDD1 -state {ON1 1.0}
add_port_state VDD2 -state {ON2 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only

create_pst top_pst -supplies {VDD1 VDD2}
```

```
add_pst_state ON -pst top_pst -states {ON1 ON2}
add_pst_state OFF -pst top_pst -states {OFF OFF}
```

In the above case, VCS NLP issues the UPF_PST_ESCPS warning message. Following is the sample warning message:

```
Warning-[UPF_PST_ESCPS] Pst entry with conflicting port states check_pst_state_scl.upf, 35
```

```
State ON of tb/top/top_pst refers to different voltage states for equivalent supplies. Port/power state 'ON1' (1) defined for supply VDD1 has different voltage from state 'ON2' (0.8) defined for supply VDD2.
```

Supply Source State Does Not Match with the State of Equivalent Supply Source

Functionally equivalent supply sources should have the same voltage in testbench.

Consider the following test5.upf and testbench files where VDD1 and VDD2 are functionally equivalent, but have different voltages at specified in testbench for the same timestamp.

Example 3-14 test5.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON1 1.0}
add_port_state VDD2 -state {ON2 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

Example 3-15 Testbench

```
Testbench:  
#10  
supply_on(VDD1 1.0)  
supply_on(VDD2, 0.8)
```

At the timestamp #10, the voltage value of the equivalent supply pads have different voltage set. You must perform this check at runtime.

The LP_EQUIVALENT_SUPPLY_MISMATCH low power assertion is flagged by default for functionally equivalent supply sources with conflicting voltages/states at a given timestamp.

In the above case, VCS NLP issues the LP_EQUIVALENT_SUPPLY_MISMATCH error message at runtime. Following is the sample output in simulation log:

```
[10 ns] [INFO] [LP_SUPPLY_PAD_CALL] Supply Pad function  
'UPF::supply_on' called for Supply Pad 'VDD1'. Supply Pad  
value changed to '{FULL_ON, 1000000}'. File: tb.v, Line:  
21.
```

```
[10 ns] [INFO] [LP_SUPPLY_PAD_CALL] Supply Pad function  
'UPF::supply_on' called for Supply Pad 'VDD2'. Supply Pad  
value changed to '{FULL_ON, 800000}'. File: tb.v, Line: 22.
```

```
[10 ns] [ERROR] [LP_EQUIVALENT_SUPPLY_MISMATCH] Supply  
source 'tb/top/VDD2' state {FULL_ON, 0.80 V} does not match  
with state {FULL_ON, 1.00 V} of equivalent supply source  
'tb/top/VDD1'
```

The default severity of the LP_EQUIVALENT_SUPPLY_MISMATCH message is ERROR. You can change the severity using the set_lp_msg_severity option in the configuration file during runtime.

Using upf_version Command

VCS NLP supports the `upf_version` UPF command. Support for `upf_version` is limited to parse and ignore only. No version compatibility checks or semantics are applied for UPF commands based on it.

Syntax

```
upf_version [string]
```

Irrespective of the version string, the return value of the `upf_version` command is IEEE-1801.

Using upf_tool Variable

VCS NLP supports the usage of Tcl variable `upf_tool` in the UPF file. For VCS NLP, the value of `upf_tool` is internally set to `MVSIM_NATIVE`, whereas for MVTOOLS, it is set to `MVSIM`. You can use this variable to have two different UPF commands for VCS NLP or MVSIM in the same UPF file.

Usage Example

The following is the usage example:

```
if {[info exists upf_tool]} {  
  
    if {$upf_tool == "MVSIM_NATIVE"} {  
  
        set_retention V1_RETENTION_CLK_FREE \  
                      -domain Island_V1 \  
                      -retention_power_net V1_ret \  
                      -restore_condition {{!reset}} \  
                      -save_signal {save posedge} \  
                      -restore_signal {save high} \  
                      -retention_condition {save}  
  
    } elseif {$upf_tool eq "MVSIM"} {  
  
        set_retention V1_RETENTION_CLK_FREE -domain Island_V1 -  
                    retention_power_net V1_ret  
        set_retention_control V1_RETENTION_CLK_FREE -domain  
        Island_V1 -save_signal {save posedge} -restore_signal {save  
        high}  
  
    }  
}
```

You can also use this variable to add or skip UPF commands for VCS NLP or MVSIM.

```
if {$upf_tool == "MVSIM_NATIVE"} {  
    set_design_attributes ....  
}  
  
if {$upf_tool ne "MVSIM_NATIVE"} {  
    set_related_supply_net ...  
}
```

Refining Power Domain Elements

VCS NLP supports refinement of the power domain elements using the `create_power_domain` command. VCS NLP supports the following way to refine power domain elements:

Power Domain With Empty Elements List

VCS NLP supports power domain created with empty elements list and updated with elements as follows:

```
create_power_domain d1 -elements {}
create_power_domain d2 -elements {}
create_power_domain d1 -update -elements e1
create_power_domain d2 -update -elements e2
```

Support for Negative Voltages

VCS NLP allows you to specify negative voltages in the UPF using the `add_port_state`/`add_power_state` commands. VCS NLP uses the negative voltages in PST state evaluation.

Key Points to Note

- Negative voltages can only be used in the following commands:
 - `create_supply_net`
 - `connect_supply_net`
 - `set_port_attributes -driver_supply/receiver_supply`

- set_related_supply_net
- add_port_state
- add_power_state
- You must make sure to add -no_isolation strategy on the signal paths with negative-related supplies (Isolation strategy on a negative voltage results in isolation insertion).
- You must make sure to apply don't touch (UPF_dont_touch) to prevent buffering on signal paths with negative related supplies.
- Static-level shifting and isolation checks support negative voltages.

Example

[Example 3-16](#) shows negative voltage applied for supply nets in the -driver_supply and -receiver_supply options of the set_port_attributes command. The negative voltage is specified using the create_pst command.

Example 3-16 test.upf

```

create_supply_net VDD
create_supply_net VSS
create_supply_port VDD
create_supply_port VSS

connect_supply_net VDD      -ports VDD
connect_supply_net VSS      -ports VSS

add_port_state VDD -state {VDD_12      -1.2}
add_port_state VSS -state   {VSS_00      0.0}

create_pst top_pst -supplies {VDD VSS      VDD1 VDD2}
add_pst_state s1 -pst top_pst -state {VDD_12  VSS_00 VDD1_12

```

```

VDD2_10}

create_supply_set SS_TOP -function {power VDD} -function
{ground VSS}

set_port_attributes -ports {reset} -driver_supply ss_TOP -
receiver_supply ss_TOP

```

Using **synopsys_program_name** Variable

VCS NLP supports the usage of Tcl variable

`$synopsys_program_name` in the UPF file. For VCS NLP, the value of `$synopsys_program_name` is internally set to `vcs`. You can use this variable for conditional compilation in UPF similar to `$upf_tool`.

Usage Example

The following is the usage example:

```

if {[info exists synopsys_program_name]} {

    if {$synopsys_program_name == "vcs"} {

        set_retention V1_RETENTION_CLK_FREE \
                      -domain Island_V1 \
                      -retention_power_net V1_ret \
                      -restore_condition {{!reset}} \
                      -save_signal {save posedge} \
                      -restore_signal {save high} \
                      -retention_condition {save}

    } elseif {

        set_retention V1_RETENTION_CLK_FREE -domain Island_V1 -
        retention_power_net V1_ret
        set_retention_control V1_RETENTION_CLK_FREE -domain
    }
}

```

```
Island_V1 -save_signal {save posedge} -restore_signal {save
high}

}
}
```

You can also use this variable to add or skip UPF commands for VCS NLP or other tools.

```
if {$synopsys_program_name == "vcs"} {
set_design_attributes ...
}

if {$synopsys_program_name ne "vcs"} {
set_related_supply_net ...
}
```

Handling Conflicting Port State Names in the `add_port_state` Command

You can use the `add_port_state` command to add state information to a supply port.

Syntax

```
add_port_state port_name
{-state {name <nom | min max | min nom max | off>}}
```

The `name` argument defines the name for a state of the supply port.

Two supply ports which are at two different scopes can be connected using a supply net. The supply net serves as a way to propagate voltage between these two supply ports.

You cannot use the same port state name with different voltage values for multiple supply ports tied together. VCS generates the Duplicate Port States error message for such usage. See the following examples for more information.

Example-1: In this example, the name of the supply port state ‘ON’ is used to refer to two voltage values, therefore VCS generates the following error message:

Error-[UPF_CPSE] Conflicting supply states error

top.upf

```
set_design_top TB/DUT
..
add_port_state V1 -state {ON 2.0}
connect_supply_net VT -ports {V1 child/V2}
..
load_upf child.upf -scope child
```

child.upf

```
create_supply_set ..
..
add_port_state V2 -state {ON 4.0}
..
```

Example-2: This example describes the proper usage.

top.upf

```
set_design_top TB/DUT
..
add_port_state V1 -state {ON 2.0}
connect_supply_net VT -ports {V1 child/V2}
..
load_upf child.upf -scope child
```

child.upf

```
create_supply_set ..  
..  
add_port_state v2 -state {ON1 4.0}  
..
```

Example-3: In this example, different port state names are used to refer the same voltage value. Two state names with same voltage value are aliased to one state name, and the following warning message is generated:

Warning-[UPF_MSSV] Multiple states with same voltage values

top.upf

```
set_design_top TB/DUT  
..  
add_port_state v1 -state {ON 2.0}  
connect_supply_net VT -ports {v1 child/v2}  
..  
load_upf child.upf -scope child
```

child.upf

```
create_supply_set ..  
..  
add_port_state v2 -state {ON_1 2.0}
```

Example-4: In this example, same port state names are used to refer to the same voltage value. One of the definitions is considered while the other is ignored, and the following note message is generated:

Note-[UPF_DPSSV] Duplicate supply port state

```
top.upf
```

```
set_design_top TB/DUT
..
add_port_state v1 -state {ON 2.0}
connect_supply_net VT -ports {v1 child/v2}
..
load_upf child.upf -scope child
```

```
child.upf
```

```
create_supply_set ..
..
add_port_state v2 -state {ON 2.0}
```

Setting Supply Port Default Value

This feature allows you to set the default state on the supply port using `set_design_attributes`. Once the default state is applied on a supply port which is a supply pad, it starts from that state.

Use Model

Set the default state for the given port using `set_design_attributes`, as shown below, which is defined in `add_port_state`.

Syntax:

```
add_port_state port_name
{-state {name <nom | min max | min nom max | off>}}
```



```
set_design_attributes -element { <supply_port | supply_net>
} -attribute SNPS_default_supply_net_state
<state_in_add_port_state>
```

Example

Consider the following example code. The supply port V3 starts with its default state ON_D, as defined in set_design_attributes.

```
set_design_top testbench/inst
..
add_port_state V3 -state {ON 4.0} \
                -state {ON1 3.5} \
                -state {ON2 3.0} \
                -state {ON3 2.0} \
                -state {ON_D 2.1}
set_design_attributes -elements { V3 } -attribute
SNPS_default_supply_net_state ON_D
```

Note:

If set_design_attributes is used to define default state of the supply port, and at the same time if supply_on() sets value of that supply port to different value, then supply_on() gets higher priority.

Driving Supply Expression Using Logic Expression

VCS NLP supports the design attribute

SNPS_logic_expr_drives_supply_expr to drive the supply set function which is either undriven or has a single undriven root supply driver (for more information, see “[Root Supply Driver](#)” section) based on the logic expression.

Use Model

Use the set_design_attributes command with the SNPS_logic_expr_drives_supply_expr attribute in the UPF to enable this feature.

Following is the syntax:

```
set_design_attributes
  -elements {<element_list>}
  -attribute SNPS_logic_expr_drives_supply_expr TRUE
```

Key Points to Note

- The `SNPS_logic_expr_drives_supply_expr` attribute is a global attribute. You must specify this attribute for the top design instance. VCS NLP issues the `UPF_ATTR_NOT_FOR_TOP_SCOPE` error message if this attribute is not specified for the top scope.
- If the `SNPS_logic_expr_drives_supply_expr` attribute is specified multiple times with different values for different scopes in the design, then VCS NLP performs a consistency check that all values should match. That is, if this attribute is specified anywhere in the UPF (for any scope), it must also be defined for the top scope with the same value. If the values are inconsistent, VCS NLP issues the `UPF_ATTRS_CONFLICT` error message.

Support for Auto-Completion of UPF Commands

By default, the auto-completion for the UPF commands and command options is disabled.

- The following error message is issued for incomplete UPF command:

```
Error-[UPF_IUC] Invalid UPF command
```

- The following error message is issued for incomplete UPF command option:

```
Error-[UPF_UUCO] Unknown UPF command option
```

You can use the `-power=auto_complete` compile-time option to enable the auto-completion for the UPF commands and command options. It is recommended to correct the UPF command or command option instead of using `-power=auto_complete`.

Support for Feedthrough Paths Inside Macro Blocks

Feedthrough ports are the set of ports on the interface of a module or cell that are directly connected to each other inside a module or cell to create a feedthrough path. In a macro block, a pure feedthrough path is represented by a wire connecting two or more ports of the block (logical feedthrough).

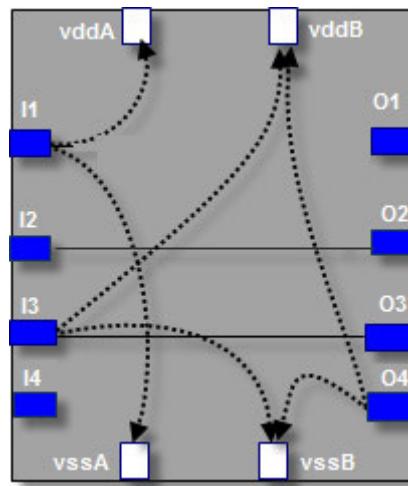
VCS NLP supports creating pure feedthrough paths inside a macro block by allowing you to directly connect set of ports of a model to each other using the `-feedthrough` option of the `set_port_attributes` command.

Following is the syntax of the `-feedthrough` option:

```
set_port_attributes -model <model> -feedthrough -ports  
<port_list>
```

Consider the following macro cell with two set of supplies: `vddA / vssA` and `vddB / vssB`.

Figure 3-3 Macro Cell



In [Figure 3-2](#), I1 drives logic powered by vddA / vssA, I2 is directly connected to port O2, I3 drives logic powered by vddB / vssB and the connection to port O3, port O4 is driven by logic powered by vddB / vssB.

Example:

```
set_port_attributes -ports {I2 O2} -model cellX -feedthrough
```

Key Points to Note

- If **-model** is specified, only ports that belong to the given model can be specified as an argument to the command. Hierarchical names are not supported.
- You must use **-feedthrough** in combination with the **-ports** option. At least two elements must be specified in **-ports**.
- VCS NLP issues an error message if **-model** is used in combination with the **-elements** option.

- `-model` can refer to the PAD or macro library cells, and black box elements.
- Multiple instances of a command are allowed for the same `-model` reference. Overlapped instances of the referenced port with previous settings are parsed and ignored.
- VCS NLP issues an error message if you use the `-unconnected` and `-feedthrough` options on the same `-port` element.
- `set_related_supply_net (SRSN)` overrides the `set_port_attributes -feedthrough` setting on a port.

Limitations

- Wildcards are not supported with the `-ports` option.
- Isolation is not supported for mixed designs.

Generic Handles Support

VCS NLP allows you to use generic handles in the retention strategy and the `bind_checker` command. This section includes the following topics:

- [Generic Handles Support in Retention Strategy](#)
- [Generic Handles Support in the bind_checker Command](#)

Generic Handles Support in Retention Strategy

VCS NLP supports specifying generic names for the clock and asynchronous load as part of `-save_condition`, `-restore_condition`, and `-retention_condition` of the `set_retention` command. You can define the conditions using the generic handles `UPF_GENERIC_CLOCK` and `UPF_GENERIC_ASYNC_LOAD`.

Inference of the generic clock and asynchronous load follow the below guidelines:

1. Coding style used must be valid as per the supported templates for synthesis.
2. Both signals acting as clock and asynchronous load must be in the sensitivity list, and must be edge-sensitive.
3. `UPF_GENERIC_ASYNC_LOAD` is inferred based on the sense specified in the sensitivity list and the conditions specified inside the always block.

Following are the list of scenarios and `GENERIC_CLOCK`/`ASYNC_LOAD` inferred:

Example-1: `GENERIC_CLK`: `clk`, `ASYNC_LOAD`: `~reset`

```
always @ (posedge clk or negedge reset)
    if (~reset)
        dataout1 <= 0;
    else
        dataout1 <= data_in;
```

Example-2: `GENERIC_CLK`: `clk`, `ASYNC_LOAD`: `1'b1`

```
always @ (posedge clk or negedge reset)
```

```

if (reset)
    dataout1 <= 0;
else
    dataout1 <= data_in;

```

Example-3: GENERIC_CLK: clk, ASYNC_LOAD: 1'b1

If any of the expression is not edge sensitive, for example:

```

always @ (posedge clk or reset)
if (reset)
    dataout1 <= 0;
else
    dataout1 <= data_in;

```

Example-4: GENERIC_CLK = 1'b1, ASYNC_LOAD = 1'b1

ASYNC_LOAD is not inferred, for example:

```

always @ (posedge clk or posedge reset or posedge set)
if (reset || set) // or if (reset || 1'b1)
    dataout1 <= 0;
else
    dataout1 <= data_in;

```

Specify generic (canonical) design signal names of clock
 (UPF_GENERIC_CLOCK) and asynchronous load
 (UPF_GENERIC_ASYNC_LOAD) as part of save/restore/retention
 condition of the retention policy, as shown below:

```

set_retention RET1 -domain PD1
-save_signal {save_sig} -restore_signal(restore_sig)
-retention_power_net VDD_RET_NET
-retention_ground_net VSS_RET_NET
-save_condition {! UPF_GENERIC_ASYNC_LOAD}
-retention_condition
{UPF_GENERIC_CLOCK && UPF_GENERIC_ASYNC_LOAD}

```

Note:

- Usage of UPF_GENERIC_ASYNC_LOAD must be limited to the retention strategies that have asynchronous reset flops as elements.
- UPF_GENERIC_ASYNC_LOAD is taken as value 0 for the synchronous reset flops, non-resettable flops, and latches.
- Usage of UPF_GENERIC_CLOCK must be limited to the retention strategies that do not have any latches as elements.
- UPF_GENERIC_CLOCK is taken as value 1 for latches.

Limitations

- Supported for retention in Verilog context only.

Generic Handles Support in the bind_checker Command

This section includes the following topics:

- [Generic Handles Support for Isolation in the bind_checker Command](#)
- [Generic Handles Support for Retention in the bind_checker Command](#)

Generic Handles Support for Isolation in the bind_checker Command

VCS NLP supports binding of the user-defined checker modules for isolation with the generic handles of the signals like isolation DATA and OUTPUT.

VCS NLP allows you to bind the checker module for each boundary port on which the isolation is applied with UPF_GENERIC_DATA (input of the clamp) and UPF_GENERIC_OUTPUT (output of the clamp) as part of a generic port or parameter list. For more information, see “[Isolation Example](#)” .

Use Model

For VCS two-step flow:

```
% vcs -sverilog -debug_access+all -upf <upf_file> -lpa_bind  
file_name.tcl    file_name.sv  
% simv
```

Example:

```
% vcs -sverilog design.v -debug_access+all -upf test.upf -  
lpa_bind checker.tcl checker.sv  
% simv
```

For VCS three-step flow:

```
% vlogan -sverilog file_name.sv  
% vcs testbench -debug_access+all -upf <upf_file> -sverilog  
-lpa_bind file_name.tcl  
% simv
```

Example:

```
% vlogan -sverilog design.v checker.sv  
% vcs testbench -debug_access+all -upf test.upf -sverilog  
-lpa_bind checker.tcl  
% simv
```

Isolation Example

Example 3-17 Testcase checker.sv

```
module iso_checker (isoSupply, isoGround, ISO_IN, ISO_OUT,  
ISOEN);
```

```

input isoGround, isoSupply, ISO_IN, ISO_OUT, ISOEN;
bit isopoweroff;
import UPF::*;
import SNPS_LP_MSG::*;

supply_net_type isoGround, isoSupply;
parameter string PD_NAME = "";
parameter string ISOPOLICY1 = "";
parameter string ISO_GRND_NET_NAME = "";
parameter string ISO_OUT_GEN = "";
parameter string ISO_IN_GEN = "";
parameter string ISO_PWR_NET_NAME = "";
assign isopoweroff = ((isoSupply.state == OFF) ||
(isoGround.state == OFF));

always @(ISO_OUT)
begin : my_msg_iso_checker
    my_assert : assert(ISO_OUT !== 1'bx) else
    begin
        lp_msg_print("LP_MSG_ISO_OUT_CORRUPT",
$formatf(lp_msg_get_format("LP_MSG_ISO_OUT_CORRUPT"),
ISO_OUT_GEN, ISOPOLICY1));
    end
end : my_msg_iso_checker

always @ (ISO_IN, ISO_OUT)
begin
    $display (" %m %0t [CHECKER] ISO_EN = %d
%s = %d, %s = %d for policy %s \n", $time,
ISOEN, ISO_IN_GEN, ISO_IN, ISO_OUT_GEN, ISO_OUT, ISOPOLICY1);
end
endmodule

```

Example 3-18 *Tcl Script checker.tcl*

```

proc iso_check {} {
foreach PD [query_power_domain *] {
    foreach ISO_POLICY [query_isolation * -domain $PD] {
        array set iso_detail [query_isolation $ISO_POLICY -domain
$PD -detailed]
        set PD_NAME [list PD_NAME $iso_detail(domain)]
        set isoSupply [list isoSupply
$iso_detail(isolation_power_net)]

```

```

        set isoGround [list isoGround
$iso_detail(isolation_ground_net)]
        set ISO_IN [list ISO_IN UPF_GENERIC_DATA]
        set ISO_OUT [list ISO_OUT UPF_GENERIC_OUTPUT]
        set ISOEN [list ISOEN $iso_detail(isolation_signal)]
        set ISO_PWR_NET_NAME [list ISO_PWR_NET_NAME
$iso_detail(isolation_power_net)]
        set ISO_GRND_NET_NAME [list ISO_GRND_NET_NAME
$iso_detail(isolation_ground_net)]
        set ISO_IN_GEN [list ISO_IN_GEN UPF_GENERIC_DATA]
        set ISO_OUT_GEN [list ISO_OUT_GEN UPF_GENERIC_OUTPUT]
        set ISO_FULLSCREEN @$PD.isolation.$ISO_POLICY
        set ISOPOLICY1 [list ISOPOLICY1 [query_original_name
$ISO_FULLSCREEN]]
        set ports_list {}
lappend ports_list $isoGround $isoSupply $ISO_IN $ISO_OUT
$ISOEN
        set params_list {}
lappend params_list $PD_NAME $ISO_PWR_NET_NAME
$ISO_GRND_NET_NAME $ISOPOLICY1 $ISO_IN_GEN $ISO_OUT_GEN

bind_checker no_iso_msg \
    -module iso_checker \
    -elements @$PD.isolation.$ISO_POLICY \
    -parameters $params_list \
    -ports $ports_list
array unset iso_detail
}
}
}
iso_check

```

Limitation

- For mixed designs, this feature is supported only for the isolation inserted in Verilog.

Generic Handles Support for Retention in the bind_checker Command

VCS NLP supports binding of the user-defined checker modules for retention with the generic handles `CLOCK` and `ASYNC_LOAD`.

VCS NLP supports the following to provide access to the design signals like `CLOCK` and `ASYNC_LOAD` load of the sequential element for which the UPF retention policy is applied:

- Instantiation of the checker module in the design scope where the corresponding UPF policy is applied.
- Identification of the type of the generic signals (`UPF_GENERIC_CLOCK`, `UPF_GENERIC_ASYNC_LOAD`) present in the design scope.

For more information, see “**Retention Example**” .

Use Model

For VCS two-step flow:

```
% vcs -sverilog -debug_access+all -upf <upf_file> -lpa_bind  
file_name.tcl file_name.sv  
% simv
```

Example:

```
% vcs -sverilog design.v -debug_access+all -upf test.upf -  
lpa_bind checker.tcl checker.sv  
% simv
```

For VCS three-step flow:

```
% vlogan -sverilog file_name.sv
```

```
% vcs testbench -debug_access+all -upf <upf_file> -sverilog
-lpa_bind file_name.tcl
% simv
```

Example:

```
% vlogan -sverilog design.v checker.sv
% vcs testbench -debug_access+all -upf test.upf -sverilog
-lpa_bind checker.tcl
% simv
```

Retention Example

Example 3-19 Testcase checker.sv

```
module ret_checker (RET_GRND_NET, RET_PWR_NET, GEN_CLK,
GEN_RST);
    input RET_GRND_NET, RET_PWR_NET, GEN_CLK, GEN_RST;

    import UPF::*;
    import SNPS_LP_MSG::*;

    supply_net_type RET_GRND_NET, RET_PWR_NET;
    parameter string PD_NAME = "";
    parameter string RET_GRND_NET_NAME = "";
    parameter string RET_PWR_NET_NAME = "";

    always @ (GEN_CLK or GEN_RST)
        begin
            <User Checker Code>
        end
endmodule
```

Example 3-20 Tcl Script checker.tcl

```
proc ret_check {} {
foreach PD [query_power_domain *] {
    foreach RET_POLICY [query_retention * -domain $PD] {
        array set ret_detail [query_retention $RET_POLICY -domain
$PD -detailed]
        set PD_NAME [list PD_NAME $ret_detail(domain)]
        set RET_PWR_NET [list RET_PWR_NET
```

```

$ret_detail(retention_power_net)]
    set RET_PWR_NET_NAME [list RET_PWR_NET_NAME
$ret_detail(retention_power_net)]
    set RET_GRND_NET [list RET_GRND_NET
$ret_detail(retention_ground_net)]
    set RET_GRND_NET_NAME [list RET_GRND_NET_NAME
$ret_detail(retention_ground_net)]
    set GEN_CLK [list GEN_CLK UPF_GENERIC_CLOCK]
    set GEN_RST [list GEN_RST UPF_GENERIC_ASYNC_LOAD]
    set ports_list {}
    lappend ports_list $RET_GRND_NET $RET_PWR_NET
$GEN_CLK $GEN_RST
    set params_list {}
    lappend params_list $PD_NAME $RET_GRND_NET_NAME
$RET_PWR_NET_NAME
    bind_checker ret_check \
        -module ret_checker \
        -elements @$PD.retention.$RET_POLICY \
        -parameters $params_list \
        -ports $ports_list
    array unset ret_detail
}
}
ret_check

```

Limitation

- This feature is supported only for Verilog in case of mixed designs.

Bias Support for Low Power Designs

Leakage current was negligible in earlier CMOS technologies. However, with shrinking device sizes and reduced threshold voltages, leakage power is becoming significant. Leakage power sometimes approaches the levels of dynamic power dissipation. The main causes of leakage power are reverse-bias p-n junction diode leakage, sub-threshold leakage, and gate leakage.

Substrate-biasing (back-biasing) is a low power design technique which enables dynamic tuning of performance and static power consumption of a CMOS device by applying a variable voltage on the nwell/pwell terminal of the device.

VCS NLP allows you to specify a bias supply network for the design and its intended usage through the following UPF commands:

- Using `enable_bias` design attribute in the UPF to enable the bias feature. Transitive effect is applied on the child blocks under the top bias block
- If a supply set is created (using `create_supply_set`) under the scope of bias enabled module, then nwell and pwell function is automatically created for this supply set
- If a power domain is created (using `create_power_domain`) under the scope of bias enabled module, then bias is automatically enabled for all instances under top bias block

Enabling Bias Feature

The bias feature is disabled by default. To enable bias feature for the design, you must set the `enable_bias` design attribute to TRUE as follows:

```
set_design_attributes -attribute enable_bias TRUE  
-elements { . }
```

The elements list can be any hierarchical instance. The value of the attribute can be TRUE or FALSE. Setting `enable_bias` on a design effectively designates all instances in the design as bias blocks. You

can disable the bias feature for individual sub-blocks of the design by setting the `enable_bias` attribute to `FALSE` on those blocks, thereby designating them as non-bias blocks.

A design whose top-level block has an `enable_bias` attribute is called a bias design, and the top-level block is called a bias block. All supply sets created under the scope of a bias block have four functions: power, ground, pwell and nwell. This includes implicit supply sets created for power domains under the scope of a bias block.

VCS NLP performs the following checks for the `enable_bias` attribute:

- The value of the attribute can only be set to `TRUE` if it is `TRUE` for all hierarchical blocks above the current scope. Setting the value to `TRUE` is always permitted if the current scope is top.
- The root elements of a power domain must be either all bias blocks or all non-bias blocks. That is, if the value of the `enable_bias` attribute is set to `FALSE`, then all parts of the block with the attribute should belong to the power domains scoped within the block. The only exception to this rule shall be:
 - The top power domain of the block maybe scoped outside the block as long as all root cells of the power domains are user-marked non-bias blocks.

Setting the `enable_bias` attribute on an instance has the following effects:

- All instances at or below the current scope automatically inherits the value of the attribute.

- All power domains inside a non-bias block in a bias design are converted to fully-restricted domains, that is, they cannot use any bias supply set declared in the scopes above the block. This prevents a supply set with bias rails from being used inside a non-bias block.
- The conversion is performed by the implementation tools automatically for any power domain created under the scope of a non-bias block, and is written to the output UPF.

For example, the following command:

```
create_power_domain PD -elements U1
```

is converted to:

```
create_power_domain PD -elements U1 -supply
{extra_supplies " "}
```

Bias Support for add_power_state

VCS NLP supports bias functions and negative floating point voltage value on pwell and nwell supply nets in the `add_power_state` command.

Example:

```
add_power_state TOP_SS -state STBY {-supply_expr {nwell ==
`{FULL_ON, -0.8} } -simstate CORRUPT_ON_ACTIVITY}
```

Establishing Connections to Bias Pins/Supplies

Bias Support for `create_supply_set`

The `create_supply_set` command is enhanced to accept the following predefined supply set functions, if the current scope is a bias block.

- power
- ground
- pwell
- nwell

If the current scope is not a bias block, VCS NLP generates an error message if you try to define the bias functions of a supply set.

Following is an example of how a bias supply set function is used.

Example 3-21 Bias Support for `create_supply_set`

```
create_supply_port TOP_VNW_SP  
create_supply_net TOP_VNW  
connect_supply_net TOP_VNW -port TOP_VNW_SP  
create_supply_set TOP_SS -function {nwell TOP_VNW}
```

In this example, the supply net `TOP_VNW` is defined for the `nwell` function of the supply set `TOP_SS`. Any standard or power management instance powered by `TOP_SS` will have its `nwell` bias pin connected to the supply net `TOP_VNW`.

Bias Support for `create_power_domain`

If the current scope is a bias block, the `create_power_domain` command is enhanced to implicitly create a supply set with bias functions if implicit supply sets are used. If explicit supply sets are used, a supply set with bias functions can be supplied through the `-supply` option of the `create_power_domain` command.

For example, consider the following command where `TOP_SS` is the supply set defined in [Example 3-21](#). This command implicitly connects the bias supply `TOP_VNW` to all normal instances using the primary power supply set.

```
create_power_domain TOP_PD -supply {primary  
TOP_SS}
```

Any standard or power management instance powered by `TOP_SS` will have its nwell bias pin connected to the supply net `TOP_VNW`.

Controlling Assertions

You can use the `SNPS_assertion_control` design attribute at compile-time to control power aware simulation behavior of assertions in the design. This attribute allows you to specify the simulation behavior either on the entire design or on specific power domains in the design.

You must specify the `SNPS_assertion_control` attribute in the UPF. Following is the syntax:

```
set_design_attributes \  
[-elements { <one_or_more_power_domains>} ] \  
-attribute {SNPS_assertion_control CONTINUE|SUSPEND|KILL}
```

Where, `-elements` is optional. If `-elements` list is not specified, assertion control is applied on the entire design hierarchy.

Following is the behavior of the `SNPS_assertion_control` attributes:

CONTINUE

No assertion control is exercised. This is the default setting.

SUSPEND

Assertions are turned OFF just before power down and resumed immediately after wakeup.

KILL

Assertions are turned OFF just before power down and are never resumed.

Examples

Example-1:

```
set_design_attributes \
-attribute {SNPS_assertion_control SUSPEND}
```

VCS NLP suspends all assertions in the design upon power down and resumes on wakeup.

Example-2:

```
set_design_attributes \
-elements {PD_CORE PD_IO}
-attribute {SNPS_assertion_control KILL}
```

Design assertions are turned OFF when power domains `PD_CORE` and `PD_IO` are powered down.

Querying Power Domains to Find Always-On and Collapsible Power Domains

A power domain whose supply is not routed through a power switch is considered as always-on power domain.

A power domain whose supply is routed through a switched supply (for example, power switch, power mux, LDOs, and so on) is considered as collapsible power domain.

VCS NLP is enhanced to find always-on and collapsible power domains from a specified list of power domains. This enhancement enables finer low power assertion control through query mechanism.

To enable this feature, you must use the `Tcl` property `is_switted_domain` for each power domain. The following is the `Tcl` procedure example using the `is_switted_domain` property:

```
proc supply_net_check {} {
    foreach PD [query_power_domain *] {
        set PRIMARY_POWER_NET [list PRIMARY_POWER_NET
$PD(primary_power_net)]
        set PRIMARY_GROUND_NET [list PRIMARY_GROUND_NET
$PD(primary_ground_net)]
        set PD_NAME [list PD_NAME $PD(domain_name)]
        set IS_SWITCHED_DOMAIN[list IS_SWITCHED_DOMAIN
$PD(is_switted_domain)]
        set ports_list {}
        lappend ports_list 1
        set params_list {}
        lappend params_list $PD_NAME $PRIMARY_POWER_NET
$PRIMARY_GROUND_NET $IS_SWITCHED_DOMAIN
        bind_checker_supply_net_msg \
        -module supply_net_checker \
        -elements @$PD \
        -parameters $params_list \
```

```
-ports $ports_list  
array unset supply_net_detail
```

If there are multiple root supply drivers to a power domain, VCS NLP checks if any of the root supply driver is a power switch port. If any one driver is a power switch port, the domain is switched (collapsible represented by 1).

If there is a single root supply driver, and if it is connected to a power switch port, then the domain is switched. Otherwise, the domain is not switched (always-on represented by 0).

Querying Supply Information of a Cell Instance

VCS NLP allows you to query the supply information of a hierarchical cell instance using the `query_pg_info_cell` command.

Following is the syntax of this command:

```
query_pg_info_cell -instance <instance_hier_path>
```

You must specify this command in the Tcl file provided with the `-lpa_bind` option.

This command returns key value pairs for .lib cell instances, where key is the pgtype of the pg pin and value is a list of pairs. The first value in the pair is the pg pin name and the second value is the handle of the UPF::supply_net_type supply.

Key	Value
pg_type, that is, primary_power, primary_ground, backup_power, backup_ground, internal_power, pwell, nwell	<p>List of pairs. The first value in the pair is the pg pin name and second value in the pair is the handle of the UPF::supply_net_type supply.</p> <p>Example:</p> <ol style="list-style-type: none"> For a cell with pg pin of type primary_power: Key: "primary_power" Value: {{VDD <handle>}} For a cell with 2 pg pins of type primary_power: Key: "primary_power" Value: {{VDD <handle>} {VDD1 <handle>}}

The output of this query command can be used to write a bind_checker directive to bind an instance with the required assertions. For example:

```
array set inst_pg [query_pg_info_cell $instance]
    set handle [lindex [lindex $inst_pg(primary_power) 0] 1]
    bind_checker PG_BIND_SW -module bm_snt -elements .
    ports [list "bm_power $handle"]
```

VCS NLP issues a warning message when the instance path provided to the query_pg_info_cell command is not a valid cell instance. Following is the sample warning message:

```
Warning-[UPF_NVCI] Not a valid cell instance
lpa_custom.tcl.s, 1
    Instance 'inst_path' is not a valid cell instance.
    Instance specified is not a valid cell instance. Please
specify a valid cell
```

Limitations

VHDL cell instances are not supported.

Searching the Logic Hierarchy for the Supply Ports

The `find_objects` UPF command allows you to search for instances, nets, ports, supply ports, or processes that are defined in the logic hierarchy.

VCS NLP allows you to search the logic hierarchy for the supply ports. When the `find_objects -object_type supply_port` command is specified, `find_objects` returns the supply ports that match the `search_pattern` specified in the `-pattern` option.

Following is the syntax:

```
find_objects scope
  -pattern search_ pattern
    [-object_type <model | inst | port | supply_port | net
     | process>]
    [-direction <in | out | inout>]
    [-transitive [<TRUE | FALSE>]]
    [-regexp | -exact]
    [-ignore_case]
    [-non_leaf | -leaf_only]
```

Example:

```
create_supply_port VDD
set result = [find_objects . -pattern VDD -object_type
supply_port]
puts "find_objects => $result"
```

Following is the output:

```
find_objects => VDD
```

Key Points to Note

- If the following options are specified with the `find_object` command, the result is filtered based on these options:
 - `direction` and `-ignore_case`
- The `-transitive`, `-regexp`, and `-exact` options are supported for `-object_type supply_port`.
- The search matches the following:
 - Supply ports created in the UPF prior to executing the `find_objects` command
 - PG Pins defined in the liberty file
 - HDL supply port of type `supply0` and `supply1`
- The following are not considered for match when `-object_type supply_port` is specified:
 - HDL logic ports. The UPF command which promotes the HDL port to supply port is called after the `find_object` command.
 - Supply nets with HDL implementation
- The `find_objects` command does not consider supply ports when it is called without the `-object_type` option.
- The `find_objects -object_type` port does not consider supply ports and only return the logic ports.
- VCS NLP issues error messages for the following cases:
 - Scope or search pattern starts with “..” or “/”

- Scope or `-pattern` is not specified
- Using "/" to match the hierarchy separator with `-regexp`
- The `-direction` option is specified with object types other than ports and `supply_port`
- The `-regexp` and `-exact` options are specified together
- The `-non_leaf` and `-leaf_only` options are specified together
- The `-non_leaf` and `-leaf_only` options are specified with `-object_type net` or `process`
- [Table 3-10](#) summarizes the scenarios of how an HDL object is treated by the `find_objects` command.

Table 3-10 Object Type Categorization Scenarios

Scenario	Behavior (<code>find_objects</code> should treat object as)
The <code>create_supply_port</code> command in the UPF	Supply port
PG pins in liberty	Supply port
PG port HDL ports connected to PG pins, but do not have explicit <code>create_supply_port</code> command in the UPF. a. It can be a top level PG port b. It can be an intermediate hierarchy port c. It can be connected to logic pins also	Logic port
Ports on HDL model of type <code>supply0/supply1</code>	Supply port
Ports on HDL model of type <code>UPF::supply_net_type</code> or <code>UPF::SupplyTypeT</code>	Supply port
Ports on HDL of any other type	Logic port
Data pins in DB	Logic port

Supply Equivalence Checks by VCS NLP

The `set_equivalent` command allows you to declare two supply nets (or supply sets) as electrically or functionally equivalent. Electrical equivalence is primarily related to supply ports and nets, whereas functional equivalence is primarily related to supply sets.

VCS NLP performs semantic checks for the following cases:

- Functionally equivalent supply sources with conflicting port states
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources with Conflicting Port States](#)” .
- Functionally equivalent supply sources with missing port states
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources with Missing Port States](#)” .
- Functionally equivalent supply sources having conflicting type
This check is performed for the functionally equivalent supply sources. For more information, see “[Functionally Equivalent Supply Sources Having Conflicting Type](#)” .
- PST entry with conflicting port states
This check is performed for both functionally and electrically equivalent supply sources. For more information, see “[PST Entry with Conflicting Port States](#)” .
- Supply source state does not match with the state of equivalent supply source

This check is performed for the functionally equivalent supply sources. For more information, see “[Supply Source State Does Not Match with the State of Equivalent Supply Source](#)” .

Functionally Equivalent Supply Sources with Conflicting Port States

Functionally equivalent supply sources should not have different voltages defined for same state name.

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test1.upf` file where VDD1 and VDD2 are functionally equivalent, but the state named ON have different voltage values.

Example 3-22 test1.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON 1.0}
add_port_state VDD2 -state {ON 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

VCS NLP issues the UPF_FEWCP warning message for this case. Following is the sample warning message:

```
Warning-[UPF_FEWCP] Functional equivalent supply with
conflicting port states
```

```
Port state 'ON' with voltage value '0.8' defined for Supply Net/Supply Port 'tb/top/VDD2' has a conflicting definition with different voltage value '1' defined for Supply Net/Supply Port 'tb/top/VDD1' at (test1.upf, 23).
```

Functionally Equivalent Supply Sources with Missing Port States

Functionally equivalent supply sources should have the same number of states defined.

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test2.upf` file where VDD1 and VDD2 are functionally equivalent, but have different number of states defined where VDD1 has 1 state while VDD2 has 2 states.

Example 3-23 test2.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON 0.8}
add_port_state VDD2 -state {ON 0.8}
add_port_state VDD2 -state {ON2 1.0}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

In the above case, VCS NLP issues the `UPF_FEWMPs` warning message for missing state `ON2` for supply VDD1.

Following is the sample warning message:

```
Warning-[UPF_FEWMPS] Functional equivalent supply with
missing port states
State 'ON2' missing for Supply Net/Supply Port 'tb/top/VDD1'
has been defined for functional equivalent Supply Net/Supply
Port 'tb/top/VDD2' with voltage value '1' at (test2.upf, 23).
```

Functionally Equivalent Supply Sources Having Conflicting Type

Functionally equivalent supply sources should not have different supply type (that is, power/ground).

You must use the `-power=hier_pst_check` compile-time option to enable this check.

Consider the following `test3.upf` file where `VDD1` and `VSS` are functionally equivalent, but have different supply type (that is, power/ground).

Example 3-24 test3.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VSS
create_supply_net VSS_net
connect_supply_net VSS_net -ports VSS

add_port_state VDD1 -state {ON1 1.0}
add_port_state VSS -state {ON 0.0}

set_equivalent -nets {VDD1_net VSS_net} -function_only

create_power_domain -include_scope -primary_power_net
VDD1_net -primary_ground_net VSS_net
```

In the above case, VCS NLP issues the `UPF_FESCT` error message. Following is the sample error message:

```
Error-[UPF_FESCT] Functional equivalent supply have
conflicting type
electrical_equivalent_nets.upf, 10
Supply Net/Supply Port 'tb/top/VDD1' used as 'power' has
functional equivalent Supply Net/Supply Port 'tb/top/VSS'
used as 'ground'.
```

PST Entry with Conflicting Port States

Functionally or electrically equivalent supply sources should have the same state in a PST state.

This check is applicable for both electrical and functional equivalent supplies. For functional supplies, the check is enabled using the `-power=hier_pst_check` compile-time option.

Consider the following `test4.upf` where VDD1 and VDD2 are functionally equivalent. The PST state have the same supply source, but different port states with different supply voltage.

Example 3-25 test4.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {OFF off}
add_port_state VDD2 -state {OFF off}

add_port_state VDD1 -state {ON1 1.0}
add_port_state VDD2 -state {ON2 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only

create_pst top_pst -supplies {VDD1 VDD2}
```

```
add_pst_state ON -pst top_pst -states {ON1 ON2}
add_pst_state OFF -pst top_pst -states {OFF OFF}
```

In the above case, VCS NLP issues the UPF_PST_ESCPS warning message. Following is the sample warning message:

```
Warning-[UPF_PST_ESCPS] Pst entry with conflicting port states check_pst_state_scl.upf, 35
```

```
State ON of tb/top/top_pst refers to different voltage states for equivalent supplies. Port/power state 'ON1' (1) defined for supply VDD1 has different voltage from state 'ON2' (0.8) defined for supply VDD2.
```

Supply Source State Does Not Match with the State of Equivalent Supply Source

Functionally equivalent supply sources should have the same voltage in testbench.

Consider the following test5.upf and testbench files where VDD1 and VDD2 are functionally equivalent, but have different voltages at specified in testbench for the same timestamp.

Example 3-26 test5.upf

```
create_supply_port VDD1
create_supply_net VDD1_net
connect_supply_net VDD1_net -ports VDD1

create_supply_port VDD2
create_supply_net VDD2_net
connect_supply_net VDD2_net -ports VDD2

add_port_state VDD1 -state {ON1 1.0}
add_port_state VDD2 -state {ON2 0.8}

set_equivalent -nets {VDD1_net VDD2_net} -function_only
```

Example 3-27 Testbench

```
Testbench:  
#10  
supply_on(VDD1 1.0)  
supply_on(VDD2, 0.8)
```

At the timestamp #10, the voltage value of the equivalent supply pads have different voltage set. You must perform this check at runtime.

The LP_EQUIVALENT_SUPPLY_MISMATCH low power assertion is flagged by default for functionally equivalent supply sources with conflicting voltages/states at a given timestamp.

In the above case, VCS NLP issues the LP_EQUIVALENT_SUPPLY_MISMATCH error message at runtime. Following is the sample output in simulation log:

```
[10 ns] [INFO] [LP_SUPPLY_PAD_CALL] Supply Pad function  
'UPF::supply_on' called for Supply Pad 'VDD1'. Supply Pad  
value changed to '{FULL_ON, 1000000}'. File: tb.v, Line:  
21.
```

```
[10 ns] [INFO] [LP_SUPPLY_PAD_CALL] Supply Pad function  
'UPF::supply_on' called for Supply Pad 'VDD2'. Supply Pad  
value changed to '{FULL_ON, 800000}'. File: tb.v, Line: 22.
```

```
[10 ns] [ERROR] [LP_EQUIVALENT_SUPPLY_MISMATCH] Supply  
source 'tb/top/VDD2' state {FULL_ON, 0.80 V} does not match  
with state {FULL_ON, 1.00 V} of equivalent supply source  
'tb/top/VDD1'
```

The default severity of the LP_EQUIVALENT_SUPPLY_MISMATCH message is ERROR. You can change the severity using the set_lp_msg_severity option in the configuration file during runtime.

Interpretation of ‘/’ in the UPF Commands

VCS NLP supports interpretation of `set_scope /` in hierarchical designs involving the `load_upf` command as per the *IEEE 1801-2018 UPF LRM*.

Definitions

The following table describes the definitions of the terms used in this document:

Term	Description
Root scope	Absolute root of the design, that is, <code>\$root</code> .
Design top instance	The design top instance (represented by a hierarchical name relative to the root scope) is an instance in the hierarchy representing a design for which power intent is defined.
Design top module	The design top module is the module for which the UPF file expressing the power intent is written.
Current scope	The current scope is an instance that is, or is a descendant of, the design top instance (represented by a relative pathname from the design top instance).
Design-relative hierarchical name	A design-relative hierarchical name is interpreted relative to the current design top instance by removing the leading <code>/</code> character and interpreting the remainder as a rooted name in the scope of the current design top instance.

Rules of Interpretation of ‘/’ in the UPF

- For the `load_upf -scope` command,
 - VCS NLP implicitly changes the design top instance (and design top module) to the `-scope` value.

- If `-scope` is not specified, `load_upf` executes the commands in the current scope. In this case, current scope, design top instance, and design top module are not affected. This is equivalent to Tcl source command.
- After completion of `load_upf -scope`, the design top instance and design top module are restored to previous values.
- A hierarchical name that starts with a leading '/' character is a design-relative hierarchical name.
- For the `set_scope` command,
 - The current scope is changed locally within the subtree depending on the current design top instance or design top module.
 - The name of the previous scope is returned as a design-relative hierarchical name.
 - The instance name can be a simple name, a scope-relative hierarchical name, a design-relative hierarchical name, the symbol /, the symbol ., or the symbol ...
 - If the instance name is /, the current scope is set equal to the current design top instance.
 - If the instance name is ., the current scope is unchanged.
 - If the instance name is ..., and the current scope is not equal to the current design top instance, the current scope is changed to the parent scope.
 - An error message is issued if the instance name is .. and the current scope is equal to the current design top instance.

- The `set_scope` command is only allowed to change scope within this subtree. It cannot change the scope to:
 - A scope above the design top instance
- By default, VCS NLP defines design top instance as the power top or from first `set_design_top`.
 - If `set_design_top` refers to a model, then it is the self instance of the model
 - If `set_design_top` refers to pathname, then it is the pathname.

Examples

The following table lists the examples for the interpretation of `set_scope /` in the hierarchical designs involving the UPF commands from this release onwards. All pathnames specified in the following table are absolute pathnames.

Command	<i>Design Top Instance</i>	<i>Current Scope</i>
<code>set_scope bot</code>	<code>/top</code>	<code>/top/mid/bot</code>
<code>set_scope .</code>	<code>/top</code>	<code>/top/mid/bot</code>
<code>set_scope ..</code>	<code>/top</code>	<code>/top/mid</code>
<code>set_scope /</code>	<code>/top</code>	<code>/top</code>
<code>load_upf mid.upf -scope mid</code>	<code>/top/mid</code>	<code>/top/mid</code>
<code>set_scope .</code>	<code>/top/mid</code>	<code>/top/mid</code>
<code>set_scope bot</code>	<code>/top/mid</code>	<code>/top/mid/bot</code>
<code>set_scope /</code>	<code>/top/mid</code>	<code>/top/mid</code>
<code>set_scope ..</code>	# ERROR	
<code># load_upf finished</code>	<code>/top</code>	<code>/top</code>

load_upf mid.upf	/top	/top
set_scope mid/bot	/top	/top/mid/bot
set_scope /	/top	/top
load_upf macro.upf -scope macro_inst	/top/macro_inst	/top/macro_inst
set_scope /bot	/top/macro_inst	/top/macro_inst/bot
set_scope /	/top/macro_inst	/top/macro_inst
set_scope ..	# ERROR	
# load_upf finished	/top	/top
set_scope macro_inst/bot	# ERROR	

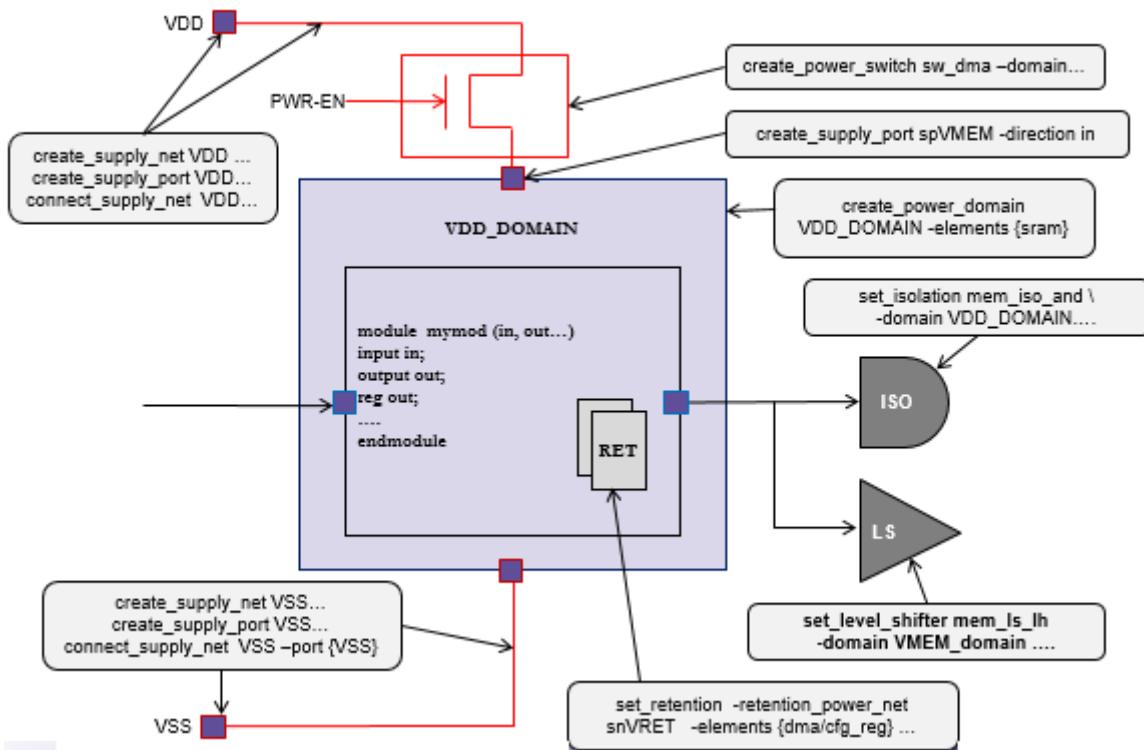
UPF Structural Components

HDL does not support low power details and semantics. UPF compliments the HDL by capturing structure and semantics of power domains in the design.

Table 3-11 UPF and HDL Structural Components

HDL	UPF
Module	Power domains
Logic ports	Supply ports
Wires	Supply nets
Logic gates	Power switches
Assertions	Power State Table
Behavioral logic	Isolation/Retention/Level shifter policies

Figure 3-4 UPF Structural Components



UPF Script Examples

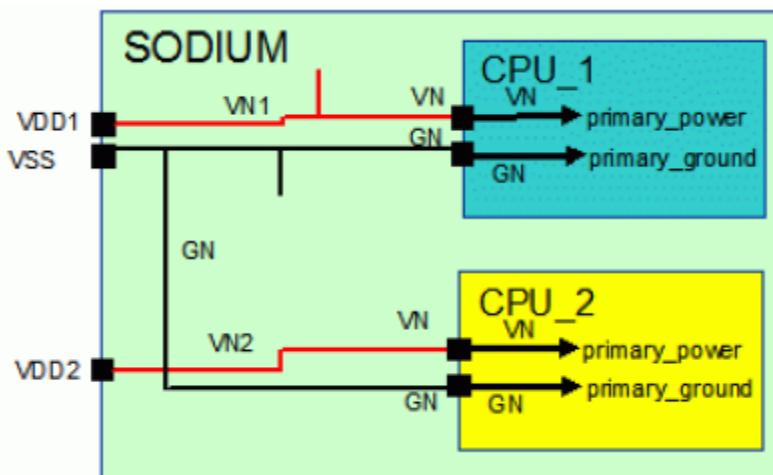
The following topics demonstrate the UPF syntax used for specifying power intent:

- Simple Multi-Voltage Design
- Bottom-Up Power Intent Specification
- Top-Down Power Intent Specification
- Supply Set Example

Simple Multi-Voltage Design

The simple multi-voltage chip design shown in [Figure 3-5](#) has two power supplies, VDD1 and VDD2, at different voltage levels. The chip has two internal blocks, CPU_1 and CPU_2, both of which are instances of the same CPU block. Block CPU_1 and the top level of the chip use VDD1, whereas block CPU_2 uses VDD2. Both VDD1 and VDD2 operate as always-on power throughout the chip. The same ground, VSS, is used throughout the chip.

Figure 3-5 Simple Multi-Voltage Chip Design



Bottom-Up Power Intent Specification

In a bottom-up flow, the lower-level CPU blocks are designed independently from the top-level chip. Therefore, their power intent must be specified at the block level and then later integrated into the chip-level power intent specification.

Following is the power intent script at the block level, `CPU.upf`:

```
create_power_domain PD
```

```

create_supply_net VN -domain PD
create_supply_net GN -domain PD
set_domain_supply_net PD -primary_power_net VN -
primary_ground_net GN
create_supply_port VN
create_supply_port GN
connect_supply_net VN -ports {VN}
connect_supply_net GN -ports {GN}

```

Following is the power intent script at the top level, SODIUM.upf:

```

load_upf CPU.upf -scope CPU_1
load_upf CPU.upf -scope CPU_2
# still at scope SODIUM
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_power_domain PD
create_supply_net VN1 -domain PD
connect_supply_net VN1 -ports {VDD1 CPU_1/VN}
create_supply_net VN2 -domain PD
connect_supply_net VN2 -ports {VDD2 CPU_2/VN}
create_supply_net GN -domain PD
connect_supply_net GN -ports {VSS CPU_1/GN CPU_2/GN}
set_domain_supply_net PD -primary_power_net VN1 -
primary_ground_net GN
# PD, CPU_1/PD and CPU_2/PD are different power domains.

```

Top-Down Power Intent Specification

If the chip is designed from the top down, the power intent of the design can be specified directly from the top level, if desired, as demonstrated by the following example:

```

create_power_domain PD_CPU_1 -elements {CPU_1}
create_power_domain PD_CPU_2 -elements {CPU_2}
create_power_domain PD_SODIUM
create_supply_port VDD1
create_supply_port VDD2

```

```

create_supply_port VSS
create_supply_net VN1 -domain PD_CPU_1
create_supply_net VN1 -domain PD_SODIUM -reuse
connect_supply_net VN1 -ports {VDD1}
create_supply_net VN2 -domain PD_CPU_2
connect_supply_net VN2 -ports {VDD2}
create_supply_net GN -domain PD_CPU_1
create_supply_net GN -domain PD_CPU_2 -reuse
create_supply_net GN -domain PD_SODIUM -reuse
connect_supply_net GN -ports {VSS}
set_domain_supply_net PD_CPU_1 \
-primary_power_net VN1 -primary_ground_net GN
set_domain_supply_net PD_CPU_2 \
-primary_power_net VN2 -primary_ground_net GN
set_domain_supply_net PD_SODIUM \
-primary_power_net VN1 -primary_ground_net GN

```

Supply Set Example

The following script demonstrates the usage of explicit supply sets:

```

# UPF section for Explicit Supply Set definition
## Power Domain Supply Sets
#set_scope /
create_supply_set VDD_VSS
create_supply_set VDD_LOW_VSS
create_supply_set VDD$VSS_p0
create_supply_set VDD$VSS_p1
create_supply_set VDD$VSS_minc
create_supply_set VDD_L$O$VSS_p2
create_supply_set VDD_L$O$VSS_p3
# UPF section for Power Domain definition
## Number of PDs: 6
create_power_domain TOP \
-supply {extra_supplies_0 VDD_VSS} \
-supply {extra_supplies_1 VDD_LOW_VSS}
set_domain_supply_net TOP -primary_power_net VDD_VSS.power \
-primary_ground_net VDD_VSS.ground
create_power_domain LEON3_p0 \
-elements {u0_0/pd_switchable} \
-supply {extra_supplies_0 VDD$VSS_p0} \
-supply {extra_supplies_1 VDD$VSS}
set_domain_supply_net LEON3_p0 -primary_power_net VDD$VSS_p0.power \
-primary_ground_net VDD$VSS_p0.ground
create_power_domain LEON3_p1 \
-elements {u0_1/pd_switchable} \
-supply {extra_supplies_0 VDD$VSS_p1} \

```

```

-supply {extra_supplies_1 VDD_VSS}
set_domain_supply_net LEON3_p1 -primary_power_net VDDS_VSS_p1.power \
-primary_ground_net VDD_VSS_p1.ground
create_power_domain LEON3_p2 \
-elements {u0_2/pd_switchable} \
-supply {extra_supplies_0 VDD_LOWS_VSS_p2} \
-supply {extra_supplies_1 VDD_VSS} \
-supply {extra_supplies_2 VDD_LOW_VSS}
set_domain_supply_net LEON3_p2 -primary_power_net VDD_LOWS_VSS_p2.power \
-primary_ground_net VDD_LOWS_VSS_p2.ground
create_power_domain LEON3_p3 \
-elements {u0_3/pd_switchable} \
-supply {extra_supplies_0 VDD_LOWS_VSS_p3} \
-supply {extra_supplies_1 VDD_VSS} \
-supply {extra_supplies_2 VDD_LOW_VSS}
set_domain_supply_net LEON3_p3 -primary_power_net VDD_LOWS_VSS_p3.power \
-primary_ground_net VDD_LOWS_VSS_p3.ground
create_power_domain LEON3_misc \
-elements {u_m/u_m_pd} \
-supply {extra_supplies_0 VDDS_VSS_misc} \
-supply {extra_supplies_1 VDD_VSS}
set_domain_supply_net LEON3_misc -primary_power_net VDDS_VSS_misc.power \
-primary_ground_net VDD_VSS_misc.ground
# UPF section for port and supply set connections
## High Voltage supply 1.08
create_supply_port VDD
connect_supply_net VDD_VSS.power -ports VDD
## Low Voltage Supply 0.7
create_supply_port VDD_LOW
connect_supply_net VDD_LOW_VSS.power -ports VDD_LOW
## Ground nets
create_supply_port VSS
connect_supply_net VDD_VSS.ground -ports VSS
# Power Switches
create_power_switch leon3_p0_sw \
-domain LEON3_p0 \
-input_supply_port {in VDD_VSS.power} \
-output_supply_port {out VDDS_VSS_p0.power} \
-control_port {p0_sd u_m/u_power_controller_top/p0_sd} \
-on_state {p0_on_state in {!p0_sd}}
create_power_switch leon3_p1_sw \
-domain LEON3_p1 \
# Updates
create_supply_set VDD_LOW_VSS -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_p0 -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_p1 -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_misc -function {ground VDD_VSS.ground} -update
create_supply_set VDD_LOWS_VSS_p2 -function {ground VDD_VSS.ground} \
-update
create_supply_set VDD_LOWS_VSS_p3 -function {ground VDD_VSS.ground} \
-update
# UPF Power States
## Top Supply States
add_power_state VDD_VSS -state TOP_HV \
{-supply_expr {power == `'{FULL_ON, 1.08}`}}
add_power_state VDD_VSS -state GND \
{-supply_expr {ground == `'{FULL_ON, 0.0}`}}
add_power_state VDD_LOW_VSS -state TOP_LV \

```

```

{-supply_expr {power == `{FULL_ON, 0.7}}}
## Off Supply states
add_power_state VDDS_VSS_p0 -state P0_HV \
{-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDDS_VSS_p0 -state P0_OFF \
{-supply_expr {power == `{OFF}}}
add_power_state VDDS_VSS_p1 -state P1_HV \
{-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDDS_VSS_p1 -state P1_OFF \
{-supply_expr {power == `{OFF}}}
add_power_state VDD_LOWS_VSS_p2 -state P2_LV \
{-supply_expr {power == `{FULL_ON, 0.7}}}
add_power_state VDD_LOWS_VSS_p2 -state P2_OFF \
{-supply_expr {power == `{OFF}}}
add_power_state VDD_LOWS_VSS_p3 -state P3_LV \
{-supply_expr {power == `{FULL_ON, 0.7}}}
add_power_state VDD_LOWS_VSS_p3 -state P3_OFF \
{-supply_expr {power == `{OFF}}}
add_power_state VDDS_VSS_misc -state MISC_HV \
{-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDDS_VSS_misc -state MISC_OFF \
{-supply_expr {power == `{OFF}}}
# PST
create_pst LEON3_MP_PST -supplies \
{VDD_VSS.power VDD_LOW_VSS.power VDDS_VSS_p0.power VDDS_VSS_p1.power \
VDD_LOWS_VSS_p2.power VDD_LOWS_VSS_p3.power VDDS_VSS_misc.power \
VDD_VSS.ground}
add_pst_state INIT -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_OFF P2_OFF P3_OFF MISC_OFF GND}
add_pst_state P0_BOOT -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_OFF P2_OFF P3_OFF MISC_OFF GND}
add_pst_state P0_HV_P1_BOOT -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_HV P2_OFF P3_OFF MISC_OFF GND}
add_pst_state P0_HV_P1_HV_P2_BOOT -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_HV P2_LV P3_OFF MISC_OFF GND}
add_pst_state ALL_ON -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_HV P2_LV P3_LV MISC_HV GND}
add_pst_state HIGH_PERF -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_HV P2_OFF P3_OFF MISC_HV GND}
add_pst_state LOW_PERF -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_OFF P2_LV P3_LV MISC_HV GND}
add_pst_state P3_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_OFF P2_OFF P3_LV MISC_HV GND}
add_pst_state P2_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_OFF P2_LV P3_OFF MISC_HV GND}
add_pst_state P1_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_HV P1_OFF P2_OFF P3_LV MISC_HV GND}
add_pst_state P0_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_HV P2_OFF P3_LV MISC_HV GND}
add_pst_state ULTRA_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV \
P0_OFF P1_HV P2_OFF P3_OFF MISC_OFF GND}

```


4

Wildcard Support in UPF Commands

VCS NLP supports wildcard characters to refer design objects in the UPF. Support for wildcard characters is only applicable for the design (HDL)/UPF objects relevant to the option of the command. You can use ? and * wildcard characters in the string pattern to match any single character or a sequence of zero or more characters, respectively.

The supported wildcard characters are:

- ? — Matches any single character
- * — Matches any sequence of zero or more characters

You can use these wildcard characters in a string or different hierarchies. A wildcard character does not match:

- The hierarchy separator or patterns involving the hierarchy separator.

- The * wildcard character does not expand to the generate instances. Generate labels are treated as a separate hierarchy, hence explicit wildcard per generate label name is needed for a match.

Examples

Verilog Example:

Consider the following generate instances:

```
U1/gen1[0]/inst
U1/gen1[1]/inst
U1/gen1[2]/inst
```

Where, gen1 is the generate label.

You must use the following wildcard pattern in VCS NLP to match the above generate instances:

```
u1/gen1[*]/* or u1/gen1/*
```

VHDL Example:

Consider the following generate instances:

```
U1/gen1(0)/inst
U1/gen1(1)/inst
U1/gen1(2)/inst
```

Where, gen1 is the generate label.

You must use the following wildcard pattern in VCS NLP to match the above generate instances:

```
u1/gen1(*)/*
```

- Wildcard in index for arrays/vectors.

For example:

`inst*` — Matches instance `inst[0]`, `instA[1]`, `inst2[0]` ..

`array*[0]` — Does not match `array[3][0]`

`array[*][1]` — Does not match `array[0][1]`, `array[1][1]`

Table 4-1 UPF Commands Supporting Wildcard Characters

Commands	Options
<code>create_power_domain</code>	<code>-elements</code>
<code>set_isolation</code>	<code>-elements, -exclude_elements</code>
<code>set_retention</code>	<code>-elements</code>
<code>set_related_supply_net</code>	<code>-object_list</code>
<code>set_design_attributes</code>	<code>-elements, -models</code>
<code>set_level_shifter</code>	<code>-elements, -exclude_elements</code>
<code>connect_logic_net</code>	<code>-ports</code>
<code>set_port_attributes</code>	<code>-elements and -ports</code>
<code>map_isolation_cell</code>	<code>-elements</code>
<code>connect_supply_net</code>	<code>-ports, -exclude_elements</code>

Examples

Wildcard characters:

`?` – For single character

`*` – For zero or more characters

Following is the basic example with `create_power_domain`:

Consider two instances: Muxa, Muxb

To add both the instances in PD1:

```
create_power_domain PD1 -elements {Mux?}
```

To add only Muxa using wildcard:

```
create_power_domain PD1 -elements {*xa}
```

Wildcard Usage with Isolation

The following example shows the usage of wildcard in isolation policy.

List of elements to be isolated:

```
alu_i/Add1/MUXCY_L0/muxcy_inst1/CI,  
alu_i/Add1/MUXCY_L0/muxcy_inst1/DI,  
alu_i/Add2/MUXCY_L0/muxcy_inst1/CI,  
alu_i/Add2/MUXCY_L0/muxcy_inst1/DI
```

You can write the above list of elements using wildcard as follows:

```
set_isolation iso_dom3  
-domain <ref_domain_name>  
-applies_to both  
-elements {alu_i/Add?/M*Y_L0/muxcy_i*1/?I }  
-location self
```

Wildcard Usage with Retention

For the above mentioned elements, you can write the retention policy as follows:

```
set_retention ret_dom3
-domain domain3
-retention_power_net VDD -retention_ground_net VSS
-elements {alu_i/Add1/M*Y_L0/muxcy_i*1/?I }
```

Wildcard Usage with `set_design_attributes`

You can write model names in similar way, as illustrated above, using wildcard in `UPF_dont_touch` policy.

```
set_design_attributes -attribute UPF_dont_touch TRUE -
models sta*
set_design_attributes -attribute UPF_dont_touch TRUE -
models MUXCY_?
```

Wildcard Usage with `set_related_supply_net`

The following example shows the usage of wildcard with `set_related_supply_net`:

```
set_related_supply_net -object_list {u_wrap_A/
ou?_*[1]}
```

Wildcard Usage with `connect_supply_net`

The following example shows the usage of wildcard with `connect_supply_net`:

```
connect_supply_net VDD -ports {A/* /VDD*}
```

Matches both supply ports in the UPF and ports in HDL. The above expression matches A/inst/VDD, A/inst/VDD1, A/inst1/VDDPP, and so on.

Treating Generate Instance as a Single Token for Wildcard Matching

By default, VCS NLP treats each level of generate block name as a separate token while matching the wildcard patterns in the UPF.

You can use the `-power=dcccompat_gen_inst_name_match` compile-time option to treat the generate instance name (any depth) as a single token for matching the wildcard character “`*`” specified in the UPF.

Note:

The `-power=dcccompat_gen_inst_name_match` option is not needed for wildcard matching in the `find_objects` command.

For example, consider the following instance path where `gen_inv[0].i_inv` is a generate instance:

`i_wrap/i_bloc/gen_inv[0].i_inv`

Consider the following command in the UPF file to match the above instance:

```
create_power_domain PD_INV -elements {i_wrap/  
i_bloc/gen*.i_inv}
```

By default, the above wildcard pattern does not match `i_wrap/i_bloc/gen_inv[0].i_inv` as `gen_inv[0]` and `i_inv` are treated as separate instances.

With `-power=dcccompat_gen_inst_name_match`, the generate instance `gen_inv[0].i_inv` is treated as a single instance name and single token for wildcard matching, thus resulting in a match.

Limitations of Wildcard Support

- Multi-select expressions cannot be specified with wildcard characters:

Example: a/*/b/c[0][1]
a/b/c/d[*][1]

The following case is treated as single select, and is supported:

a/b/c/d[0][*]

- Wildcard characters to access members of struct, union, interface, or modports are supported only for the `set_isolation` and `set_retention` commands.
- Wildcard characters are not supported with -elements if the elements refer to the internals of the module specified in -models. Wildcard characters in -elements are supported for matching hierarchical references only.
- Union members are not supported as part of -elements.

Note:

VCS NLP generates the following warning message when a wildcard expression does not find any matching object:

“Warning- [UPF_NMFW] No match found for wildcard”

5

Supply Set Based Power Intent Specification

VCS NLP supports supply set based specification of power intent. Supply set is a collection of supply functions that provides a comprehensive power source for the design elements. VCS NLP supports the following pre-defined supply net functions:

- Power
- Ground
- pwell
- nwell
- User-defined functions

The supply set approach is a more abstract way of specifying the power network than the explicit supply port or supply net based format. This feature allows you to specify different states of supply set and power domains using the `add_power_state` command.

VCS NLP supports the following supply set based UPF features:

- [Converting an Existing Supply Net Based UPF to a Supply Set](#)
- [Explicit Supply Sets with Implicit Supply Nets](#)
- [Implicit Supply Sets](#)
- [Supply Network Initialization](#)
- [Supply Set Association](#)
- [Disabling the Propagation of Supply Set State Names to Functional Nets](#)
- [Supply Sets in Strategies](#)
- [Setting Power State Using `set_supply_set_state` Function from Testbench](#)
- [Simstates](#)

Converting an Existing Supply Net Based UPF to a Supply Set

VCS NLP allows you to convert an existing supply net based UPF to a supply set. To enable this feature, create supply sets and connect the supply set handles to the existing supply nets using the

`create_supply_set` command. Supply ports can be driven using the `supply_on` and `supply_off` calls. Consider the code shown in [Example 5-1](#).

Example 5-1 Existing UPF

```
create_supply_port VDD
create_supply_net VDD
connect_supply_net VDD -ports VDD
create_supply_port VSS
create_supply_net VSS
connect_supply_net VSS -ports VSS
#Define supply set and explicit nets
create_supply_set SS_TOP -function {power VDD} -function
{ground VSS}
create_supply_port VDD
create_supply_net VDD
connect_supply_net VDD -ports VDD
create_supply_port VSS
create_supply_net VSS
connect_supply_net VSS -ports VSS

#add_power_state command for supply set SS_TOP

add_power_state SS_TOP -state HV {-supply_expr {power ==
`{FULL_ON, 1.08} } }
add_power_state SS_TOP -state TOP_OFF {-supply_expr {power
== OFF} }
```

You can replace the `create_power_domain` and `set_domain_supply_net` commands with a single `create_power_domain -supply` command, as shown in [Example 5-2](#).

Example 5-2 Existing UPF with Create Power Domain

```
create_power_domain PD_TOP
set_domain_supply_net PD_TOP -primary_power_net VDD -
primary_ground_net VSS

#Supply set based
```

```
create_power_domain PD_TOP -supply {primary SS_TOP}
```

You can use the explicitly created supply nets and ports to drive different voltage values on to the supply set functions with the `supply_on` and `supply_off` calls.

Explicit Supply Sets with Implicit Supply Nets

With this approach, there are no explicit supply ports or supply nets. The power domain is associated with a supply set as shown in [Example 5-3](#).

Example 5-3 Explicit Supply Sets

```
create_supply_set SS_TOP  
create_power_domain PD_TOP -supply {primary SS_TOP}
```

The power domain is driven to different states based on logic expressions defined in the `add_power_state` command (see [Example 5-4](#)).

Example 5-4 Add Power State Command

```
add_power_state SS_TOP -state HV {-logic_expr {pd_top}  
-simstate NORMAL }  
add_power_state SS_TOP -state TOP_OFF {-logic_expr {!pd_top}  
-simstate CORRUPT }
```

Implicit Supply Sets

This feature allows you to use default power domain handles in the UPF (see [Example 5-5](#)), thereby avoiding the need to create supply sets.

Example 5-5 Implicit Supply Sets

```
create_power_domain TOP
add_power_state TOP.primary -state HV { -logic_expr {pd_top}
-simstate NORMAL }
add_power_state TOP.primary -state TOP_OFF { -logic_expr
{!pd_top} -simstate CORRUPT }
```

In [Example 5-5](#), the power domain name is treated as a supply set handle. Following are the available sets:

- `TOP.primary` — Primary of domain
- `TOP.default_retention` — Used as retention power when a retention strategy for the domain has no retention power specified.
- `TOP.default_isolation` — Used as isolation power when an isolation strategy for the domain has no isolation power specified.

All power domain state transitions happen based on the value of the specified logic expression.

Supply Network Initialization

All the supply ports and supply set functions with no explicit nets are initialized to `ON` state by default using the `add_port_state` or `add_power_state` command. If there are multiple `ON` states, VCS NLP selects one of the `ON` states. The default `ON` state can also be selected by specifying the following command in the UPF or in the runtime configuration file:

```
set_design_attributes -elements {
hierarchical_name_of_supply_net} -attribute
SNPS_default_supply_net_state STATE_NAME
```

If no port or power states are defined, then all the supply ports and supply set functions with no explicit nets start in **FULL_ON** state with 1.0V. Any undriven supply net is initialized to OFF.

Use the **-power=undriven_supplies_off** compile-time option or the **SNPS_treat_undriven_as_off** runtime design attribute to initialize all the supply ports (except supply pads) and supply set functions with no explicit nets to OFF state.

Use the **SNPS_treat_undriven_as_undetermined** runtime design attribute to initialize all the undriven supply ports/nets and supply set functions with no explicit nets to UNDETERMINED state.

Use the

SNPS_treat_undriven_supply_nets_as_undetermined runtime design attribute to initialize all the undriven supply nets to UNDETERMINED state.

Following is the use model to use the runtime configuration file:

```
% simv -power power_config.tcl
```

The attribute is defined in the **power_config.tcl** file. For example, as follows:

```
set_design_attributes -elements {  
hierarchical_name_of_supply_net} -attribute  
SNPS_default_supply_net_state STATE_NAME
```

Supply Set Association

The semantics of supply set association that are in alignment with IEEE 1801-2013 (Section 4.6.2, Power state of a supply set) are listed as follows:

- Power states are not merged between the explicit supply set and associated supply set handle
- You can define power states for explicit supply set and supply set handle (`PD.primary`) separately
- Reporting and coverage on the explicit supply sets is enabled only when the power states are specified. Both reporting and coverage are turned off for implicit supply sets
- Multiple power states can be active and most conservative simstate is applied in case of a conflict (`CORRUPT > CORRUPT_ON_ACTIVITY > NORMAL`)

Disabling the Propagation of Supply Set State Names to Functional Nets

As per the IEEE Std 1801 LRM, the state names defined for the supply set in the `add_power_state` command must not be propagated to the functional net.

VCS NLP supports the above LRM behavior. The state name of the supply set specified in the `add_power_state` command is not propagated to the functional nets defined in the `-supply_expr` expression by default.

This feature disables the propagation of supply set state names to the associated supply nets and stops the creation of an incorrect system PST. It helps you to write Power State Table (PST) according to the IEEE Std 1801 LRM.

Case-1: Supply set implicit net is used in the PST with the state defined in the add_power_state command.

Example:

```
create_supply_set SS1
add_power_state SS1 -state ON1 {-supply_expr {power ==
{FULL_ON 2.0}}}

create_pst MY_PST -supplies      {SS1.power}
add_pst_state STATE -pst MY_PST {ON1}
```

VCS NLP issues the following error message for this case:

```
Error-[UPF_PST_STUNRSLV] Pst entry could not be resolved
test.upf, 39
Pst entry 'ON1' corresponding to SupplyNet/SupplyPort 'tb/
top/SS1.power' in State 'STATE' of PST 'tb/top_inst/core1/
MYPST' could not be resolved.
```

Case-2: Supply port is used in the PST with the state defined in the add_power_state command.

Example:

```
create_supply_port VDD
create_supply_set SS1 -function {power VDD}
add_power_state SS1 -state ON1 \
{-supply_expr {power == {FULL_ON 1.0}}}

create_pst MY_PST -supplies      {VDD}
add_pst_state STATE -pst MY_PST {ON1}
```

VCS NLP issues the following error message for this case:

```
Error-[UPF_PST_STUNRSLV] Pst entry could not be resolved  
test.upf, 39  
Pst entry 'ON1' corresponding to SupplyNet/SupplyPort 'tb/  
top/VDD' in State 'STATE' of PST 'tb/top_inst/core1/MYPST'  
could not be resolved.
```

Case-3: Supply ports without the state defined using the `add_port_state` command.

Consider the case where the supply ports are specified without the `add_port_state` command and without explicit initialization in the testbench.

```
create_supply_net VDD1  
create_supply_port VDD1  
  
connect_supply_net VDD1 -ports {VDD1}  
  
create_supply_set SS1 -function {power VDD1} ...  
  
add_power_state SS1 \  
-state ON {-supply_expr {power == {FULL_ON 1.2}}} }
```

The states of the supply net function are not propagated to the connected ports, in this case from `SS1.power` to `VDD1`. Hence, they remain uninitialized at time 0, and supply set `SS1` is initialized to the `DEFAULT_NORMAL` state.

```
[0] [INFO] [LP_PORT_STATE_INIT] Supply Source 'tb/top_inst/  
VSS1' initialized to 'ON' state with root net 'tb/top_inst/  
VSS1' at state 'FULL_ON' and voltage 0 V.
```

```
[0] [INFO] [LP_SS_INIT] Supply Set 'tb/top_inst/SS1'  
initialized to state 'DEFAULT_NORMAL' with simstate  
'NORMAL'.
```

You must explicitly set the default start state using runtime calls from the testbench.

Backward Compatibility

For backward compatibility, you must specify the `set_design_attributes enable_state_propagation_in_add_power_state TRUE` attribute at the scope for which you want to enable the supply set state propagation.

Note:

- Only the `-elements` option of the `set_design_attributes` command is supported.
Only top design module or design module instances are allowed in `-elements`.
- The `-models`, `-exclude_elements`, and `-transitive` options of the `set_design_attributes` command are not supported.

VCS NLP issues an error message for the following cases where the `enable_state_propagation_in_add_power_state` design attribute is set to TRUE.

- The `create_power_state_group` command is used.
- The `-group` option is used in the `add_power_state` command.

Supply set state names are propagated to the supply net functions for the following scenario:

```
set_design_attributes -elements {.} \
-attributenable_state_propagation_in_add_power_state
```

TRUE

```
create_supply_set SS1 -function {power VDD}
add_power_state SS1 -state ON1 \
{-supply_expr {power == {FULL_ON 1.0}}}

create_pst MY_PST -supplies {SS1.power}
add_pst_state STATE -pst MY_PST {ON2}
```

The `enable_state_propagation_in_add_power_state` attribute value will not be allowed to change in the current scope once the state is defined for a Supply Set using the `add_power_state` command. For example:

```
set_design_attributes -elements {.} \
-attribute
enable_state_propagation_in_add_power_state
TRUE
create_supply_set SS1 -function {power VDD}
add_power_state SS1 -state ON1 \
{-supply_expr {power == {FULL_ON 1.0}}}

//NOT ALLOWED
set_design_attributes -elements {.} \
-attribute
enable_state_propagation_in_add_power_state
FALSE
```

Valid Usage Examples

Following are the supported usage scenarios for the `enable_state_propagation_in_add_power_state` design attribute:

- The design attribute is set on a design module instance named `block`

```
set_design_attributes -elements {block} \
    -attribute enable_state_propagation_in_add_power_state
TRUE
```

- The design attribute is set on a design module instance named block

```
set_scope block
set_design_attributes -elements {.} \
    -attribute enable_state_propagation_in_add_power_state
TRUE
```

- If the design attribute is set to TRUE for the block element in the following example, it is applicable for all its lower scopes.

```
set_design_attributes -elements {block} \
    -attribute enable_state_propagation_in_add_power_state
TRUE
```

```
set_scope block/mid
create_supply_set SS2
add_power_state SS2 -state ON2 {-supply_expr {power ==
{FULL_ON 2.0}}}
```

```
create_pst MY_PST -supplies {SS2.power}
add_pst_state STATE -pst MY_PST {ON2}
```

Error Scenarios

VCS NLP issues an error message for the following usages of the enable_state_propagation_in_add_power_state design attribute:

- Empty elements list specified with the design attribute

```
set_design_attributes -elements { } \
    -attribute enable_state_propagation_in_add_power_state
TRUE
```

- The -elements list not specified with the design attribute

```
set_design_attributes \
-attribute enable_state_propagation_in_add_power_state
TRUE
```

- -models/-exclude_elements specified with the design attribute

```
set_design_attributes -models {M1} -exclude_elements
{U1}\ \
-attribute enable_state_propagation_in_add_power_state
TRUE
```

- Attribute value other than true/TRUE specified with the design attribute

```
set_design_attributes -elements {.} \
-attribute enable_state_propagation_in_add_power_state
JUNK
```

- An invalid object or object other than design top/design module instance specified with the design attribute

```
set_design_attributes {JUNK clk}\ \
-attribute enable_state_propagation_in_add_power_state
TRUE
```

- An upper scope has value specified as TRUE and lower scope has value specified as FALSE

```
set_design_attributes -elements {.} \
-attribute enable_state_propagation_in_add_power_state
TRUE
```

```
set_scope block
set_design_attributes -elements {.} \
-attribute enable_state_propagation_in_add_power_state
FALSE
```

Supply Sets in Strategies

Isolation and retention strategies support the use of supply set functions instead of power and ground nets.

Limitation

- Only NORMAL, CORRUPT and CORRUPT_ON_ACTIVITY simstates are supported.

Setting Power State Using `set_supply_set_state` Function from Testbench

VCS NLP allows you to control supply set states through testbench using the `set_supply_set_state` function.

Syntax:

```
set_supply_set_state (input string supply_set_name, input  
string power_state_name);
```

Where, `supply_set_name` is the name of the supply set, and `power_state_name` is the name of the power state.

The `set_supply_set_state` call is applicable for supply sets with power state definitions using `-supply_expr` with supply set function, which is either undriven or has a single undriven root supply driver (see [Root Supply Driver](#)).

For supply sets matching the above criteria, the following semantics apply when `set_supply_set_state` is called:

1. Supply set function in the `supply_expr` is driven to the state/voltage specified in `supply_expr`.
2. If only `-supply_expr` is specified, then power state is active.
3. If `-logic_expr` is also specified, the power state will be active only if the logic expression evaluates to TRUE.

Consider the following example where VDD is undriven:

```
## Power Domain Associated with Supply Set ##
create_supply_set SS -function {power VDD} -function {ground VSS}
create_power_domain PD -supply SS

## Power States ##
add_power_state PD.primary -state HV {-supply_expr {power ==
'{FULL_ON, 1.2}} -simstate NORMAL}
add_power_state PD.primary -state LV {-supply_expr {power ==
'{OFF, 0.0}} -simstate CORRUPT}

initial begin
  set_supply_set_state("tb/dut/PD.primary", "HV");
#100
  set_supply_set_state("tb/dut/PD.primary", "LV");
end
```

The above `set_supply_set_state` calls result in VDD to be `{FULL_ON, 1.2V}` at time 0 and `{OFF, 0V}` at time 100. For the supply set, PD.primary, HV and LV are the active power states at time 0 and 100 respectively.

Simstates

The following are the two default simstates when there is no explicit simstate specified:

- `DEFAULT_NORMAL`

- DEFAULT_CORRUPT

A supply set is in DEFAULT_NORMAL if all functions are in FULL_ON state. It is in DEFAULT_CORRUPT, if any one supply function (power/ground) is OFF. Also, the state of a supply set is DEFAULT_CORRUPT when at least one of the functions defined for the supply set is not associated with a supply net.

If multiple power state specifications of a supply set match at the same time, the simstate of a domain is based on the precedence order specified in the LRM. The order is DEFAULT_CORRUPT, CORRUPT, NORMAL, and DEFAULT_NORMAL.

The default simstate for a power state is taken as CORRUPT if a supply expression uses OFF supply state for a supply function power or ground.

In alignment with IEEE 1801-2013 (Section 4.6.2 Power state of a supply set), simstate defined on the explicit supply set is not propagated to the supply set handle, and add_power_state with -simstate must be defined on supply set handle PD.primary.

```
## Power States on Supply Set Handle (domain.primary) ##
add_power_state PD.primary -state HV {-supply_expr {power == '{FULL_ON, 1.2}} -simstate NORMAL}
add_power_state PD.primary -state STBY {-supply_expr {power == '{FULL_ON, 0.8}} -simstate CORRUPT_ON_ACTIVITY}
add_power_state PD.primary -state LV {-supply_expr {power == '{OFF, 0.0}} -simstate CORRUPT}
```

6

Low Power Simulation

The following sections describe the low power design simulation in VCS NLP:

- [Running Low Power Designs](#)
- [HDL Constructs and Techniques in VCS](#)
- [Support for UPF Supply Functions](#)
- [Voltage-Based supply_on](#)
- [Monitoring the Simstate of a Power-Managed Instance](#)
- [Initial Block Retriggering](#)

Running Low Power Designs

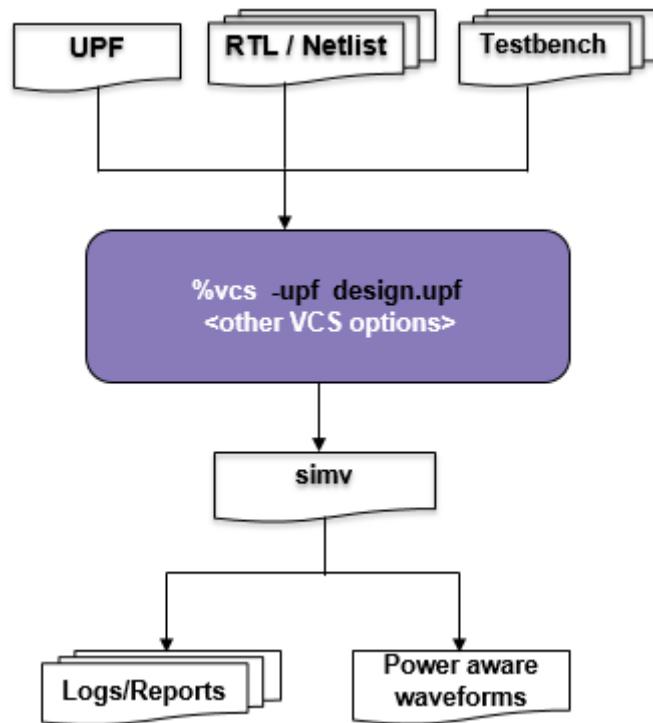
This section describes the VCS NLP low power simulation flow and the options required to compile and run low power designs in the following sections:

- [VCS NLP Low Power Simulation Flow](#)
- [Using Low Power in VCS Two-Step Flow](#)
- [Using Low Power in VCS Three-Step Flow](#)

VCS NLP Low Power Simulation Flow

Specifying the `-upf` option at compile time automatically enables VCS NLP. You must specify the UPF file using the `-upf` option at `vcs` step as shown in [Figure 6-1](#).

Figure 6-1 VCS NLP Low Power Simulation Flow



Using Low Power in VCS Two-Step Flow

You can specify only a single top-level UPF file for compilation in VCS two-step flow.

To compile the UPF file, use the following command:

```
vcs -upf <upf_file> -power=<power_options> <vcs_options>  
<source_files>
```

```
-upf
```

Allows you to pass the UPF file to VCS.

```
<upf_file>
```

Identifies the UPF file.

<power_options>

Used to specify VCS NLP specific compile-time options.

<vcs_options>

Identifies VCS options. You can use any HDL compilation options in VCS with the `-upf` option. For example, `-v`, `-y`, `+incdir`, `+define`, `-f`, and so on.

<source_file>

Identifies the HDL source file.

Specifying Power Top

To specify the power top, use any of the following commands:

`set_design_top <heir_path_of_instance>`

Hierarchical instance level where the UPF is applied on the design.

`-power_top <TOP module name>`

Captures the module whose instance is the design top.

- It is advisable to have only single instance of the module in the design, else use `set_design_top` to specify power top.
- Do not specify the `-power_top` command if you have specified the `set_design_top` command in the UPF.

Note:

- UPF commands can be called from Tcl procedures and can be interleaved with various Tcl coding styles. VCS executes the Tcl commands that are part of the UPF file.
- Pure Tcl commands with outputs ('puts') are executed during UPF compile time. The results are either displayed on the screen or reported in a log file as indicated by the options specified on the VCS command line.

Power Techniques Supported in VCS

VCS supports the following power techniques:

- Power gating
- Isolation - Automatically insert isolation cells as per the defined isolation strategy in the UPF.
- Retention - Save and restore events as described in the UPF strategy.

Using Low Power in VCS Three-Step Flow

To use low power mixed designs in VCS three-step flow:

1. Analyze your design:

For VHDL files,

```
% vhdlan [vhdlan_options] file1.vhd file2.vhd
```

For Verilog files,

```
% vlogan [vlogan_options] file1.v file2.v
```

SystemVerilog files

```
% vlogan -sverilog [vlogan_options] file1.sv file2.sv  
file3.v
```

2. Elaborate your design using the **-upf** option:

```
% vcs [elab_options] [libname].design_unit -upf  
<upf_file>
```

3. Simulate your design:

```
% simv
```

Power Techniques Supported in VCS Three-step Flow

VCS three-step flow supports the following power techniques:

- Power gating
- Shutdown
- Isolation
- Retention

Limitations

- HDL constructs are not supported for isolation are enum, record, int, real, and boolean
- HDL constructs are not supported for retention are real, int, and record

HDL Constructs and Techniques in VCS

VCS supports the following HDL constructs in syntax and in low power semantics:

1. Verilog 95
2. Verilog 2K
3. SystemVerilog design constructs
 - Not corrupted:
 - Time
 - sv-string and character
 - event
 - Dynamic data types: class objects (dynamic arrays, smart queues, and so on.)

Note:

- In VHDL, all data types are corrupted with `left. The only exception is std_logic, which is corrupted by X.
- All data types having `left as 0 are corrupted by 0.

Example: real, natural, and so on.

- The corrupted value of enum depends on the enum base type.
- Corrupted with SIZE 'X:
 - Integer
 - logic

- Corrupted with SIZE '0:
bit, byte, real, int, short int, and long int

Support in syntax, but not in semantics

The SVTB HDL syntax is supported in VCS. VCS issues a parse warning if any SystemVerilog testbench construct is encountered. However, low power semantics are not applied on this construct during simulation.

Support for UPF Supply Functions

VCS NLP supports the following functions from the UPF package:

```
function bit supply_on( string pad_name, real value);
function bit supply_off( string pad_name );
function bit supply_partial_on( string pad_name, real value );
function supply_net_type get_supply_value( string name );
function real get_supply_voltage( supply_net_type arg );
function bit get_supply_on_state( supply_net_type arg );
function state get_supply_state( supply_net_type arg );
```

You must import the UPF package before using the above functions:

```
import UPF::*;


```

The names of the supply nets, supply ports, and supply set functions in the following function calls can be full hierarchical names from the simulation top or from the power top scope.

The `supply_on`, `supply_partial_on`, and `supply_off` calls work only on undriven supply nets, supply ports, or supply set functions.

The `get_supply_value` call works on any supply nets, supply ports, or supply set functions.

- `supply_on/supply_partial_on/supply_off` — Sets the state and voltage value on a supply net/port.

Verilog Example:

```
import UPF::*;

initial begin
    supply_on("VDD", 1.08);
    supply_on("testbench/TOP_U/u2/V_U", 1.06);
    supply_on("SS1.power", 1.0); //Explicit supply set
    supply_on("PD1.primary.power", 1.0); //Implicit supply set
    supply_off("VDD");
    supply_off("testbench/TOP_U/u2/V_U");
    supply_partial_on("testbench/TOP_U/VDD", 1.00);
end
```

VHDL Example:

```
use IEEE.upf.all;
entity testbench is
end testbench;
architecture behave of testbench is
...
    signal dummy :boolean;
begin
    simulate:process
begin
    dummy <= supply_on("testbench/TOP_U/Vdd", 1.08);
    dummy <= supply_on("testbench/TOP_U/V1", 1.08);
    wait for 100 ns;
    dummy <= supply_off("testbench/TOP_U/Vdd");
    ...
end process;
```

Note:

You can also specify the `-liblist` UPF compile-time option as an alternative.

- `get_supply_value` — Return value is of type `UPF::supply_net_type`
- `get_supply_voltage` — Gets the voltage value of the supply. Return value is of type ‘real’.
- `get_supply_on_state` — Return value is of type ‘bit’. Returns 1 if the state of the supply is `FULL_ON`, else 0.
- `get_supply_state` — Gets the information on the current state (`FULL_ON/OFF/PARTIAL_ON/UNDETERMINED`) of the supply. Return value is of type `UPF::state`.

Example:

```
module test;
import UPF::*;
supply_net_type snt;
real voltage;
bit isOn;
state currState;

initial begin
    snt = get_supply_value("testbench/TOP_U/u2/V_U");
    voltage = get_supply_voltage(snt);
    isOn = get_supply_on_state(snt);
    currState = get_supply_state(snt);
end
..
endmodule
```

UCLI Support for Supply Functions

For UCLI, the hierarchical path to the supply port must be absolute path. Relative paths are not supported.

Use Model:

```
% vcs file_name.v -upf test.upf -debug_access+all
```

```
% simv -ucli
ucli% lp_show -supply_on <port name> <voltage value>
ucli% run <time>
ucli% lp_show -supply_off <port name>
```

Example:

```
ucli% lp_show -supply_on testbench/TOP_U/Vdd 1.08
ucli% lp_show -supply_on testbench/TOP_U/V2 1.08
ucli% run 300ns
ucli% lp_show -supply_off testbench/TOP_U/Vdd
```

Limitations

- This feature is not supported for VHDL testbench.

Voltage-Based supply_on

By default, a UPF `supply_on` function call changes the state of the supply pad to `FULL_ON` irrespective of the voltage specified.

If the `SNPS_voltage_based_supply_on` attribute is set to TRUE, as shown below, in the UPF or runtime configuration file (`-power <config_file>`), then the `supply_on("SUPPLY_PAD," 0.0)` function with voltage as 0 on a supply pad of type power changes the state of the supply to OFF.

```
set_design_attributes -attribute
SNPS_voltage_based_supply_on TRUE
```

The `SNPS_voltage_based_supply_on` attribute has no impact on the supply of type ground. Supply is determined as ground type, if the supply pad is driving a supply net:

- which is `primary_ground_net` of a domain

or

- which is `isolation_ground_net` or `retention_ground_net`
or
- which is a related ground supply used in `set_related_supply_net`
or,
- tied to a macro PG pin of type `primary_ground`

Monitoring the Simstate of a Power-Managed Instance

VCS NLP provides a way to monitor the simstate of the power-managed instances from the testbench using the `upf_simstate` variable of type `UPF::power_state_simstate`. The `power_state_simstate` is a predefined type in the UPF package.

```
typedef enum {NORMAL,  
             CORRUPT,  
             CORRUPT_ON_ACTIVITY,  
             CORRUPT_STATE_ON_CHANGE,  
             CORRUPT_STATE_ON_ACTIVITY  
         }  
power_state_simstate;
```

You must import the UPF package and declare a variable of type `power_state_simstate` by name `upf_simstate` in the power-managed module. Once declared, the user-defined `upf_simstate` variable mirrors the simstate of an instance during simulation. You can access this variable from the testbench using the XMR.

Example

```
module testbench;
    M1 dut(..);

    always @ (dut.upf_simstate)
begin
    if (dut.upf_simstate == NORMAL)
        $display("Domain PD1 is in NORMAL state");
    else if (dut.upf_simstate == CORRUPT)
        $display("Domain PD1 is in CORRUPT state");
end
..
endmodule

module M1(..);
..
import UPF::*;

power_state_simstate upf_simstate;
..
endmodule
```

Note:

- Variable name must be `upf_simstate` in order to mirror the simstate of an instance.
- `upf_simstate` can only be read.
- Access the `upf_simstate` from the testbench or an always-on module using XMR since accessing it locally in the power-managed module results in the monitoring logic (for example, always block) to be inactive during the CORRUPT simstate.

Initial Block Retriggering

VCS NLP supports initial block retriggering (`reinit`). You can use initial blocks in your design to assign power-up values to the storage elements, write behavioral models of analog macros for the digital simulations, handle memory reads through tasks, and so on.

However, for the design with multiple power domains, when you remove voltage from the power domain containing the initial blocks, VCS NLP corrupts the variables used in the initial blocks. This might eventually cause the simulation to fail, as these initial blocks are executed only once by the simulator and not every time the power is restored.

Enabling Initial Block Retriggering in VCS NLP

Use the `set_design_attributes` command to enable initial block retriggering (`reinit`) on a design element. Following is the syntax of this command:

Syntax

```
set_design_attributes
  -elements element_list
  -models model_list
  [-attribute name value]*
  -transitive <TRUE|FALSE>
```

Arguments

- `-elements element_list` — List of design elements or named initial blocks.
- `-models model_list` — List of models to be attributed.

- `-attribute name value` — For the enumerated design element or model, associate the attribute `SNPS_reinit` or `SNPS_dont_reinit` with a value of TRUE or FALSE.
 - `-transitive <TRUE|FALSE>` — When `-transitive` is TRUE, the command applies to the descendants of the elements. Default value is FALSE.
-

Initial Block Retriggering Supported Attributes

Initial block retriggering supports the attributes shown in [Table 6-1](#).

Table 6-1 Supported Attributes for Initial Block Retriggering

Attribute Name	Description
<code>SNPS_reinit</code>	The supported values for this attribute are TRUE and FALSE: <ul style="list-style-type: none"> • TRUE value enables <code>reinit</code> on the initial blocks targeted by the command. • FALSE value resets the <code>reinit</code> marking on the initial blocks targeted by the command.
<code>SNPS_dont_reinit</code>	The supported values for this attribute are TRUE and FALSE: <ul style="list-style-type: none"> • TRUE value enables <code>don't reinit</code> property on the initial blocks targeted by the command. • FALSE value resets the <code>don't reinit</code> marking on the initial blocks targeted by the command.

Reinit Property

Based on the values passed to the SNPS_reinit and SNPS_dont_reinit attributes, VCS applies one of the following three properties to the initial blocks in the power-managed part of the design:

- uninitialized — By default, every initial block has an uninitialized property. These initial blocks are not retriggered on power-up.
- reinit — Initial blocks having this property are retriggered on power-up.
- don't reinit — Initial blocks having this property are not retriggered on power-up.

VCS NLP applies the reinit or don't reinit property in descending order of priority, as listed below:

1. Named initial blocks in a module
2. Design instance
3. Design module
4. Power domain
5. Entire design

Transitive Reinit

You can apply the `reinit` property on a design instance or module transitively on all power-managed descendants of the design module instances. VCS NLP applies only the `transitive reinit` property on the initial blocks which have the `uninitialized` property.

Examples for Different Reinit Scenarios

- Module-level support. This indicates that the initial blocks of all instances of a module are retriggered on power-up.

```
set_design_attributes -models ModuleForReInit \
-attribute SNPS_reinit TRUE
```

- Named initial block support at module level.

```
set_design_attributes -elements named_initial_1 \
-models ModuleForReInit \
-attribute SNPS_reinit TRUE
```

- Transitive support for an instance. This indicates that the initial blocks of descendants are also retriggered on wakeup.

```
set_design_attributes -elements U_Inst \
-attribute SNPS_reinit TRUE \
-transitive TRUE
```

- Transitive support for a module. This indicates that the initial blocks of descendants are also retriggered on wakeup.

```
set_design_attributes \
-models ModuleForReInit \
-transitive TRUE \
-attribute SNPS_reinit TRUE
```

- Non-transitive support for an instance.

```
set_design_attributes -elements U_Inst \
-attribute SNPS_reinit TRUE
```

- Non-transitive support for a module.

```
set_design_attributes \
-models ModuleForReInit \
-attribute SNPS_reinit TRUE
```

- Power domain level support. This indicates that the initial blocks of all instances in a power domain are retriggered on wakeup.

```
set_design_attributes -elements PD_Core \
-attribute SNPS_reinit TRUE
```

- Design-wide reinit

```
set_design_attributes -attribute SNPS_reinit TRUE
```

Transitive Reinit Limitations

Hierarchical named initial blocks are not supported for `reinit` or `don't reinit`. Only named initial blocks in a module can be specified for `reinit` or `don't reinit`.

7

Compile-Time Static Reports

VCS NLP compile-time static reports are generated in the compile-time static reports directory. The default name of this directory is `mvsim_native_reports`, and its default location is your current working directory. The following sections describe compile-time static reports:

- [Redirecting/Renaming `mvsim_native_reports` Directory](#)
- [Compile-Time Static Report for `set_design_attributes`](#)
- [Default Reports Dumped Under `mvsim_native_reports` Directory](#)

Redirecting/Renaming mvsim_native_reports Directory

You can use the `-mvrpt` compile-time option to redirect or rename the compile static reports directory. Following is the syntax of the `-mvrpt` option:

```
-mvrpt=<full_path_to_directory>
```

Key Points to Note

- The `-mvrpt` option supports both absolute paths and relative paths
- If the specified directory does not exist, a new directory is created and the reports are dumped in that directory
- If the write permissions are not available for the specified directory, a warning message is issued, and the reports are dumped in the default directory `mvsim_native_reports` in the current working directory

Example

The following option dumps the VCS NLP compile static reports in the `/my_path/dir_test/reports` directory:

```
-mvrpt=/my_path/dir_test/reports
```

Compile-Time Static Report for `set_design_attributes`

VCS NLP dumps the report files with the information about all the design attributes used in the UPF with the `set_design_attributes` command.

All the reports are dumped in the **SDA** folder located in the `mvsim_native_reports` directory. If any attribute is declared at the design level, it is dumped in the common file `SDA_attribute_summary.rpt`.

Some of the attributes are defined at the power domain or design model level; for such attributes, individual reports are generated in the same **SDA** folder with the respective names of the attributes (`<attribute>.rpt`).

VCS NLP dumps the static report files for the following compile-time design attributes, if they are specified in the UPF:

- `deselect_coverage_object`
- `enable_bias`
- `is_soi`
- `lower_domain_boundary`
- `conservative_diff_supply_only_isolation`
- `SNPS_default_supply_net_state`
- `SNPS_voltage_based_supply_on`
- `SNPS_force_semantics`
- `SNPS_default_power_upf2sv_vct`

- SNPS_default_power_upf2vh_vct
- SNPS_default_ground_upf2sv_vct
- SNPS_default_ground_upf2vh_vct
- SNPS_default_power_sv2upf_vct
- SNPS_default_ground_sv2upf_vct
- SNPS_macro_port_based_corruption
- SNPS_random_corruption
- SNPS_reinit
- SNPS_dont_reinit
- UPF_dont_touch

Default Reports Dumped Under `mvsim_native_reports` Directory

[Table 7-1](#) lists all the reports that are dumped by default by VCS NLP under the `mvsim_native_reports` directory.

Table 7-1 Default Reports Dumped Under `mvsim_native_reports` Directory

Report Name	Description
<code>flop_inference.rpt</code>	Captures the details of the flops which are retained using retention strategy. For more information, see “ flop_inference.rpt ” .
<code>hierarchical_flop_inference.rpt</code>	Compile-time static report for Verilog. Captures the details of the inferred retention registers. For more information, see “ hierarchical_flop_inference.rpt ”
<code>hierarchical_flop_inference_vhdl.rpt</code>	Compile-time static report for VHDL. Captures the details of the inferred retention registers. For more information, see “ hierarchical_flop_inference_vhdl.rpt ”
<code>isolation_association_summary.rpt</code>	Captures the details of the associated isolation cell instances and their associated isolation strategies. For more information, see “ isolation_association_summary.rpt ”
<code>library_mapping.rpt</code>	Captures the cell information. For more information, see “ library_mapping.rpt ” .
<code>library_match.rpt</code>	Captures the matching/non-matching cell information. For more information, see “ library_match.rpt ” .
<code>library_mapping_after_upf.rpt</code>	Captures the port-based corruption information of a cell. For more information, see “ library_mapping_after_upf.rpt ” .
<code>SDA/SDA_attribute_summary.rpt</code>	Captures information of the <code>set_design_attributes</code> command specified at element or model level.
<code>srsn_spa_association.rpt</code>	Captures information about the SRSN and SPA buffers associated with a port. For more information, see “ srsn_spa_association.rpt ” .

Report Name	Description
upf_state_machine.rpt	Captures the objects with legal/illegal states/transitions being covered.
upf_control_port_coverage.rpt	Captures the functional coverage on the control ports/signals of a power switch, isolation strategy, and retention strategy.
upf_supply_set_power_state.rpt	Captures information on all the power states of the supply sets in the design. For more information, see “ upf_supply_set_power_state.rpt ” .
undefined_db_cell.rpt	Lists the cells or modules defined within `celldefine and `endcelldefine, but not having a corresponding matched liberty db cell.
upf_supply_source_rsd.rpt	Captures information on all types of supply sources like root supply of a supply source and net/ports connected to the source.
upf_construct_statistics.rpt	Captures information on the number of low power constructs used in the UPF file.

The following topics describe the common VCS NLP reports in detail:

- [flop_inference.rpt](#)
- [hierarchical_flop_inference.rpt](#)
- [hierarchical_flop_inference_vhdl.rpt](#)
- [isolation_association.rpt](#)
- [isolation_association_summary.rpt](#)
- [isolation_insertion_info.txt](#)

- library_mapping.rpt
- library_mapping_after_upf.rpt
- library_match.rpt
- repeater_port_association.rpt
- srsn_spa_association.rpt
- upf_supply_set_power_state.rpt
- latch_inference.rpt

flop_inference.rpt

Description

The flop_inference.rpt file reports only the details of the flops which are retained using retention strategy.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
flop_inference
```

Sample Report

```
1 RETENTION POLICY/MODULE MAPPING WHERE A POLICY IS BEING APPLIED
2 =====
3 Retention Policy: tb/top/A/retention/RET_1, Module : flop1
4 =====
5 Inferred sequential process: module: flop1 file:flop.v line: 4
6 Clock(s):posedge clk
7 Retention policy:tb/top/A/retention/RET_1
8 Inferred FF:out
9 Inferred FF:save ____ out
10 -----
11 Retention Policy: tb/top/A/retention/RET_1, Module : flop2
12 =====
13 Inferred sequential process: module: flop2 file:flop.v line: 15
14 Clock(s):negedge clk
15 Retention policy:tb/top/A/retention/RET_1
16 Inferred FF:out
17 Inferred FF:save ____ out
18 -----
19 Retention Policy: tb/top/A/retention/RET_1, Module : flop3
20 =====
21 Inferred sequential process: module: flop3 file:flop.v line: 26
22 Clock(s):posedge clk
23 Retention policy:tb/top/A/retention/RET_1
```

hierarchical_flop_inference.rpt

Description

The `hierarchical_flop_inference.rpt` file reports the details of the inferred retention registers.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
flop_inference
```

Sample Report

```
3
4 set sr_elements {{ tb.top.v0.u1.out { 0 0 } tb.top.v0.u1.clk tb/top/save
    negedge tb/top/save low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS
  }\
5 { tb.top.v0.u2.out { 0 0 } tb.top.v0.u2.clk tb/top/save negedge tb/top/sa
ve low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS }\
6 { tb.top.v0.u3.out { 0 0 } tb.top.v0.u3.clk tb.top.v0.u3.reset tb/top/sav
e negedge tb/top/save low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VS
S }\
7 { tb.top.v0.u4.out { 0 0 } tb.top.v0.u4.clk tb.top.v0.u4.nset tb/top/save
    negedge tb/top/save low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS
  }\
8 { tb.top.v0.u5.out { 0 0 } tb.top.v0.u5.clk tb.top.v0.u5.nset tb/top/save
    negedge tb/top/save low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS
  }\
9 { tb.top.v0.u6.out { 0 0 } tb.top.v0.u6.clk tb.top.v0.u6.reset tb/top/sav
e negedge tb/top/save low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VS
S }\
10 { tb.top.v0.u7.out { 0 0 } tb.top.v0.u7.clk tb/top/save negedge tb/top/sa
ve low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS }\
11 { tb.top.v0.u8.out { 0 0 } tb.top.v0.u8.clk tb/top/save negedge tb/top/sa
ve low tb/top/VDD1 tb/top/VSS tb/top/VDD_ret tb/top/VSS }\
12 { tb.top.v0.u10.out { 0 0 } tb.top.v0.u10.clk tb/top/save negedge tb/t@aa
```

hierarchical_flop_inference_vhdl.rpt

Description

The `hierarchical_flop_inference_vhdl.rpt` file is compile-time static report for VHDL. This report captures the details of the inferred retention registers.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
hierarchical_flop_inference_vhdl
```

Sample Report

```
3
4 set sr_elements {{ /tb/TOP/BLK(0)/II/Q(4) tb/TOP/VDD_top tb/TOP/VSS_top tb/
TOP/VDD_1 tb/TOP/VSS_1 }\
5 { /tb/TOP/BLK(0)/II/Q(4) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
6 { /tb/TOP/BLK(0)/II/Q(3) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
7 { /tb/TOP/BLK(0)/II/Q(3) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
8 { /tb/TOP/BLK(0)/II/Q(2) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
9 { /tb/TOP/BLK(0)/II/Q(2) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
10 { /tb/TOP/BLK(0)/II/Q(1) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\
11 { /tb/TOP/BLK(0)/II/Q(1) tb/TOP/VDD_top tb/TOP/VSS_top tb/TOP/VDD_1 tb/TOP/
VSS_1 }\\
```

isolation_association.rpt

Description

The `isolation_association.rpt` file captures the undriven/floating boundary port information.

Example Path

`$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
isolation_association`

Sample Report

```
# Number of potential locations for Instrumentation:  
set ISO_SIGNET_PORT(1) { { CORE1/out1 -policy PD1/isolation/iso1 -domain PD1 }  
{ -isolation_signal iso_en -isolation_sense high -clamp_value 1 -location parent  
}  
{ -instrumented_node CORE1/out1(XferHighConnC) -instance snps_PD1_isol_snps_out1  
_0_UPF_ISO} }  
  
#-----  
-----  
  
##### POWER DOMAIN /PD2 #####  
  
# Number of potential locations for Instrumentation:  
set ISO_SIGNET_PORT(2) { { CORE2/in1 -policy PD2/isolation/iso2 -domain PD2 }  
{ -isolation_signal iso_en -isolation_sense low -clamp_value 1 -location parent }  
{ -instrumented_node CORE2/in1(XferHighConnC) -instance snps_PD2_isol_snps_in1_0  
_UPF_ISO} }  
#
```

isolation_association_summary.rpt

Description

The `isolation_association_summary.rpt` file captures the details of the associated isolation cell instances and their associated isolation strategies. This report also captures the list of isolation cells that could not be associated with any isolation strategy in the UPF.

Example Path

`$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
isolation_association`

Sample Report

```
=====DB CELL ASSOCIATION SUMMARY=====
CellType : Number of Instances
=====
Isolation Cells : 0
Pure Level Shifter Cells : 0
Enable Level Shifter Cells : 0
Retention Cells : 0
Always On Cells : 0
Power Switch Cells : 0
=====

=====POLICY ASSOCIATION SUMMARY=====
# Policy Name, Policy Type, Associated DB Instance Count
set POLICY_MAPPING_LIST {
}
set TOTAL_POLICY_COUNT 0

#:UnAssociated dbInstances::::::::::
# DB Inst Name
set DB_UNASSOCIATED_LIST {
}

#:Associated dbInstances::::::::::
# DB Inst Name, Policy Name, Engine Type, Multiple DB
```

isolation_insertion_info.txt

Description

The `isolation_insertion_info.txt` file (virtual isolation insertion report) provides details such as signal being isolated and its corresponding isolation control, isolation sense, and clamp value. The data is dumped as a Tcl list in the report file.

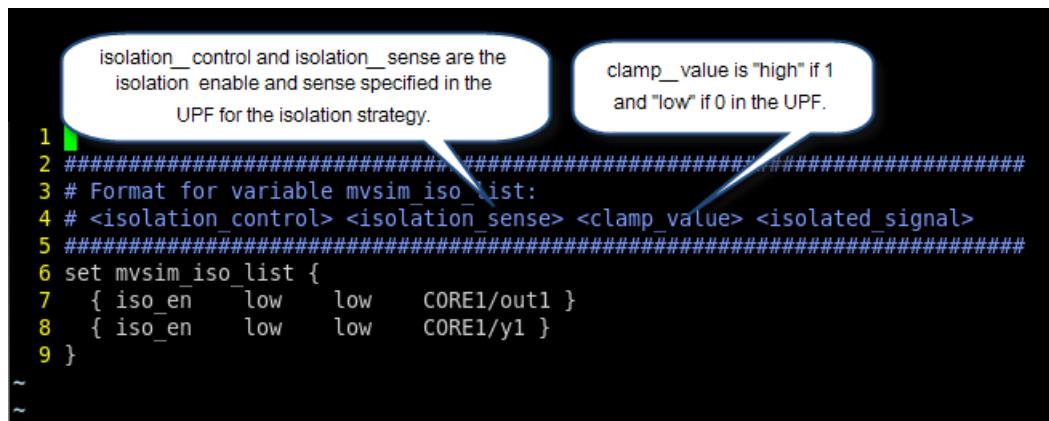
This report is dumped with the `-power=write_mvinfo` compile time option. This report can be used to compare the isolation inference report dumped by formality. Following is the reporting format:

```
<isolation_control> <isolation_sense> <clamp_value>
<isolated_signal>
```

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
isolation_insertion_info
```

Sample Report



isolation_control and isolation_sense are the isolation enable and sense specified in the UPF for the isolation strategy.

clamp_value is "high" if 1 and "low" if 0 in the UPF.

```
1 #####
2 #####
3 # Format for variable mvsim_iso_list:
4 # <isolation_control> <isolation_sense> <clamp_value> <isolated_signal>
5 #####
6 set mvsim_iso_list {
7   { iso_en    low    low    CORE1/out1 }
8   { iso_en    low    low    CORE1/y1 }
9 }
```

library_mapping.rpt

Description

The `library_mapping.rpt` file provides the cell-level and pin-level information read from `.db` per cell. This report file is generated by default in the `mvsim_native_reports` directory.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
library_match
```

Sample Report

```
3 Total Cells = 7
4
5
6 <----->
7 [Cell: DMELSR1]
8 Library = els1 : Library Path = libs/els1.db
9
10 Cell Id = 1 [Cell Type: Level-Shifter Cell]
11 Cell Function: OK
12 IsPowerAwareModuleCell: true
13 IsNameBasedMatchingDone: No
14 PortBasedCorruption: Not Set (Default)
15 is_soi: No
16 is_macro_cell: false
17
18 -----[PG PINS]-----
19 Pin --> VDD : Id = 1, Type = pg_type : primary_power
20   Direction in DB: Unknown , Resolved Direction: Input
21   Power Aware : true
22   Related Bias Pin --> VNW
23 Pin --> VSS : Id = 2, Type = pg_type : primary_ground
24   Direction in DB: Unknown , Resolved Direction: Input
25   Power Aware : true
26   Related Bias Pin --> VPW
27 Pin --> VDDO : Id = 3, [Type = pg_type : primary_power]
28   Direction in DB: Unknown , Resolved Direction: Input
29   Power Aware : true
30 Pin --> VPW : Id = 4, Type = pg_type : pwell
31   Direction in DB: Unknown , Resolved Direction: Input
32   Power Aware : true
33 Pin --> VNW : Id = 5, Type = pg_type : nwell
34   Direction in DB: Unknown , Resolved Direction: Input
35   Power Aware : true
36
37 -----LOGIC PINS-----
```

library_mapping_after_upf.rpt

Description

This file provides the cell and pin level information read from .db per cell and also includes the modifications to the liberty attributes done by the user in the UPF through the `set_port_attributes` calls.

This file provides power aware information of a cell in the following format. This reporting format is common to both
library_mapping.rpt and
library_mapping_after_upf.rpt.

```
Cell: DMAN2
Library = demo_std_10v_pg: Library Path = /u/regress/
p4_snapshots/TD/unit_LP/common_lib_db_behv_model/pgdb/
demo_std_10v_pg.db
Cell Id = 5 Cell Type: Normal Cell
Cell Function: AND
IsPowerAwareModuleCell: true
IsNameBasedMatchingDone: No
PortBasedCorruption: Enabled
is_soi: No
is_macro_cell: false
```

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/
library_match
```

Sample Report

```
3 Total Cells = 7
4
5
6 <----->
7 Cell: DMELSR1
8 Library = elsl : Library Path = libs/el1.db
9
10 Cell Id = 1      Cell Type: Level-Shifter Cell
11 Cell Function: OR
12 IsPowerAwareModuleCell: true
13 IsNameBasedMatchingDone: No
14 PortBasedCorruption: Not Set (Default)
15 is_soi: No
16 is_macro_cell: false
17
18 -----PG PINS-----
19 Pin --> VDD : Id = 1, Type = pg_type : primary_power
20   Direction in DB: Unknown , Resolved Direction: Input
21   Power Aware : true
22   Related Bias Pin --> VNW
23 Pin --> VSS : Id = 2, Type = pg_type : primary_ground
24   Direction in DB: Unknown , Resolved Direction: Input
25   Power Aware : true
26   Related Bias Pin --> VPW
27 Pin --> VDDO : Id = 3, Type = pg_type : primary_power
28   Direction in DB: Unknown , Resolved Direction: Input
29   Power Aware : true
30 Pin --> VPW : Id = 4, Type = pg_type : pwell
31   Direction in DB: Unknown , Resolved Direction: Input
32   Power Aware : true
33 Pin --> VNW : Id = 5, Type = pg_type : nwell
34   Direction in DB: Unknown , Resolved Direction: Input
35   Power Aware : true
36
37 -----LOGIC PINS-----
```

library_match.rpt

Description

The `library_match.rpt` file lists the source information for the cells and models, no matter whether the cell is matched or not. It also reports unmatched cell and reason.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
library_match
```

Sample Report

```
4  
5 <  
6 Cell: DMELSR1 : MATCHED  
7 DB ----> LibName: els1  
8 LibPath: libs/els1.db  
9 Model ----> FileName: demo_els.v  
10 Line Number: 2  
11  
12 <----->  
13 Cell: DMELSR2 : MATCHED  
14 DB ----> LibName: els2  
15 LibPath: libs/els2.db  
16 Model ----> FileName: demo_els.v  
17 Line Number: 12  
18  
19 <----->  
20 Cell: DMELSR3 : MATCHED  
21 DB ----> LibName: els3  
22 LibPath: libs/els3.db  
23 Model ----> FileName: demo_els.v  
24 Line Number: 22  
25  
26 <----->  
27 Cell: DMELSR4 : MATCHED  
28 DB ----> LibName: els4  
29 LibPath: libs/els4.db  
30 Model ----> FileName: demo_els.v  
31 Line Number: 32
```

The diagram shows two callout boxes pointing to specific lines in the report. The top box, labeled 'Source information for Verilog model.', points to line 9, which contains 'Model ----> FileName: demo_els.v'. The bottom box, labeled 'Source information for cell.', points to line 20, which contains 'Cell: DMELSR3 : MATCHED'.

repeater_port_association.rpt

Description

The `repeater_port_association.rpt` file is generated when the `set_repeater` command is present in the UPF. This file reports the ports where the repeater policy is applied and the total number of repeater elements inserted by the `set_port_attributes - repeater_supply` command.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
repeater_port_association
```

Sample Report

```
5 ### Port VS Repeater Policy Associated  
6 {  
7 { tb.DUT.InstDecode1.bot_mod1.a1 -rptr_policy tb/DUT/PD1/repeater/repeater_2 }  
8 { tb.DUT.InstDecode1.bot_mod1.b1 -rptr_policy tb/DUT/PD1/repeater/repeater_2 }  
9 { tb.DUT.InstDecode1.bot_mod1.c1 -rptr_policy tb/DUT/PD1/repeater/repeater_2 }  
10 { tb.DUT.InstDecode1.bot_mod1.d1 -rptr_policy tb/DUT/PD1/repeater/repeater_2 }  
11 { tb.DUT.InstDecode1.bot_mod1.iso_en -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
12 { tb.DUT.InstDecode2.bot_mod1.a1 -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
13 { tb.DUT.InstDecode2.bot_mod1.b1 -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
14 { tb.DUT.InstDecode2.bot_mod1.c1 -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
15 { tb.DUT.InstDecode2.bot_mod1.d1 -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
16 { tb.DUT.InstDecode2.bot_mod1.iso_en -rptr_policy tb/DUT/PD1/repeater/repeater_1 }  
17 }  
18  
19 ### Repeaters inserted by SPA -repeater_supply  
20 {  
21 }  
22  
23 set TOTAL_RPTR_ELEMENTS 10
```

srsn_spa_association.rpt

Description

The `srsn_spa_association.rpt` file reports information about the SRSN and SPA buffers associated with a port. This report also points to the node where the SPA/SRSN buffers are placed (lowconn or highconn depending on the models for which the policy is written). For SRSN, power and ground net of the corresponding port is reported. For SPA, driver or receiver supply information is reported.

When both SRSN/SPA are specified on the same port or if multiple SPA/SRSN are written on the same port, then depending on the precedence rules, policy having the highest precedence is reported in the `srsn_spa_association.rpt` file.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
srsn_spa_association
```

Sample Report

```
3 # File : mvsim_native_reports/srsn_spa_association.rpt.  
4 # This file is generated by MVSIM.  
5 # It contains SRSN and SPA buffers associated with a port.  
6  
7  
8 SIGNET_PORT(1) tb.dut_top_inst.counter_i.in_logic_vdd1 -instrumented_node  
    tb.dut_top_inst.in_logic_vdd1 -policy SRSN -scoped FALSE -power_net tb/v  
    spal -ground_net tb/VSSw  
9  
10 SIGNET_PORT(2) tb.dut_top_inst.counter_i.a -instrumented_node tb.dut_top_  
    inst.a -policy SPA -scoped FALSE -receiver_supply tb/spa_ss  
~
```

In this report, `-scoped TRUE/FALSE` indicates whether the SPA/SRSN is specified on the whole instance or on individual signals.

TRUE - SPA/SRSN is specified on the entire scope

FALSE - SPA/SRSN is specified on individual signals

upf_supply_set_power_state.rpt

Description

The `upf_supply_set_power_state.rpt` file reports information on the power states of all the supply sets (explicit and implicit) in the design.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
upf_supply_set_power_state
```

Sample Report

```
----- Supply Set Power State Details -----  
Generated by VCS-0-2018.09-Beta (Build Date = Jun 22 2018 20:26:24) on Fri Aug 10 01:58:41 201  
Copyright (c) 1991-2018 by Synopsys Inc.  
Description : This report has details on all the power-states of all Supply Sets in the design  
  
-----  
Supply Set : tb/dut_top_inst/TOP_prim  
States : TOP_HV TOP_HV_OFF TOP_DEFAULT TOP_USER_DEFAULT TOP_COA  
Supply Set : tb/dut_top_inst/PD1_prim  
States : PD1_HV PD1_HV_OFF PD1_DEFAULT PD1_USER_DEFAULT PD1_COA  
Supply Set : tb/dut_top_inst/domain_V2.primary  
States : PD2_COA
```

latch_inference.rpt

Description

The `latch_inference.rpt` file reports details of all the latches which are inferred using retention strategy.

Example Path

```
$VCS_HOME/doc/examples/NLP/Low_Power_Reports/  
latch_inference
```

Sample Report

```
2 RETENTION POLICY/MODULE MAPPING WHERE A POLICY IS BEING APPLIED
3 =====
4 Retention Policy: tb/top/A/retention/RET_1, Module : flop12
5 =====
6 Retention policy:tb/top/A/retention/RET_1
7 Inferred FF:out
8 Shadow FF:save____out
9 -----
10 Retention Policy: tb/top/A/retention/RET_1, Module : flop14
11 =====
12 Retention policy:tb/top/A/retention/RET_1
13 Inferred FF:out
14 Shadow FF:save____out
15 -----
16 Retention Policy: tb/top/A/retention/RET_1, Module : flop15
17 =====
18 Retention policy:tb/top/A/retention/RET_1
19 Inferred FF:out
20 Shadow FF:save____out
21 -----
22 Retention Policy: tb/top/A/retention/RET_1, Module : flop16
23 =====
24 Retention policy:tb/top/A/retention/RET_1
25 Inferred FF:out
26 Shadow FF:save____out
27 -----
28 Retention Policy: tb/top/A/retention/RET_1, Module : flop19
29 =====
30 Retention policy:tb/top/A/retention/RET_1
31 Inferred FF:out
32 Shadow FF:save____out
33 -----
```


8

Handling Corruption in Low Power Designs

The following sections describe VCS NLP corruption semantics for low power designs:

- [Simulation Semantics for Corruption](#)
- [Random Corruption](#)
- [Preventing Corruption of Memories During Power Shutdown](#)
- [Controlling the Corruption and Wake-up Semantics of the Initial and Always Blocks](#)
- [Treating Continuous Assignments as Pass Through](#)
- [CORRUPT_ON_ACTIVITY Simstate \(Low-Vdd Standby\)](#)
- [Using UPF_dont_touch Attribute](#)
- [Skipping Corruption](#)

- Corrupting Isolation Cell Output on Assertion/De-assertion of Isolation Enable
- Setting User-Defined Values for the Corrupted Real Data Type Signals
- Power Black Box

Simulation Semantics for Corruption

The following semantics apply when the domain simstate transitions to CORRUPT:

- All 2-state variables are corrupted to 0
- All 4-state variables are corrupted to X
- All the queued events including HDL forces are flushed (see the “[Overriding Corruption with HDL Forces](#)” section to configure the behavior of HDL forces during corruption)
- PLI/VPI/UCLI forces do not take effect
- All events scheduled in future time are flushed and no new events are allowed
- All the always blocks are disabled
- All the initial blocks and their child threads are terminated
- A domain turned off at time 0 enables corruption at time 0

Overriding Corruption with HDL Forces

Use the `-power=force_priority` compile-time option to override corruption with the HDL force. The following are the simulation semantics for HDL forces and corruption under `-power=force_priority`:

- Force happens in **NORMAL** state — When the signal goes to **CORRUPT** state, the forced value overrides corruption until it is released. On wake-up, force continues to override until it is released.
- Force is released in **CORRUPT** state — Back to the corruption value and normal simulation semantics apply.

*Table 8-1 Semantics for HDL Forces and Corruption under
-power=force_priority*

Current Value = A Forced Value = F Corruption Value = X	Simstate=NORMAL (Domain=ON)	Simstate=CORRUPT (Domain=OFF)	(CORRUPT -> NORMAL) (Domain=ON)
Force in NORMAL	A -> F	F	F (until released)
Force in NORMAL , release in CORRUPT	A -> F	F -> X	X

Note:

- If the force is active on a part (bit or part select) of the node or memory, corruption on the whole node or memory is skipped when the simstate goes from **NORMAL** to **CORRUPT**.
- The `-power=force_priority` option does not have any impact on the wire data type.

Random Corruption

By default, VCS NLP corrupts 4-state variables to x and 2-state variables to 0 in a power-managed module during shutdown.

The random corruption feature enables you to select a different random value for corruption for each shutdown event. If a power domain is shut down multiple times during simulation, the corruption value during each shutdown is selected randomly from the set of values you specify.

Usage

Use the following options to enable this feature in VCS NLP:

- `-pa_random_corrupt` compile-time option
- `-power < Tcl filename >` runtime option

Where,

`< Tcl filename >` — Tcl file which sets the `SNPS_random_corruption` attribute using the `set_design_attributes` command. You can use this command to set the corruption value to any of 0/1/x/z during shutdown. Use the following syntax:

```
set_design_attributes -elements {<power domain>} \  
-attribute SNPS_random_corruption <value>
```

You must specify one of the following values with the `SNPS_random_corruption` attribute:

- 0 - Corruption to 0

- 1 - Corruption to 1
- X - Corruption to X
- Z - Corruption to Z
- 01 - Randomly choose either 0 or 1 per shutdown event
- 01X - Randomly choose one of 0/1/X per shutdown event
- 01XZ - Randomly choose one of 0/1/X/Z per shutdown event

The values not in the above list are illegal.

Note:

- If X or Z is chosen randomly as the corruption value, 2-state variables are corrupted to 0.
- The hierarchical separator for a power domain specified with -elements is '/'.

You can also specify a seed using the `set_design_attributes` command. A seed is a number that initializes a pseudo-random generator used for corruption during shutdown. You can use it to recreate the same shutdown corruption pattern from different runs of the same test case. Use the following syntax to specify a seed:

```
set_design_attributes
-attribute SNPS_random_seed <value>
```

Key Points to Note

- The random seed applies to the entire design. Specifying random seed for each domain level using the `SNPS_random_seed` design attribute with `-elements` is not supported.

- Random corruption is applicable to elements that are corrupted based on domain simstate going to CORRUPT/CORRUPT_ON_ACTIVITY.
- Random corruption is not applicable to corruption on elements due to exception connections like set_related_supply_net, set_port_attributes.
- Random corruption is not applied for port-based corruption in case of macros.
- Random corruption is not applied on retention registers.
- Random corruption is not applied for corruption due to isolation/retention supplies being OFF.
- SNPS_random_corruption and SNPS_random_seed are the runtime design attributes.

Following are the only design attributes that can be used at runtime. The rest of the design attributes mentioned in this document are for compile time.

- SNPS_random_corruption
- SNPS_random_seed
- SNPS_treat_undriven_as_off
- SNPS_treat_undriven_as_undetermined
- SNPS_treat_undriven_supply_nets_as_undetermined
- SNPS_keep_unassociated_iso_on
- SNPS_power_switch_on_delay
- SNPS_voltage_based_supply_on

Examples

Compile:

```
% vcs -sverilog test.v -upf test.upf -pa_random_corrupt
```

Run:

```
% simv -power lpconfig.tcl
```

Examples for lpconfig.tcl file:

1. Random corruption on a power domain

```
set_design_attributes -elements \
{power_pa_tb/noRandomCorruption} \
-attribute SNPS_random_corruption 01X
```

2. Random corruption on the entire design

```
set_design_attributes -attribute \
SNPS_random_corruption 01
```

3. Applying a random seed on the entire design

```
set_design_attributes -attribute SNPS_random_seed 37
```

Note:

- Enum variables, when corrupted with 0, take the value 0. When they are corrupted to 1, they take the value of all ones (32'hffff_ffff).
- VCS NLP supports random corruption of all individual bits of a vector and all the vectors in a given domain. That is, all the bits of a vector and different vectors gets the values which are independent of each other. For example, as shown in [Table 8-2](#).

Table 8-2 Randomized Corruption of Vectors and Individual Bits of a Vector

a[2:0]	b[2:0]
3'b110	3'b01x
3'b001	3'b1z1
3'b010	3'bx1x
3'b001	3'b11z

Limitation

Following is the limitation of this feature:

- Vector/Bit-level randomization is supported only for the Verilog portion of the design. It is not applicable to the VHDL elements.

Preventing Corruption of Memories During Power Shutdown

Use the `-power=dont_touch_mem` compile-time option to prevent the corruption of memories that are initialized using `$readmemh*` and have no other structural drivers.

```
% vcs -sverilog -upf <upf_file> file_name.sv  
-power=dont_touch_mem
```

Limitations

Following is the limitation of this feature:

- Partial assignment (where only some locations of memory are initialized with `$readmemh`) is not supported.

Controlling the Corruption and Wake-up Semantics of the Initial and Always Blocks

You can use the attributes listed in [Table 8-3](#) to control the corruption and wake-up semantics of the initial and always blocks. You must use the `-power=attributes_on` compile-time option to enable these attributes.

Table 8-3 List of Attributes Used to Control the Corruption and Wake-up Semantics of the Initial and Always Blocks

Attribute	Description
vcs_reinit	<p>Allows you to retrigger the selected initial blocks on power-up without using the UPF commands.</p> <p>You can mark an initial block by specifying <code>(* vcs_reinit *)</code> attribute in the RTL. For example, as shown below:</p> <pre>(* vcs_reinit *) initial begin:block_reinit #5 mem[0] = 8'b00000001; mem[1] = 8'b00000010; mem[2] = 8'b00000100; mem[3] = 8'b00001000; end</pre>
vcs_dont_reinit	<p>Disables the retriggering of the selected initial block after power-up.</p> <p>Specify <code>(* vcs_dont_reinit *)</code> before an initial block in the RTL. For example, as shown below:</p> <pre>(* vcs_dont_reinit *) initial begin:block_reinit #5 mem[0] = 8'b00000001; mem[1] = 8'b00000010; mem[2] = 8'b00000100; mem[3] = 8'b00001000; end</pre> <p>This attribute does not retrigger the selected initial block on power-up even if it is marked for reinit using the <code>SNPS_reinit</code> design attribute in the UPF.</p>

Attribute	Description
vcs_always_on	<p>Marks the selected always block as always-on.</p> <p>Specify (* vcs_always_on *) before an always block in the RTL. For example, as shown below:</p> <pre data-bbox="833 536 1274 642">(* vcs_always_on *) always @ (posedge clk) begin q <= d; end</pre> <p>The above always block is marked as always-on, and is not affected by power-down. No corruption is observed on this part of code when the domain is powered off.</p>
vcs_dont_trigger_on_wakeup	<p>Disables the automatic retriggering of the selected always block on power-up.</p> <p>Specify (* vcs_dont_trigger_on_wakeup *) before an always block in the RTL. For example, as shown below:</p> <pre data-bbox="833 1079 1348 1216">(* vcs_dont_trigger_on_wakeup *) always @ (ISO_IN, ISO_OUT) begin OUT = ISO_IN & ISO_OUT; end</pre>

Treating Continuous Assignments as Pass Through

VCS NLP treats the following statements as pass through and skips corruption:

- Simple Verilog continuous assignments (assign a=b)

- buf primitives
 - Simple VHDL signal assignments (`temp1 <= temp2`)
 - Signal assignments with composite data type (records)
 - Simple assignment in VHDL procedures
 - Assignments with simple concatenations
-

Limitations

Following are the limitations of this feature:

- Only corruption is skipped on Verilog `buf` primitives
- The following are not treated as pass through for isolation analysis:
 - Assignments inside VHDL process statements
 - VHDL signal assignments with delay

Configuring Semantics for Runtime Forces During Power Shutdown and Wake-up

VCS NLP supports the following modes to configure force semantics for runtime forces:

- Force Priority
- Force and Corrupt
- Flush Forces

- [Defer Forces](#)

Key Points to Note:

- This feature is supported for simstate CORRUPT only. It is not supported for any other simstates with corruption semantics.
- The above force semantics for runtime forces are applicable for:
 - The design as a whole. There will be no granularity with respect to the design instances, signals or power domains.
 - All types of runtime forces (PLI, VPI (both deposit and vpiForceFlag), UCLI, and so on) follow the same semantics.

Force Priority

- Force happens in NORMAL state — When the signal goes to the CORRUPT state, the forced value overrides the corrupt value until released. On wake-up, the force continues to override until released.
- Force happens in CORRUPT state — Forced value overrides corrupt value. On wake-up, the force continues to override until released.
- Force is released in CORRUPT state — Back to corrupt value and normal simulation semantics apply.

See [Table 8-4](#) for more information.

Force and Corrupt

- Force happens in NORMAL state — When the signal goes to the CORRUPT state, VCS NLP corrupts it, but keeps the forced state. On wake-up, the force overrides with the corrupt value until it is released.
- Force happens in CORRUPT state — VCS NLP continues with the corrupt value, but keeps the forced state. On wake-up, the force overrides with the corrupt value until it is released.
- Force is released in CORRUPT state — Forced state -> Normal simulation semantics.

Note:

Forced state is a state that the simulator tracks for each signal to determine whether a force is active or not.

See [Table 8-4](#) for more information.

Flush Forces

- Force happens in NORMAL state — When the signal goes to the CORRUPT state, VCS NLP releases force and continues with the corruption semantics.
- Force happens in CORRUPT state — Force is ignored and an error message is issued.
- Force is released in CORRUPT state — Not applicable

See [Table 8-4](#) for more information.

Defer Forces

- Force happens in NORMAL state — When the signal goes to the CORRUPT state, VCS NLP corrupts it, but keeps the forced state. On wake-up, the signal is forced with the earlier forced value until it is released.
- Force happens in CORRUPT state — VCS NLP continues with the corrupt value, but keeps the forced state. On wake-up, the force overrides the corrupt value until it is released.
- Force is released in CORRUPT state — Forced state -> Normal simulation semantics.

See [Table 8-4](#) for more information.

Table 8-4 Force Semantics for Corrupt

Option	Current Value = A Forced Value = B	NORMAL State (Domain=ON)	CORRUPT State (Domain=OFF)	WAKEUP (Domain=ON)
Force Priority (Continue with force)	Force at NORMAL	A -> B	B	B (until released)
	Force at CORRUPT	A	X-> B	B (until released)
	Force released at CORRUPT	A -> B	B -> X	X
Force and Corrupt (Corrupt, but keep the forced state)	Force at NORMAL	A -> B	X	X (until released)
	Force at CORRUPT	A	X	X (until released)
	Force released at CORRUPT	A -> B	X	X
	Force at NORMAL	A -> B	X	X

Option	Current Value = A Forced Value = B	NORMAL State (Domain=ON)	CORRUPT State (Domain=OFF)	WAKEUP (Domain=ON)
Flush Forces (Release force and corrupt)	Force at CORRUPT	A	X	X
	Force released at CORRUPT	Not Applicable	Not Applicable	Not Applicable
Defer Forces (Corrupt on power down and restore forced value on power up)	Force at NORMAL	A->B	X	B (until released)
	Force at CORRUPT	A	X	B (until released)
	Force released at CORRUPT	A->B	X	X

Use Model

To enable the configurations mentioned above, you must specify the following UPF command in the `upf` file:

```
set_design_attributes -attribute
SNPS_force_semantics <CONFIGURATION NAME>
```

Where, `CONFIGURATION NAME` can be one of the following:

FORCE_PRIORITY	FORCE_AND_CORRUPT
----------------	-------------------

FLUSH_FORCES	DEFER_FORCES
--------------	--------------

Limitations

- This feature is currently limited to the runtime forces only (Forces which appear during runtime by UCLI/VPI).
- Language forces are not supported.
- Forces on SV variables of type real are not supported.
- Forces on SV variables of type longint are not supported.
- Forces on Memories and MDAs are not supported.
- While forces on Verilog part of the design in a mixed design are supported, the forces on VHDL signals (either in mixed design or pure VHDL design) are not supported.
- Forces on ports which have exception supply connections using `set_port_attributes` or `set_related_supply_net` are not supported.
- This feature is supported for simstate CORRUPT only. It is not supported for any other simstates with corruption semantics.

CORRUPT_ON_ACTIVITY Simstate (Low-Vdd Standby)

In simstate, CORRUPT_ON_ACTIVITY, the active state of nets and state elements driven by an element remain unchanged at the transition. The processes modeling the behavior of the element remain enabled for activation (evaluation). Any net or state element that is actively driven after transitioning to this state is corrupted.

-simstate of `add_power_state` command supports CORRUPT_ON_ACTIVITY.

Examples

- Using Supply Expression

```
add_power_state Island_V1.primary \
-state V1_COA {-supply_expr {power == `{FULL_ON,0.7}} -simstate CORRUPT_ON_ACTIVITY}
```

- Using Logic Expression

```
add_power_state Island_V1.primary \
-state V1_COA {-logic_expr {psw_ctrl == 1} -simstate CORRUPT_ON_ACTIVITY}
```

Simulation Behavior

Table 8-5 describes the simulation behavior of CORRUPT_ON_ACTIVITY.

Table 8-5 Simulation Behavior of CORRUPT_ON_ACTIVITY

Construct	Behavior	
Sequential Logic (always/process)	Active clock edge	Data corrupted
	Clock Stable, Change on Data	No corruption
	Async reset active edge	Data corrupted
	Clock Stable, Sync reset active	No corruption

Construct		Behavior
Combinatorial Logic	Change on sensitivity list elements of always/process block	LHS corrupted
	Cont-assign, Signal Assignments	LHS corrupted only if change on RHS results in a change on LHS
	Primitive Gates	Output corrupted only if change on inputs result in a change on output
Assignments/Gates with Delays	Change on RHS/inputs	Delay ignored for corruption
UDPs	Change on Inputs	Corrupted

Limitations

- Corruption is not applied for generating Verilog instances under VHDL.
- In case of gates with delays, output is not corrupted if the change on output in `CORRUPT_ON_ACTIVITY` is due to a future event scheduled from `NORMAL` simstate.
- Corruption of bit/part selects is not supported.

Using UPF_dont_touch Attribute

The `UPF_dont_touch` attribute is used to exclude certain sections of the design from being power managed. This can be applied only at a module level or to specific elements of a module. This attribute supports Always ON behavior. It indicates that the signals are “always on” during power down and are simulated as an ON logic.

Use Model

You can use the `UPF_dont_touch` attribute at compile-time, as shown below, on a module or module signals transitively. For modules, all its instances and the hierarchy under the module instance are marked as `UPF_dont_touch` transitively.

```
set_design_attributes -attribute UPF_dont_touch TRUE  
                      -models <module list>  
                      -elements <signals>
```

Key Points to Note

- The instances of the `UPF_dont_touch` model does not get corrupted during shut down.
- When a model is marked as `UPF_dont_touch`, the model and its child instances are marked to be non-power managed.
- If any child instance under a module marked `UPF_dont_touch` is partitioned into different power domain, the `UPF_dont_touch` attribute does not apply to that child instance.
- Any specific pin of the `UPF_dont_touch` model cannot be marked as `UPF_dont_touch FALSE`.
- Initial block retriggering does not replay the initial blocks in the `UPF_dont_touch` model.

Backwards Compatibility

For backwards compatibility, use the `-power=UPF_dont_touch_BC` compile-time option.

For VCS releases prior to H-2013.06, VCS NLP skips instrumentation for corruption, retention, and isolation on the `UPF_dont_touch` module and its descendant instances.

Starting with the VCS H-2013.06 release, VCS NLP supports instrumentation for isolation and retention on the UPF_dont_touch modules. It does not support instrumentation for corruption. You can use the `-power=UPF_dont_touch_BC` option to avoid corruption, retention, and isolation instrumentation for the UPF_dont_touch modules.

Module Support

The following points describe the module support:

- The recommended usage of UPF_dont_touch is on leaf level modules.
- Instances of the UPF_dont_touch model do not get corrupted during the shut down.
- Isolation and retention are inserted in the dont_touch module.
- set_related_supply_net/set_port_attributes on the ports of the dont_touch module is ignored for corruption.
- When a model is marked as don't touch, the model and its child instances, if any, are also marked as don't touch.
- Any specific pin of the UPF_dont_touch model cannot be marked as UPF_dont_touch FALSE.
- Initial block retriggering does not replay the initial blocks in the UPF_dont_touch model.

Module Named Block Support

- You can mark a complete named block (for example, always/process block) as UPF_dont_touch using -elements.

- All the signals modified in the always block are disabled for shutdown corruption.
 - Signals are active even during shutdown so that the value propagation occurs.
-

Module Signal Support

- Signals from the always block can be selectively marked as UPF_dont_touch.
 - Corruption is not performed on the signal due to power activity.
 - set_related_supply_net/set_port_attributes on the port is ignored for corruption.
 - UPF_dont_touch marking does not impact isolation insertion, retention, and level-shifter inference.
-

Support for a Module Inside a Library

- You can specify the UPF_dont_touch model selectively from a particular library using the -models option of set_design_attributes.
- The library name must be prefixed to model name as lib.model.

Example: set_design_attributes -attribute UPF_dont_touch TRUE -models {lib.buffer}

Usage Examples

Model Level Support

```
set_design_attributes -models LEVEL_LEAF -  
attribute UPF_dont_touch TRUE
```

Model Signal Level Support

```
set_design_attributes -models LEVEL_LEAF -elements  
{sig1 sig2} -attribute UPF_dont_touch TRUE
```

Limitations

- The `-elements <element_list>` attribute without `-models` is not supported.
- Instance level and hierarchical signal level `UPF_dont_touch` is not supported.
- Specifying bit/part selects with `-elements` is not supported.
- Transitive property for a module level attribute applies as long as the child instance is in the same power domain as the parent that is marked `UPF_dont_touch`.

If the child is partitioned in a different power domain, the instance and the hierarchy under it is not marked `UPF_dont_touch` transitively.

Skipping Corruption

You can use the `-power=apfcompat` compile-time option to:

- Skip corruption of reals
- Skip corruption of constants
- Skip corruption of signals assigned only inside initial blocks

Note:

The `-power=apfcompat` option is supported only for Verilog designs.

You can use the `SNPS_apf_compat` design attribute at runtime to:

- Skip the corruption of VHDL variables and real types
- Corrupt integers to 0

Following is the syntax:

```
set_design_attributes -attribute SNPS_apf_compat TRUE
```

You can use the `SNPS_corrupt_vhdl_integer` design attribute at runtime to skip the corruption on VHDL integer. Following is the syntax:

```
set_design_attributes -attribute  
SNPS_corrupt_vhdl_integer NO_CORRUPT
```

This attribute can take one of the following:

`LEFT_BOUND`, `RIGHT_BOUND`, `ZERO`, or `NO_CORRUPT`.

To use these design attributes, pass the attribute through the power configuration file at runtime, as shown in the following example:

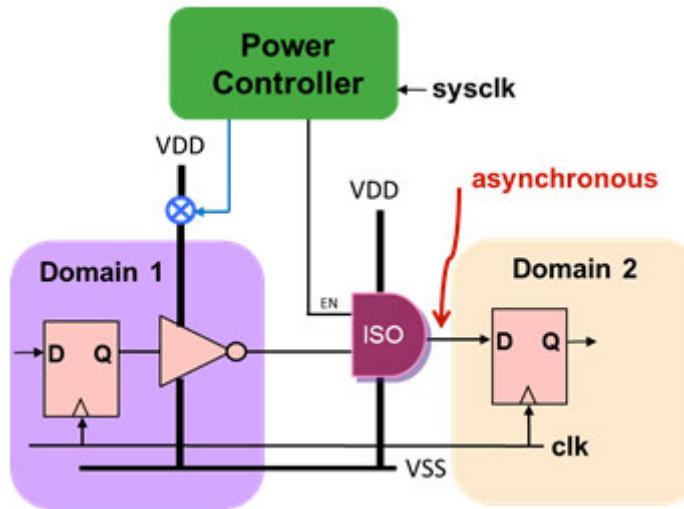
config.tcl:

```
set_design_attributes -attribute SNPS_apf_compat TRUE  
set_design_attributes -attribute  
SNPS_corrupt_vhdl_integer NO_CORRUPT  
% simv -power config.tcl
```

The SNPS_corrupt_vhdl_integer attribute has higher precedence than the SNPS_apf_compat attribute.

Corrupting Isolation Cell Output on Assertion/De-assertion of Isolation Enable

In a low power design, isolation enable control signal is often asynchronous with respect to the clock of the source or destination power domain, as shown in the following circuit:



Therefore, the resulting circuit may exhibit metastability during both isolation enable and isolation disable. A signal transitioning asynchronously due to the assertion or de-assertion of isolation enable might cause unexpected transitions in the downstream logic and trigger the logic incorrectly due to the metastability.

To verify that the downstream logic is not affected by the metastability caused by a toggle on the isolated signal when the isolation enable signal is asserted or de-asserted, apply corruption (X) on the outputs of all the isolation cells associated with the given design instance(s) for a certain amount of time using the SNPS_cdc_slow_clock design attribute. Following is the syntax of SNPS_cdc_slow_clock:

```
set_design_attributes -elements <Instances>
[-transitive true|false] -attribute
SNPS_cdc_slow_clock {<clock_signal>
number_of_edges}
```

Where, the name of Instances and clock_signal is the hierarchical name with respect to the power top specified.

The default value of number_of_edges is 2.

To specify a time delay, you can use the SNPS_cdc_clock_delay design attribute in the above syntax. This attribute controls the duration for which the isolation cell output drives an X before it starts driving the source value. Following is the syntax of SNPS_cdc_clock_delay:

```
set_design_attributes -elements <Instances>
[-transitive true|false] -attribute
SNPS_cdc_clock_delay <time>
```

Where, the name of Instances is the hierarchical name with respect to the power top specified, and time is specified as the absolute time in the range 0 to 2147483647.

The time units are based on the resolved timescale. For example, if you specify value as 10 in SNPS_cdc_clock_delay, then for a timescale of 1ns/1ps the isolation cell output drives an X for 10000ps.

SNPS_cdc_slow_clock will have higher precedence than SNPS_cdc_clock_delay when both these attributes are specified on the same element. VCS NLP generates a warning message for the following example code:

```
set_design_attributes -elements {u2/inst_unit1/buf_inst1}
```

```
-transitive TRUE -attribute SNPS_cdc_clock_delay {10}
set_design_attributes -elements {u2/inst_unit1/buf_inst1}
-transitive TRUE -attribute SNPS_cdc_slow_clock {clk 2}
```

Following is the sample warning message:

```
Warning-[POWERCONFIG_DCDCIF] Duplicate CDC instance Found
config, 3
CDC instance 'u2/inst_unit1/buf_inst1' specified in option
'-elements' for 'set_design_attributes SNPS_cdc_slow_clock'
has already been specified earlier. Attribute
SNPS_cdc_slow_clock will have higher precedence than
SNPS_cdc_clock_delay.
Please ensure an instance is specified only once either using
SNPS_cdc_slow_clock or SNPS_cdc_clock_delay.
```

Use Model

Use the following syntax in the compile time configuration file (for example, lp.cfg) to apply corruption (x) on the outputs of all the isolation cells associated with the given design instance(s) for a certain amount of time:

```
set_design_attributes -elements <Instances>
[-transitive true|false] -attribute
SNPS_cdc_slow_clock {<clock_signal>
number_of_edges}

set_design_attributes -elements <Instances>
[-transitive true|false] -attribute
SNPS_cdc_clock_delay <time>
```

For a given power domain, you can use the SNPS_disable_cdc_corruption design attribute in the compile time configuration file to specify the list of isolation strategies on which you want to disable corruption. Following is the syntax of SNPS_disable_cdc_corruption:

```
set_design_attributes -elements  
{power_domain_name} -attribute  
{SNPS_disable_cdc_corruption} {space separated  
list of isolation strategies}
```

Where, power_domain_name is the hierarchical name of the power domain with respect to the power top specified, and space separated list of isolation strategies is the simple name of the isolation strategy.

You can use the design attribute SNPS_cdc_corrupt_on_iso_inactive FALSE in the compile time configuration file to disable the SNPS_cdc_slow_clock and SNPS_cdc_clock_delay attributes.

Specify the power configuration file in the vcs command line using the -power_config <config_file> option as follows:

```
% vcs <options> -upf <upf_file> -power_config  
lp.cfg
```

Example

Consider the following UPF:

```
set_design_top testbench/inst  
create_power_domain Island_VDD  
create_power_domain Island_V1 -elements inst1  
create_power_domain Island_V2 -elements inst2
```

```

set_isolation V1_ISO -domain Island_V1 -isolation_power_net
V2 -isolation_ground_net VSS -elements {inst1/out}
-clamp_value 0 -isolation_signal iso_sig -isolation_sense
low -location self

set_isolation V2_ISO -domain Island_V2 -isolation_power_net
V2 -isolation_ground_net VSS -elements {inst2/out}
-clamp_value 1 -isolation_signal iso_sig -isolation_sense
low -location self

```

Consider the following power configuration file code:

```

set_design_attributes -elements {inst1} -attribute
SNPS_cdc_slow_clock {clk_s 1}
set_design_attributes -elements {inst2} -attribute
SNPS_cdc_clock_delay 20
set_design_attributes -elements {Island_V1} -attribute
{SNPS_disable_cdc_corruption} {V1_ISO}

```

Here, the attribute `SNPS_cdc_slow_clock` is disabled for `inst1`, and only the attribute `SNPS_cdc_clock_delay` is applied for `inst2`.

This information is also dumped in `mvsim_native_reports/isolation_association.rpt`, as follows:

```

set ISO_SIGNET_PORT(1) { {inst2/out -policy Island_V2/
isolation/V2_ISO -domain Island_V2}
{-isolation_signal iso_sig -isolation_sense low
-clamp_value 1 -location self}
{-instrumented_node inst2/out -delay 20} }

```

If the power configuration file contains the `SNPS_cdc_corrupt_on_iso_inactive` attribute, as follows:

```

set_design_attributes -elements {inst1} -attribute
SNPS_cdc_slow_clock {clk_s 1}
set_design_attributes -elements {inst2} -attribute
SNPS_cdc_clock_delay 20

```

```
set_design_attributes -elements {Island_V1} -attribute  
{SNPS_disable_cdc_corruption} {V1_ISO}  
set_design_attributes -attribute  
SNPS_cdc_corrupt_on_iso_inactive FALSE
```

then both `SNPS_cdc_slow_clock` and `SNPS_cdc_clock_delay` are ignored.

Setting User-Defined Values for the Corrupted Real Data Type Signals

By default, VCS NLP forces all the signals with real data type to 0.0 during shutdown. If these real type corrupted signals drive the modules in “ON” domain, it is difficult to differentiate whether 0.0 is the default value or the corrupted value. VCS NLP can consider 0.0 as a driven value and give wrong results.

During shutdown, VCS NLP allows you to set the desired real value (either positive or negative) for the corrupted real data type signals using the following options:

- `-power=pa_real_corrupt` compile-time option
- `-power <Tcl_config_file>` runtime option

You must specify the `SNPS_real_number_corruption` design attribute in the UPF at compile-time or the Tcl configuration file at runtime, to set a unique value for real data types under corruption. Following is the syntax:

```
set_design_attributes -elements {<power domain>}  
-attribute SNPS_real_number_corruption  
<unique_real_value>
```

Example:

```
set_design_attributes -elements {power_pa_tb/  
REAL_PD} -attribute SNPS_real_number_corruption  
-1234567
```

Note:

- The hierarchical separator for the power domain specified with `-elements` is “/”.
- VCS NLP applies the real corruption value to the entire design if `-elements` is not specified.
- If the **compile-time** `set_design_attributes` setting in the UPF is not same as the **runtime** `set_design_attributes` setting in the Tcl configuration file, the runtime `SNPS_real_number_corruption` attribute setting overrides the compile-time attribute setting. For example, consider the following commands:

Compile-time `set_design_attributes` command:

```
set_design_attributes -attribute  
SNPS_real_number_corruption -2222
```

Runtime `set_design_attributes` command:

```
set_design_attributes -attribute  
SNPS_real_number_corruption -1111
```

In this case, the runtime attribute setting overrides the compile-time attribute setting and applies the real value `-1111` to the entire design.

- The following section describes the precedence details when the compile-time `set_design_attributes` setting in the UPF is different from the runtime `set_design_attributes` setting in the Tcl configuration file.

Precedence Details

Case-1: The `set_design_attributes` command with `-elements` specified at compile-time and the `set_design_attributes` command without `-elements` specified at runtime.

Consider the following commands:

Compile-time `set_design_attributes` command:

```
set_design_attributes -elements {Island_V1} -attribute  
SNPS_real_number_corruption -2222
```

Runtime `set_design_attributes` command:

```
set_design_attributes -attribute  
SNPS_real_number_corruption -1111
```

In this case, the runtime setting does not override the compile-time setting for the specified power domain. The runtime setting applies the real value `-1111` to the entire design except `Island_V1`. The real value `-2222` is applied to `Island_V1`.

Case-2: The `set_design_attributes` command without `-elements` specified at compile-time and the `set_design_attributes` command with `-elements` specified at runtime.

Consider the following commands:

Compile-time set_design_attributes command:

```
set_design_attributes -attribute  
SNPS_real_number_corruption -1111
```

Runtime set_design_attributes command:

```
set_design_attributes -elements {Island_V1} -attribute  
SNPS_real_number_corruption -2222
```

In this case, the runtime power domain setting overrides the compile-time setting for the same power domain. The compile-time setting applies the real value -1111 to the entire design except Island_V1. The runtime setting applies the real value -2222 to Island_V1.

Key Points to Note

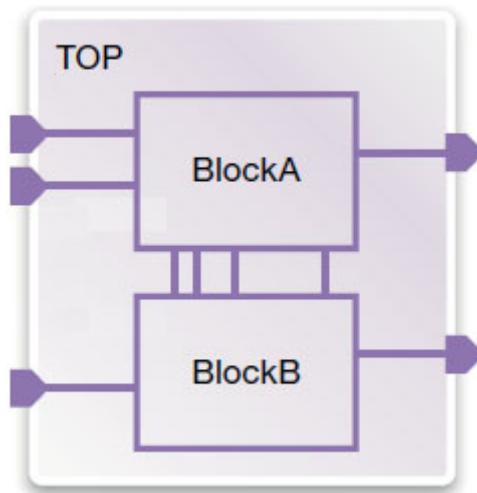
- This feature is not supported for the following:
 - Corruption on elements due to exception connections like `set_related_supply_net` and `set_port_attributes`
 - Port-based corruption in case of macros
 - Retention registers
 - Corruption due to isolation and retention supplies being OFF
 - Scope names in the `-elements` list
- VCS NLP issues the following runtime messages if you use the `SNPS_real_number_corruption` design attribute:
`LP_DOMAIN_REAL_CORRUPT`, `LP DESIGN REAL CORRUPT`

Limitations

- Wildcards are not supported in the domain names.

Power Black Box

In a non-power-aware hierarchical design flow, sub-blocks are synthesized and verified independently. When these sub-blocks are used at the top level of the design, as shown in the following figure, they are treated as black boxes.

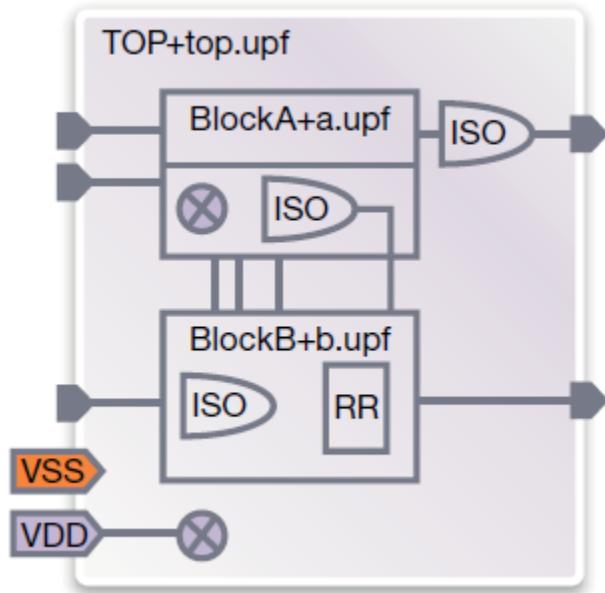


In a power-aware hierarchical design flow, when UPF is applied to a design, it can change the behavior of sub-blocks in the design. For example, if a single top level UPF file adds isolation and retention to lower level sub-blocks, the independent verification of the sub-blocks is invalidated.

If a sub-block is to be black boxed, it cannot reference objects outside of its own hierarchy and the proper power constraints must be captured to allow complete verification at the parent level.

The `set_power_black_box` configuration file option allows you to apply UPF at each block which will be verified independently to allow hierarchical sub-block verification. This block level UPF approach allows VCS NLP to capture all of the power constraints that apply to the sub-blocks and ensures consistency across block level and top level runs.

Figure 8-1 Hierarchical UPF Applied to Each Block



Use Model

Specify list of modules that are being black boxed in the power configuration file (for example, `lp.cfg`). Following is the syntax:

```
set_power_black_box -models {space separated list  
of black box modules}
```

Specify the power configuration file in the vcs command line using the `-power_config <config_file>` option as follows:

```
% vcs <options> -upf <upf_file> -power_config  
lp.cfg
```

UPF Requirements

- Domain Partitioning

The top scope of a black box should have unique power domain; different from the top level and all other black boxes:

```
set_scope <bbox_inst>  
create_power_domain <domain name> -include_scope
```

- Boundary Constraints

You must correctly determine the related supplies for each port of the block.

```
set_port_attributes -port {< black box ports >}  
-receiver_supply {receiver_supply_set} -  
driver_supply {driver_supply_set}
```

- Supply Network

All the supplies required to define the power model of the block must be available at the top scope in the block.

Supplies at hierarchies lower than the top scope of black box are ignored.

Top level UPF must not contain power network connectivity information (CSNs and so on) for cells or supplies within the black box.

- Strategies

Location parent strategies are not allowed in black box level UPF.

Isolation strategies using -location fanout should not be used if they span across blocks or block and top.

Top level UPF must not contain strategies that require isolation insertion within a black box.

Boundary Constraints

- `set_port_attributes` (SPA) is required for all ports of the black box.
- Same SPA commands should be used for top level and block verification.
- VCS NLP uses the following:
 - Receiver supply for the input port or the driver supply for the output port for the top level simulation.
 - Driver supply for the input port or the receiver supply for the output port for the block level simulation.

UPF Commands

Command	Description
create_power_domain	<p>This command is supported inside the black box if it defines the power domain at the top scope of the block and does not use any elements from inside the black box.</p> <p>The exception for this rule is the top power domain of the block. If <code>create_power_domain</code> defines the top domain of the black box and also include the elements from inside the black box, VCS NLP creates the top power domain for the block and ignores and warn about the elements inside the block.</p>
create_supply_net create_supply_set create_supply_port	VCS NLP creates supply net/set/port if it is created at the top scope of the black box, otherwise it will warn and ignore.
connect_supply_net	VCS NLP supports this command if the net and port are at the top scope of the black box, otherwise it will warn and ignore.
create_power_switch	This command is supported inside the black box if it is created at the top scope of the black box.
set_domain_supply_net	This command is supported inside the black box if the power domain is created and the supplies are scoped at the top of the black box.
associate_supply_set	This command is supported inside the black box if both the arguments are scoped at or above the black box, otherwise VCS NLP ignores and warn about it.
set_isolation set_isolation_control	These commands are supported inside the black box if the domain, isolation signal, and supplies exist, otherwise VCS NLP ignores and warn about it.
set_retention set_retention_control	Retention related commands for the domains in black boxes are ignored with warning.

Command	Description
create_pst	This command is supported inside the black box if it is at the top scope of the black box. Only the supplies at the top scope of the black box will be retained. Supplies used in PST, which are nested inside the black box, are ignored and warned.
add_pst_state	This command is supported inside the black box if the PST is created, otherwise VCS NLP ignores and warn about it.
add_port_state add_power_state	These commands are supported inside the black box if the ports are created, otherwise VCS NLP ignores and warn about it.
find_objects	The <code>find_objects</code> command will work to find the pins of the black box.
create_logic_port create_logic_net connect_logic_net	These commands are supported at the black box boundary. These commands allows you to create logic ports on black box boundary and connect the logic nets to them from top.
set_design_attributes	Attribute commands sets the attributes if the object arguments for these commands are created, otherwise VCS NLP ignores and warn about them.

9

Isolation Support

This chapter describes the VCS NLP isolation support in the following sections:

- Isolation Support for Mixed Designs
- Isolation Support for SystemVerilog Design Constructs
- Skipping Isolation on the Undriven/Floating Ports
- Inserting NOR-Style Isolation Cells as per UPF 3.0 LRM
- Support for Conservative diff_supply_only Isolation
- Name-Based Isolation Cell Association in Netlist
- Support for Isolation Terminal Points at the Hierarchical Boundaries
- Path-Based Isolation Support for Heterogeneous Loads

- Support for -exclude_elements and -update Options
- Support for -applies_to with -elements for Isolation
- Support for Specifying Instances with set_isolation -elements
- Fanout Isolation Instrumentation Support for Signals of SV Interface/Modport
- Generating Virtual Isolation Reports
- Isolation/Corruption on Ports with Logical Expressions in Port Map
- Supporting Lower Domain Boundary for Isolation
- Implementing Assertion Checks for Isolation Cells
- Updating Isolation Supplies
- Specifying Isolation Strategies Without Defining Isolation Enable

Isolation Support for Mixed Designs

VCS NLP supports all VHDL data types for isolation.

Limitations

Following are the limitations of isolation support for mixed designs:

- Isolation value for enum is always `left value
- Isolation is not applied for a crossover between VHDL record and SystemVerilog struct type
- Isolation is not applied for a crossover between VHDL integer type and Verilog reg type

- Isolation is not applied if there is a logic expression in port map
- The following specifications in -elements are not supported:
 - Part selects
 - Member of a record
 - Element of MDA

Isolation Support for SystemVerilog Design Constructs

VCS NLP supports virtual isolation of the following SystemVerilog Design (SVD) constructs:

- logic, int, struct, enum, reg, shortint, and union
- multidimensional arrays

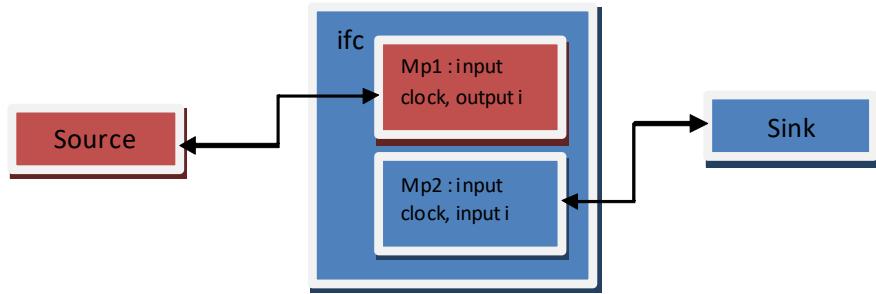
Limitations

Following are the limitations of this feature:

- Real data type is not supported
- Supply0, supply1, and tri are not supported

Isolating Interface Modports

You can isolate input/output of interface modports in SVD designs. This feature is enabled by default.



Limitations

- Isolation on interfaces without modports do not work as there is no direction associated with the signals
- Only `-self` and `-parent` locations are supported
- Clamp value `0/1/z` are supported
- Simple directional ports in the interfaces are not supported

Skipping Isolation on the Undriven/Floating Ports

VCS NLP isolates the undriven/floating boundary ports if the isolation strategy is specified on them.

You can use the `-power=dont_isolate_floating_nets` compile-time option to skip isolation on the undriven/floating boundary ports. VCS NLP captures the undriven/floating boundary port information in the `isolation_association.rpt` file generated in the `mvsim_native_reports` directory.

Following is the sample report of the isolation_association.rpt file:

```
# Number of potential locations for Instrumentation: 1
set ISO_SIGNET_PORT(1) { { dut/core/in -policy PD_AAA/
isolation/iso_in -domain PD_A }
{ -isolation_signal ISO_ENA -isolation_sense low -
clamp_value 0 -location parent }
{ -instrumented_node dut/in_core -driver_supply SS_AAA -
receiver_supply SS_BBB -src_undriven} }

# Number of potential locations for Instrumentation: 1
set ISO_SIGNET_PORT(2) { { dut/core/out -policy PD_AAA/
isolation/iso_out -domain PD_AAA }
{ -isolation_signal ISO_ENB -isolation_sense low -
clamp_value 0 -location parent }
{ -instrumented_node dut/out_core -driver_supply SS_AAA -
receiver_supply SS_BBB -receiver_floating } }
```

Inserting NOR-Style Isolation Cells as per UPF 3.0 LRM

The NOR-style isolation is applicable to the isolation strategies with clamp value 0. In general, these are single rail isolation cells with switched supply. The output of the isolation cell is 0 when the power is OFF and isolation enable is active.

VCS NLP supports the UPF 3.0 LRM behavior for inserting the NOR-style isolation cells. VCS NLP automatically inserts NOR-style isolation cells only when the following rules are met:

- The `-isolation_supply_set` option of the `set_isolation` command is specified as an empty supply set `{}`
- Isolation strategy is defined with clamp value 0

Use Model

Following is the use model:

```
set_isolation iso_name -isolation_supply_set {} -  
domain domain_name -elements {element_list} -  
clamp_value 0
```

Example:

```
set_isolation isol -isolation_supply_set {} -  
domain PD1 -elements {O1} -clamp_value 0
```

Key points to note

- VCS NLP issues a compile-time error message, if NOR-style isolation strategies with clamp value other than 0 is specified.
- The following steps describe the corruption behavior when clamp value is 0 and isolation supply is specified as {}:
 - VCS NLP performs rail-based corruption (relaxed corruption).
 1. Output is 0 if isolation enable is 1 and the ground rail of the domain where the isolation cell placed is FULL_ON.
 2. Output is equal to input if isolation enable is 0 and both power and ground rails of the domain where the isolation cell is placed are FULL_ON.
 3. Corrupt otherwise
- VCS NLP reports all the marked NOR-style isolation strategies in the mvsim_native_reports/isolation_association.rpt file.

Following is a sample `isolation_association.rpt` file:

```
##### POWER DOMAIN tb/top_inst/PD2 #####
# Number of potential locations for Instrumentation: 1
set ISO_SIGNET_PORT(1) { { CORE2/out1 -policy PD2/
isolation/V2_iso -domain PD2 }
{ -isolation_signal iso_sig -isolation_sense high
-clamp_value nor -location self }
{ -instrumented_node CORE2/out1 -driver_supply
PD2.primary -receiver_supply TOP.primary } }

#-----
#####
POWER DOMAIN tb/top_inst/PD1 #####
# Number of potential locations for Instrumentation: 1
set ISO_SIGNET_PORT(2) { { CORE1/out1 -policy PD1/
isolation/V1_iso -domain PD1 }
{ -isolation_signal iso_sig -isolation_sense high
-clamp_value nor -location self }
{ -instrumented_node CORE1/out1 -instance
tb.top_inst.CORE1.iso_cell -driver_supply PD1.primary -
receiver_supply TOP.primary } }
```

- The output is corrupted only if the ground supply of the NOR-style isolation strategy is switched off

Corruption Semantics for NOR-Style Isolation

Table 9-1 lists the corruption semantics for NOR-style isolation.

Table 9-1 Corruption Semantics for NOR-Style Isolation

Input pin of isolation	Enable pin of isolation	VDD	VSS	Output pin of isolation
Don't care	Don't care	Don't care	OFF	X
Don't care	Active	OFF	ON	0
Don't care	Inactive	OFF	ON	Same as input
Don't care	Active	ON	ON	0
Don't care	Inactive	ON	ON	Same as input

Limitations

Following are the limitations of this feature:

- Location parent/fanout/automatic in the NOR-style isolation strategy is not supported.
- Clamp value 1 is not supported.

Support for Conservative diff_supply_only Isolation

If an isolation strategy uses the `-diff_supply_only` option, as shown below, VCS NLP inserts an isolation cell for the port only when none of its global loads/sinks use the same supply as its driver/source.

```
set_isolation iso -domain P1 -applies_to outputs
-diff_supply_only TRUE -location self
```

You can use the `conservative_diff_supply_only_isolation` design attribute in the UPF to insert an isolation cell for the `-diff_supply_only` isolation strategies for the cases where one

or more of the loads/sinks might have the same supply as the driver/source, but at least one of them uses a different supply. Specify the following in the UPF:

```
set_design_attributes -attribute  
conservative_diff_supply_only_isolation TRUE
```

Under this attribute, for the `-diff_supply_only` strategies on a port fanning out to multiple power domains, VCS NLP inserts an isolation cell if **at least one** logic sink power domain is different from the logic source domain.

Consider the following example where `-diff_supply_only` isolation strategy is applied to the output ports of power domain P1 with location “self”. The logic in power domains P1 and P2 are powered by their respective primary supplies. In Figure-1, VCS NLP does not insert isolation cell as one of the loads uses the same supply as the driver.

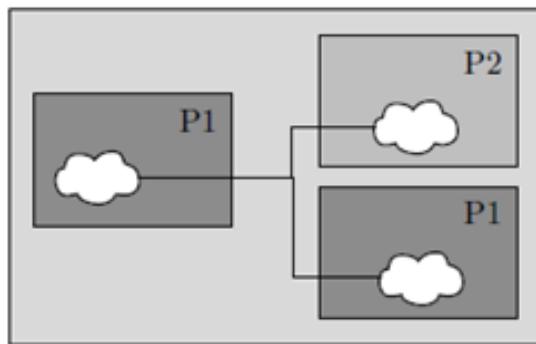


Figure-1

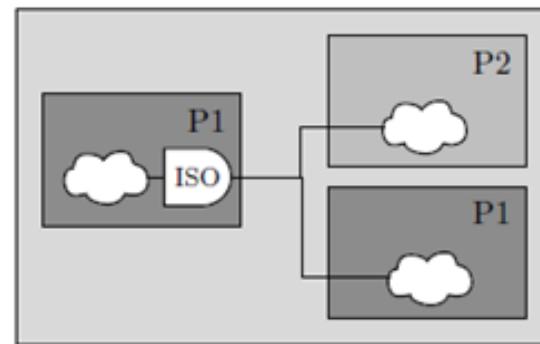


Figure-2

If you use the `conservative_diff_supply_only_isolation` design attribute, then VCS NLP inserts an isolation cell, as shown in Figure-2.

The `conservative_diff_supply_only_isolation` design attribute is a global attribute and applies to all isolation strategies with `-diff_supply_only`.

Note:

- The `conservative_diff_supply_only_isolation` attribute applies to the `-diff_supply_only` option of the `set_isolation` command only. It does not apply to the `-sink` option.
- The `conservative_diff_supply_only_isolation` attribute impacts isolation strategies with location `self` or `parent` only.

Name-Based Isolation Cell Association in Netlist

VCS NLP supports name-based association of isolation cells in the netlists written using DC/ICC with the `mv_upf_strategy_in_inst_name` variable.

Following is the naming convention of the isolation cell instance in the netlist:

```
ISO <name_prefix>_snps_<power_domain_name>__<iso_
strategy_name>_snps<pin_name>_<inst_index>_<name
_suffix>
```

The key concept of this feature is that if VCS NLP finds an isolation instance whose name matches the above naming convention, then it associates this isolation instance with the corresponding isolation strategy.

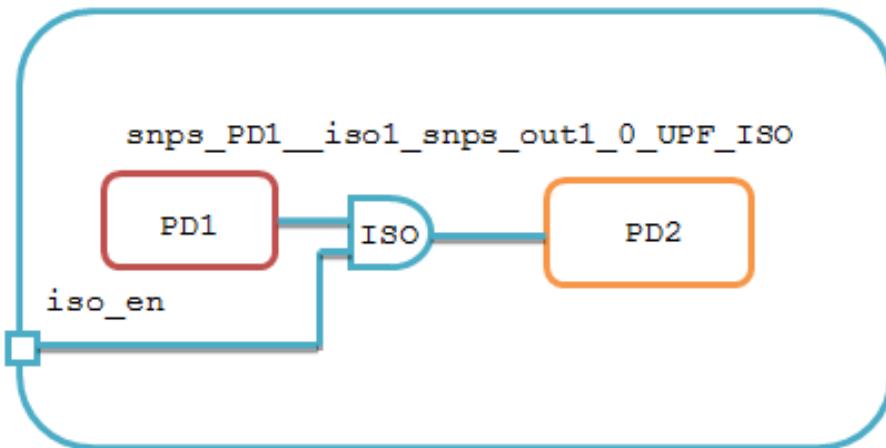
For example, in [Figure 9-1](#) the crossover starts from PD1 to PD2.

Following are the isolation strategies on the source (PD1/iso1) and sink (PD2/iso2) domains:

```
set_isolation iso1 -domain PD1  
-source PD1.primary  
-sink TOP.primary
```

```
set_isolation iso2 -domain PD2  
-source PD1.primary  
-sink PD2.primary
```

Figure 9-1 Name-based Isolation Cell Association



The source domain strategy PD1/iso1 has -source PD1.primary and -sink TOP.primary, which does not fulfill the requirement for the path-based strategy association.

The sink domain strategy PD2/iso2 has -source PD1.primary and -sink PD2.primary, which fulfills the requirement for the path-based strategy association.

However, PD1/iso1 strategy name matches with the isolation cell present on the path. Therefore, VCS NLP associates isolation cell snps_PD1__iso1_snps_out1_0_UPF_ISO with the isolation strategy PD1/iso1.

Support for Isolation Terminal Points at the Hierarchical Boundaries

This section describes a method to mark hierarchical boundaries as terminal points for isolation strategies using `-source`, `-sink`, `-diff_supply_only`, or `-location automatic`. This section contains the following topics:

- [Introduction](#)
- [Use Model](#)
- [Examples](#)
- [Limitations](#)

Introduction

The `set_design_attributes` command on the hierarchical boundary is considered for isolation analysis. You can use the design attribute `terminal_boundary` to change the behavior of the hierarchical boundaries. The hierarchical boundary which contains this attribute is considered as a terminal point for the global net. This effectively changes the behavior of `-source`, `-sink`, `-diff_supply_only`, or `-location automatic` strategies.

Use Model

You can use the `terminal_boundary` design attribute to mark hierarchical boundaries as terminal points for isolation strategies.

Syntax:

```
set_design_attributes -elements {instance_name} -attribute  
terminal_boundary true
```

Key Points to Note about terminal_boundary

- The default value of this attribute is `false`
- VCS generates an error if any element in the elements list is not a power domain boundary
- VCS generates an error if any element in the elements list is a power domain boundary which is a leaf cell (macros/pads)
- The `terminal_boundary` attribute is supported at instance level only. It is not supported at port level. When this attribute is set on an instance, the boundary ports of that instance becomes a terminal point for any isolation global net.

The purpose of using the `terminal_boundary` attribute is to give flexibility and simplify the use of source, sink, `diff_supply_only`, or the location automatic strategies in a bottom up flow. The design which has such attribute is expected to be integrated into a bigger design.

Note:

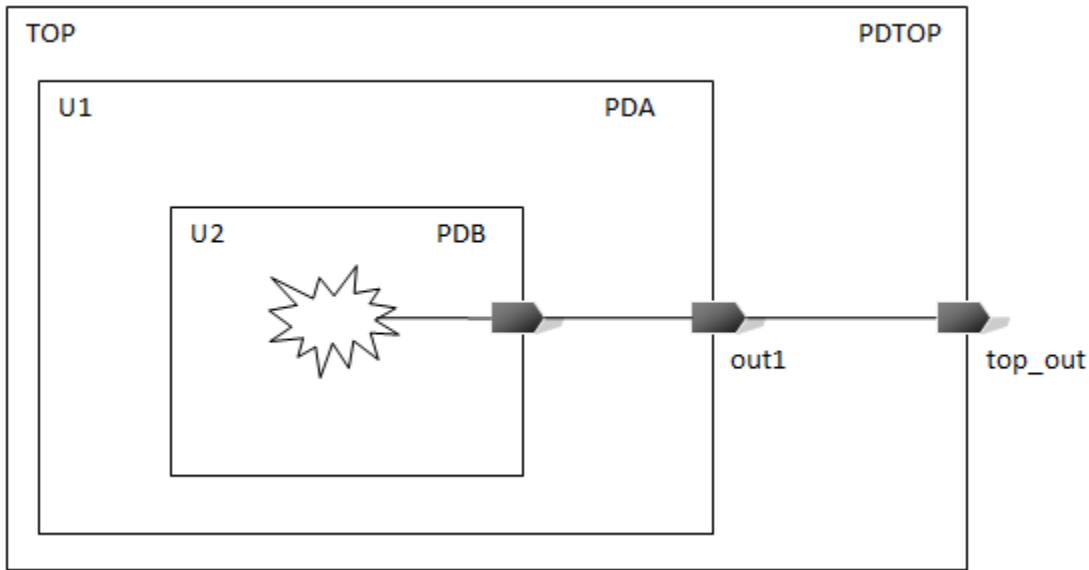
For -location automatic strategy specified on terminal boundary ports, VCS NLP always inserts virtual isolation cell at -location self.

Enhancement to the `-driver_supply` and `-receiver_supply` Port Attributes

VCS NLP supports both `-driver_supply` and `-receiver_supply` attributes for the terminal boundary ports.

Examples

Example-1: Isolation with -source



Base UPF

```
create_supply_set SSTOP
create_supply_set SSA
create_supply_set SSB
create_power_domain TOP -supply {primary SSTOP}
create_power_domain PDA -elements {U1} -supply {primary SSA}
-scope U1
create_power_domain PDB -elements {U1/U2} -supply {primary SSB}

set_isolation isol -domain PDTOP -source SSB
set_isolation_control isol -domain PDTOP -location self
set_isolation iso2 -domain PDTOP -source SSA
set_isolation_control iso2 -domain PDTOP -location self
```

Additional UPF

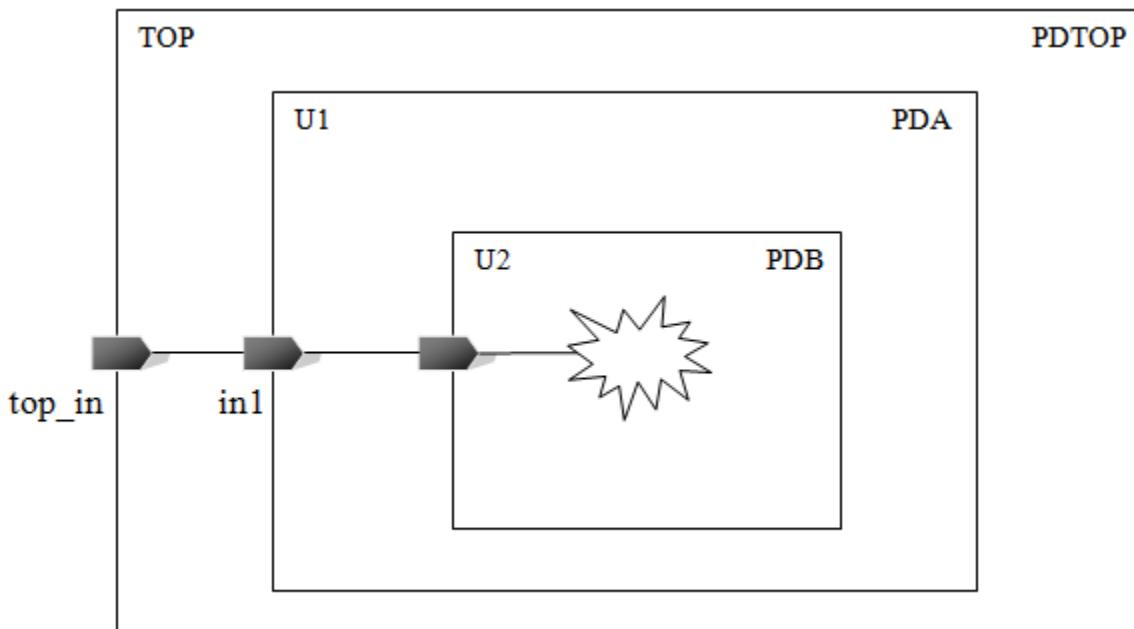
```
set_design_attributes -elements {U1} -attribute
```

```
terminal_boundary true
```

Description

If the base UPF alone is used, then only the `iso1` strategy is implemented because SSB is the driver of the single path shown. If additional UPF is included, U1 is implemented stand-alone and two paths are considered instead of one. No strategies apply to the path from the source to the hierarchical port `out1`. However, on the path from `out1` to `top_out`, the effective source supply is SSA, so strategy `iso2` is implemented.

Example-2: Isolation with -sink



Base UPF

```
create_supply_set SSTOP
create_supply_set SSA
create_supply_set SSB
create_power_domain TOP -supply {primary SSTOP}
create_power_domain PDA -elements {U1} -supply {primary SSA}
```

```

-scope U1
create_power_domain PDB -elements {U1/U2} -supply {primary
SSB}

set_isolation isol -domain PDTOP -sink SSB
set_isolation_control isol -domain PDTOP -location self
set_isolation iso2 -domain PDTOP -sink SSA
set_isolation_control iso2 -domain PDTOP -location self

```

Additional UPF

```

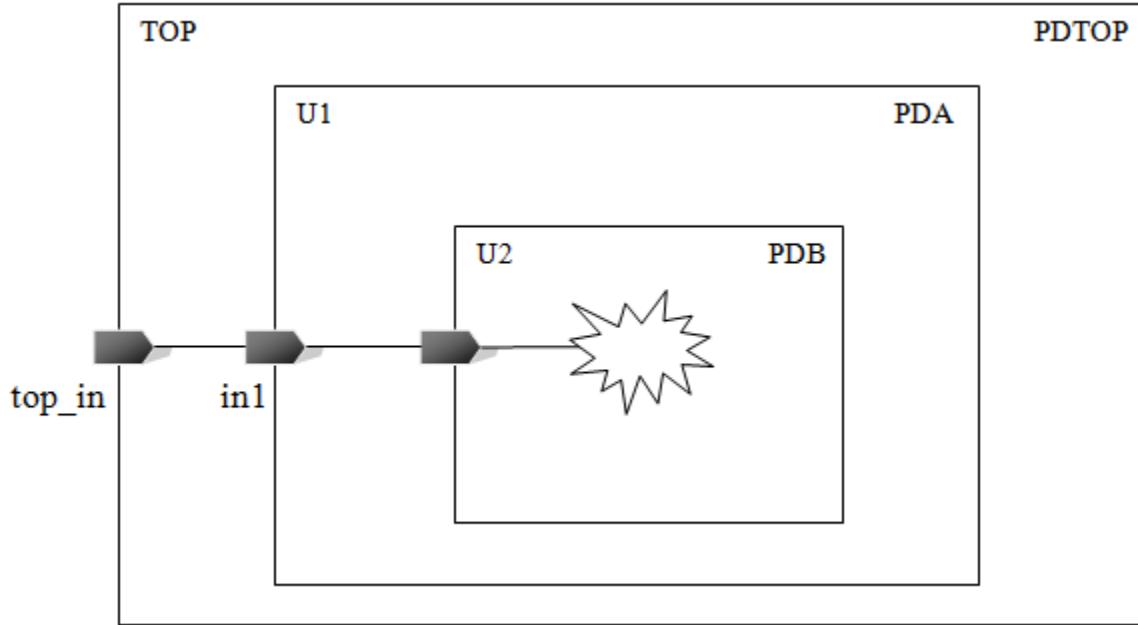
set_design_attributes -elements {U1} -attribute
terminal_boundary true

```

Description

If the base UPF alone is used, then only the `isol` strategy is implemented because `SSB` is the sink of the single path shown. If additional UPF is included, `U1` is implemented stand-alone and two paths are considered instead of one. No strategies apply to the path from the source to the hierarchical port `in1` to the sink. However, on the path from `top_in` to `in1`, the effective source supply is `SSA`, so strategy `iso2` is implemented.

Example-3: Isolation with receiver_supply



Base UPF

```
create_supply_set SSTOP
create_supply_set SSA
create_supply_set SSB
create_power_domain PDTOP -supply {primary SSTOP}
create_power_domain PDA -elements {U1} -supply {primary SSA}
-scope U1
create_power_domain PDB -elements {U1/U2} -supply {primary SSB}

set_isolation isol1 -domain PDTOP -sink SSB
set_isolation_control isol1 -domain PDTOP -location self
set_isolation iso2 -domain PDTOP -sink SSTOP
set_isolation_control iso2 -domain PDTOP -location self
set_port_attributes -elements{U1} -ports{U1/in1} -
receiver_supply SSTOP
```

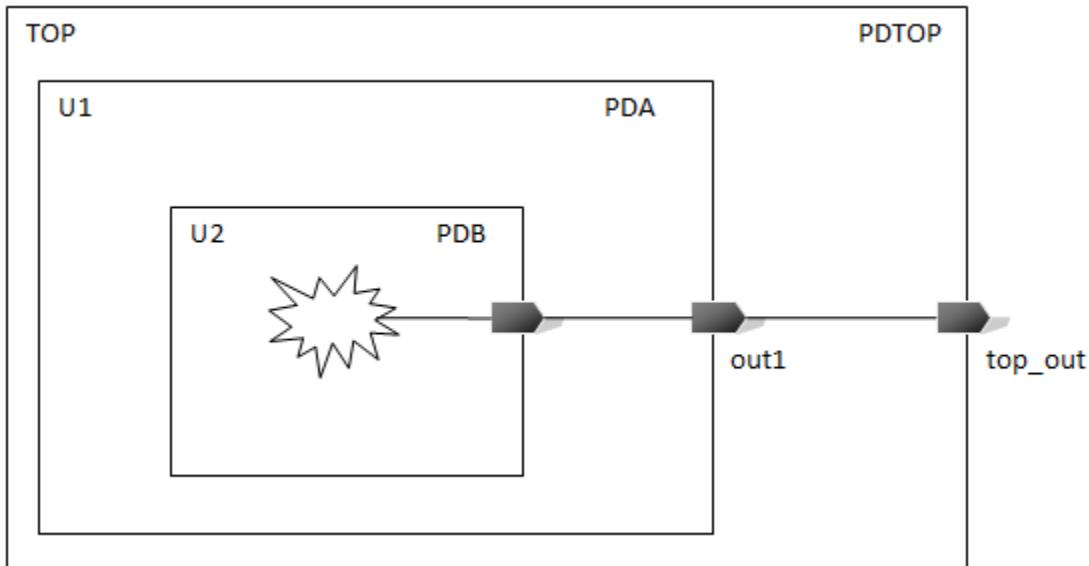
Additional UPF

```
set_design_attributes -elements {U1} -attribute
terminal_boundary true
```

Description

If the base UPF alone is used, then only the `iso1` strategy is implemented because SSB is the sink of the single path shown. If additional UPF is included, `U1` is implemented stand-alone and two paths are considered instead of one. No strategies apply to the path from the hierarchical port `in1` to the sink. However, on the path from `top_in` to `in1`, the effective sink supply is `SSTOP` due to `-receiver_supply`, so strategy `iso2` is implemented.

Example-4: Isolation with `driver_supply`



Base UPF

```
create_supply_set SSTOP
create_supply_set SSA
create_supply_set SSB
create_power_domain PDTOP -supply {primary SSTOP}
create_power_domain PDA -elements {U1} -supply {primary SSA}
-scope U1
create_power_domain PDB -elements {U1/U2} -supply {primary SSB}
```

```
set_isolation iso1 -domain PDTOP -source SSB
set_isolation_control iso1 -domain PDTOP -location self
set_isolation iso2 -domain PDTOP -source SSTOP
set_isolation_control iso2 -domain PDTOP -location self
set_port_attributes -elements{U1} -ports {U1/out1} -
driver_supply SSTOP
```

Additional UPF

```
set_design_attributes -elements {U1} -attribute
terminal_boundary true
```

Description

If the base UPF alone is used, then only the `iso1` strategy is implemented because `SSB` is the driver of the single path shown. If additional UPF is included, `U1` is implemented stand-alone and two paths are considered instead of one. No strategies apply to the path from the source to the hierarchical port `out1`. However, on the path from `out1` to `top_out`, the effective source supply is `SSTOP` due to `-driver_supply`, so strategy `iso2` is implemented.

Limitations

- Terminal boundary cannot be set on th leaf level instance for DB.
- Terminal Boundary cannot be set on non-power domain boundary hierarchy.
- Terminal boundary attribute cannot be set on scope where the domain is scoped at top level (specifying `-scope` option for `create_power_domain`).
- Isolation with location parent and fanout is not allowed on terminal boundary ports.

Path-Based Isolation Support for Heterogeneous Loads

The location specified by the isolation strategy indicates the domain of the logic hierarchy where the cell must be inserted. For example, the `self` location implies that the isolation could be inserted in the extent of the reference domain. The `parent` location implies that the isolation could be inserted in the extent of the domain of the parent hierarchy/instance.

Isolation cell insertion is not constrained to the immediate self/parent hierarchies. You can use the `hetero_fanout_isolation` design attribute, as shown below, to insert isolation cell in the extent of the domain of the self or parent hierarchy/instance.

```
set_design_attributes -elements { . } -attribute  
hetero_fanout_isolation true
```

The following behavior is supported under the `hetero_fanout_isolation` design attribute:

- Isolation cell insertion at location `self/parent` with heterogeneous loads for the `-diff_supply_only` isolation strategies. That is, for a port with multiple sinks that are driven by different power supplies, with at least one sink having similar supply as the source, you can insert isolation cell in extent of the domain of the self or parent hierarchy/instance, but not crossing the domain boundary.
- Isolation cell insertion at location `self/parent` with heterogeneous loads for the `-source/-sink` or `-sink` isolation strategies. That is, for a port with multiple sinks that are driven by different power supplies, you can insert an isolation cell in the extent of the domain of the self or parent hierarchy/instance, but not crossing the domain boundary.

Key Points to Note

- With this support, specifying multiple path-based (`-source/-sink/-diff_supply_only`) isolation strategies is supported on a single target port.
- Isolation strategies specified with the `-source` or `-sink` options support all paths, including those whose loads have heterogeneous supplies.
- Port-based isolation rule (other than `-no_isolation`) is evaluated as per the isolation strategy on the individual driver-load pairs. Therefore, presence of both port-based and path-based isolation specified with the same granularity results in the combination of path-based isolation on specific driver-load pairs of the path, and port-based isolation on the remainders of the path. If there are no remainders, then the port-based isolation does not apply. For more information, see “[Precedence Rule Scenarios](#)” .

Heterogeneous Load Scenarios

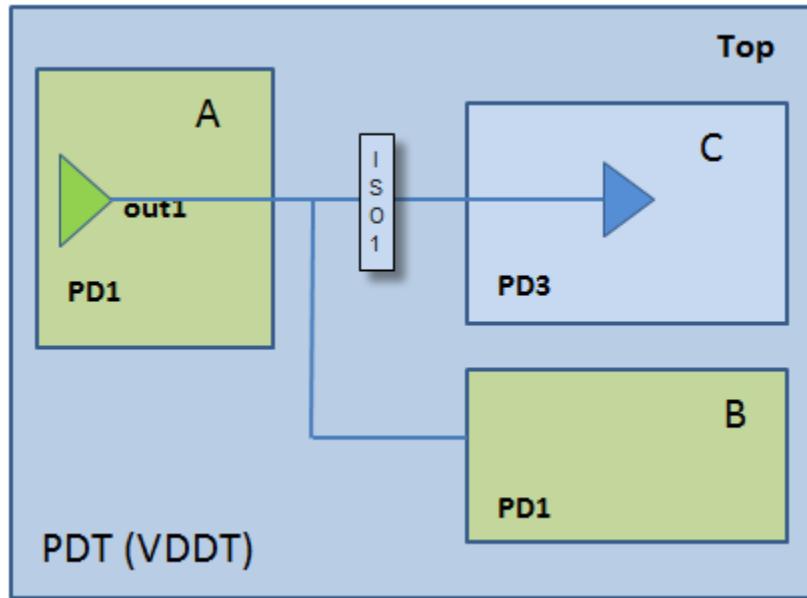
- `-location parent with -diff_supply_only`

Consider the following example where isolation strategy ISO1 with `-diff_supply_only` and `location parent` is applied to the output ports of power domain PD1.

```
set_isolation ISO1 -location parent -  
diff_supply_only -domain PD1
```

In [Figure 9-2](#), one of the loads uses the same supply as the driver. Path from **A** to **C** gets isolated without isolating path from **A** to **B**.

Figure 9-2 -location parent with -diff_supply_only



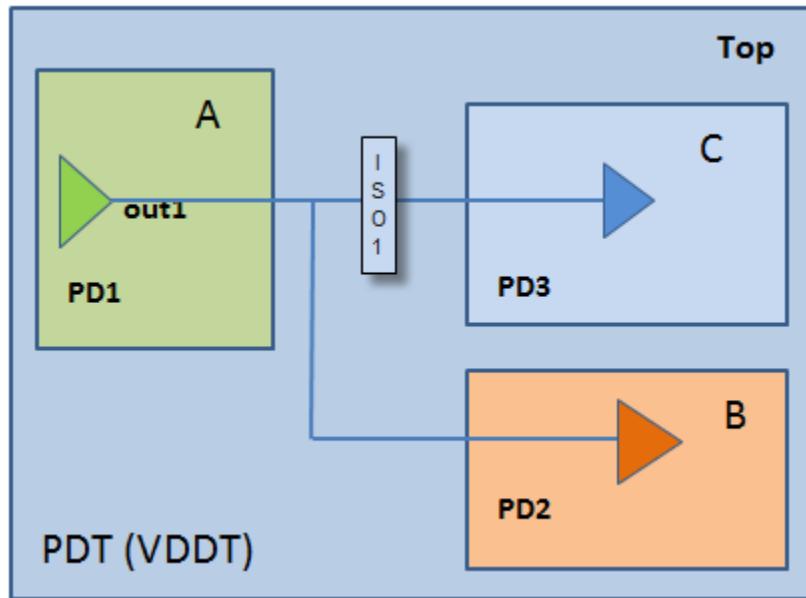
- location parent with -source/-sink or -sink

Consider the following example where isolation strategy ISO1 with -source/-sink and location parent is applied to the output ports of power domain PD1.

```
set_isolation ISO1 -location parent -source PD1  
-sink PD3 -domain PD1
```

In [Figure 9-3](#), isolation is placed in the extent of PD3 such that the path from **A** to **C** gets isolated. The path from **A** to **B** is not isolated.

Figure 9-3 -location parent with -source/-sink

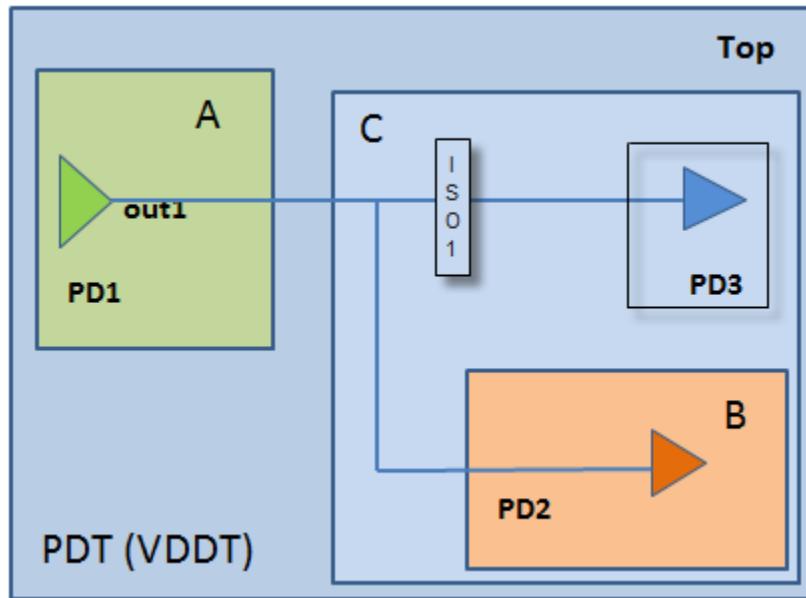


- location self with -sink

Consider the following example where the isolation strategy ISO1 with -sink and location self is applied to the input port of power domain PD3.

```
ISO1 -location self -sink PD3 -domain PD3
```

Figure 9-4 -location self with -sink

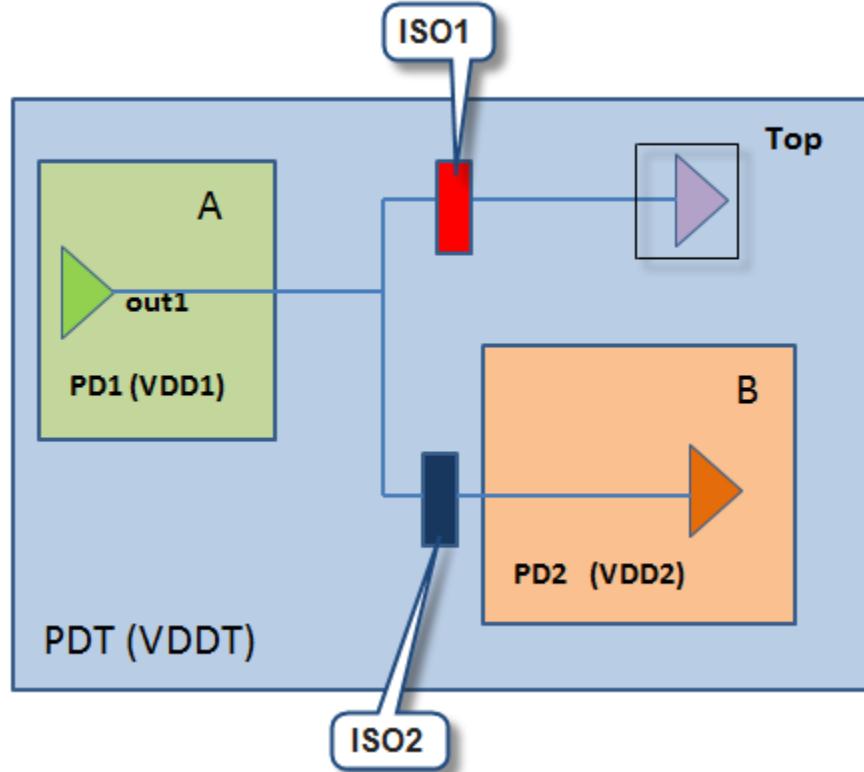


In Figure 9-4, isolation cell is placed in the extent of the power domain PD3 such that the path from PD1 to PD3 gets isolated leaving the path from PD1 to PD2 unisolated.

Precedence Rule Scenarios

Case-1

Figure 9-5 Scenario-1: All isolation strategies are on domain PD1



Scenario-1: All isolation strategies are on domain **PD1**

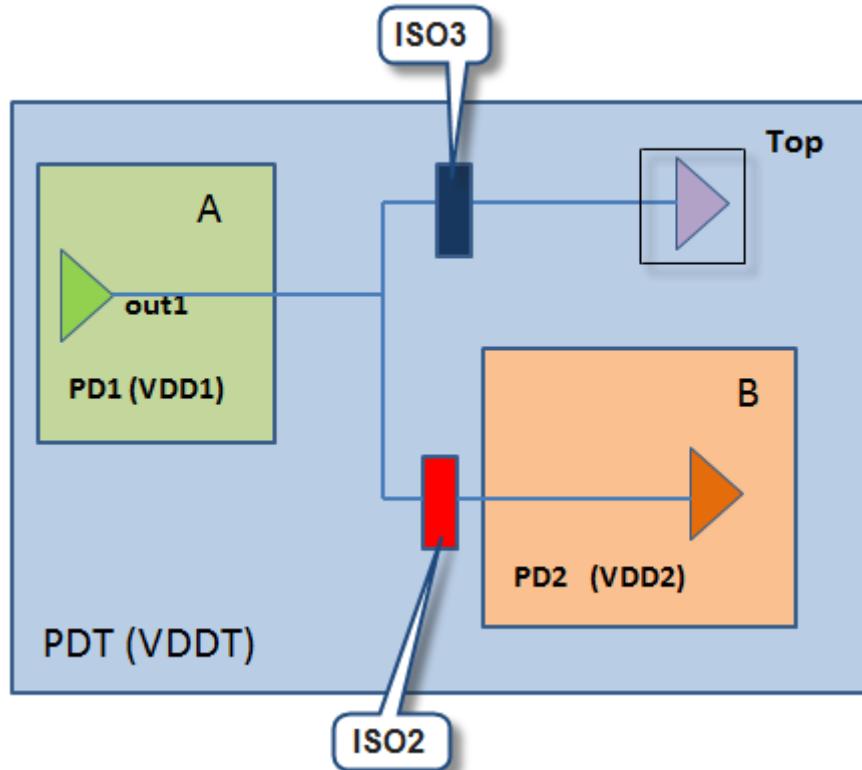
ISO1: -elements A -location parent

ISO2: -elements A -sink VDD2 -location parent

Precedence order: ISO2 == ISO1

Path-based ISO2 takes more specific path ($VDD1 \rightarrow VDD2$), while ISO1 takes the remainder paths ($VDD1 \rightarrow VDDT$).

Figure 9-6 Scenario-2: All isolation strategies are on domain PD1



Scenario-2: All isolation strategies are on domain **PD1**

ISO1: -elements A -location parent

ISO2: -elements A -sink VDD2 -location parent

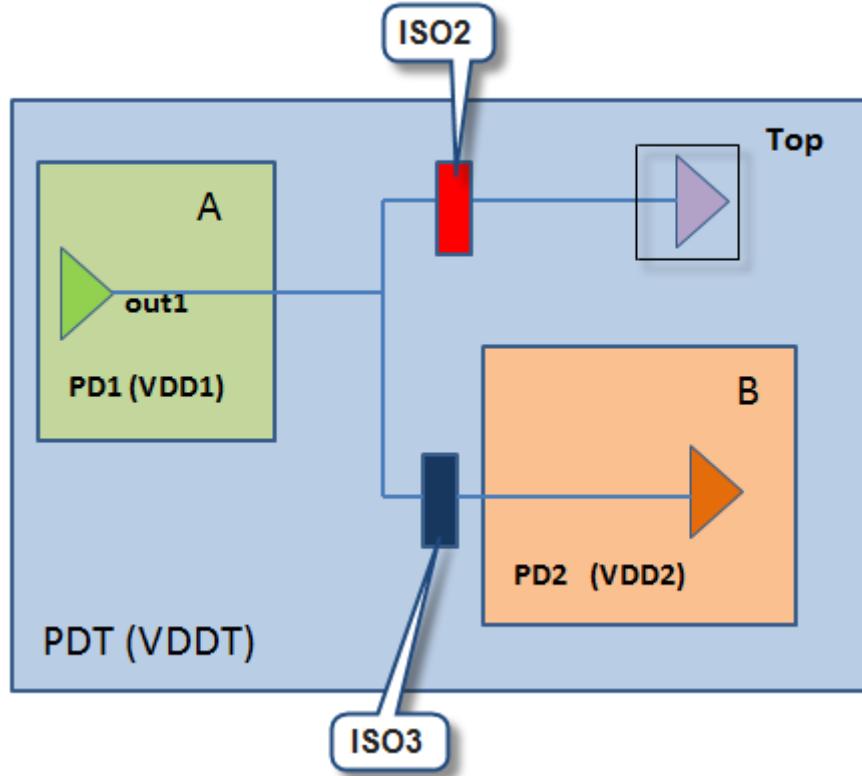
ISO3: -elements -sink VDDT -location parent

Precedence order: ISO2, ISO3, ISO1 – ISO1 dropped

ISO1 is dropped because ISO2 and ISO3 already covered all possible paths of ISO1.

Case-2

Figure 9-7 Scenario-1: All isolation strategies are on domain PD1



Scenario-1: All isolation strategies are on domain **PD1**

ISO1: -elements A -src VDD1 -location self

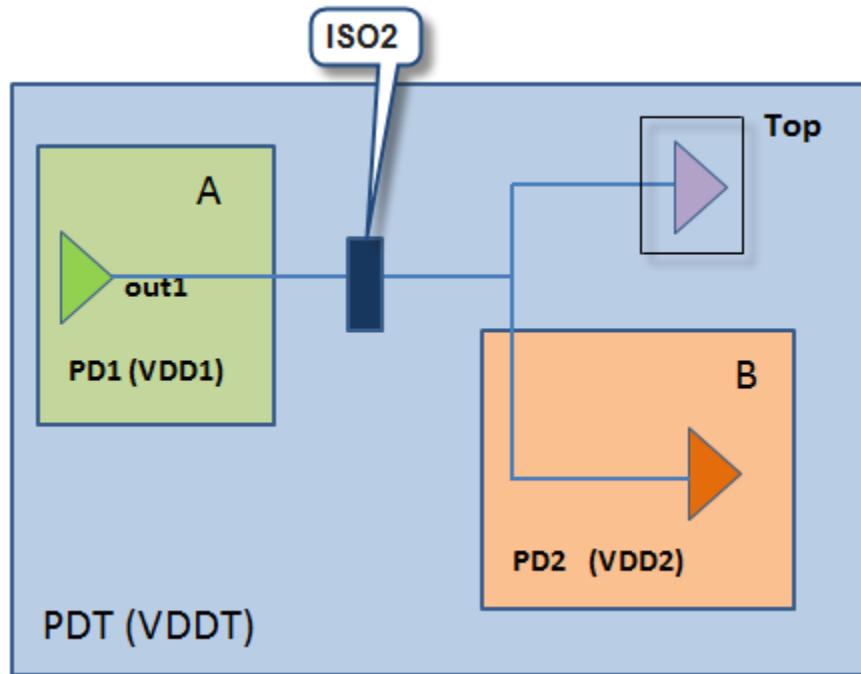
ISO2: -elements A -location parent

ISO3: -elements A -sink VDD2 -location parent

Precedence order: ISO3, ISO1, ISO2 – ISO1 dropped

ISO1 is dropped because it cannot be implemented in the presence of ISO3.

Figure 9-8 Scenario-2: All isolation strategies are on domain PD1



Scenario-2: All isolation strategies are on domain **PD1**

ISO1: -src VDD1 -location self

ISO2: -elements A -location parent

ISO3: -sink VDD2 -location parent

Precedence order: ISO2, ISO3, ISO1 - ISO1 and ISO3 dropped

ISO1 and ISO3 are dropped because ISO2 already covered all paths.

Support for **-exclude_elements** and **-update** Options

VCS NLP supports the following options with the `set_isolation` command:

- `-exclude_elements`
- `-update`

The `-update` option provides additional information for a previous command with the same strategy name and domain name, and executed in the same scope.

The `-exclude_elements` option is used to exclude any port or element from the strategy.

Use Model

The following topics describe the use model:

- [Using `-exclude_elements` with `set_isolation` Command](#)
- [Using `-update` with `set_isolation` Command](#)

Using `-exclude_elements` with `set_isolation` Command

The `-exclude_elements` option explicitly identifies the set of ports to which this strategy does not apply.

Syntax:

```
set_isolation strategy_name -exclude_elements  
{exclude_list}
```

The `exclude_list` might contain rooted names of instances or ports in the specified domain. If an instance name is specified in the `exclude_list`, it is equivalent to specifying all the ports of that instance in the `exclude_list`.

Using -update with `set_isolation` Command

The `-update` option adds information to the base command executed in the same scope.

Syntax:

```
set_isolation strategy_name
  -domain domain_name
  [-elements element_list]
  [-exclude_elements exclude_list]
  [-update]
```

Example-1: Adding elements to the isolation strategy

```
set_isolation test_strategy -domain pda
  -elements {a b c d}
  -isolation_signal {iso_en}
  -isolation_sense {LOW}
set_isolation test_strategy -domain pda -update
  -elements {e f g}
```

The `-update` option adds elements e, f, and g to `test_strategy`.

Example-2: Excluding elements from the isolation strategy

```
set_isolation iso_pd_top -domain pd_top -applies_to output
...
set_isolation iso_pd_top -domain pd_top -update -
  -exclude_elements {out1}
```

Isolation strategy `iso_pd_top` is applied on all the output ports except `out1` because `out1` is excluded from the `iso_pd_top` strategy.

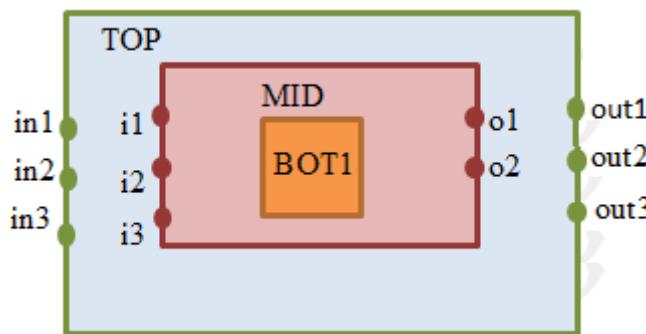
Support for `-applies_to` with `-elements` for Isolation

VCS NLP supports the `-applies_to` option as a filter in presence of the `-elements` option of the `set_isolation` command. It allows you to filter the elements specified in the `-elements` list of the `set_isolation` command based on the direction specified by the `-applies_to` option.

Key Points to Note

- The `-applies_to` option is always interpreted relative to the domain for which the strategy is being defined (also known as reference domain).
- HighConn elements are considered for filtering only when the lower domain boundary feature is enabled. When lower domain boundary feature is disabled, HighConn elements result in an error for not being on the boundary of the domain.

Consider the following design structure:



The isolation strategy ISO is applied to out1, out2, out3, MID/i1, MID/i2, and MID/i3.

```
set_scope /  
set_design_attributes -elements { . } -attribute  
lower_domain_boundary true  
create_power_domain PD_TOP  
create_power_domain PD_MID -elements { MID }  
create_power_domain PD_BOT -elements { MID/BOT1 }  
set_isolation ISO -domain PD_TOP -elements { * } -applies_to  
outputs ...
```

For the following example, lower domain boundary feature is enabled. Therefore, the isolation strategy ISO is applied to MID/o1 and MID/o2 because they are the input boundaries of the domain PD_TOP.

```
set_scope /  
set_design_attributes -elements { . } -attribute  
lower_domain_boundary true  
create_power_domain PD_TOP  
create_power_domain PD_MID -elements { MID }  
set_isolation ISO -domain PD_TOP -elements { MID } -  
applies_to inputs ...
```

VCS NLP supports this feature by default. For backwards compatibility, use the DISABLE_APPLIES_TO_WITH_ELEMENTS variable in the synopsys_bc.setup file, as shown in the following example:

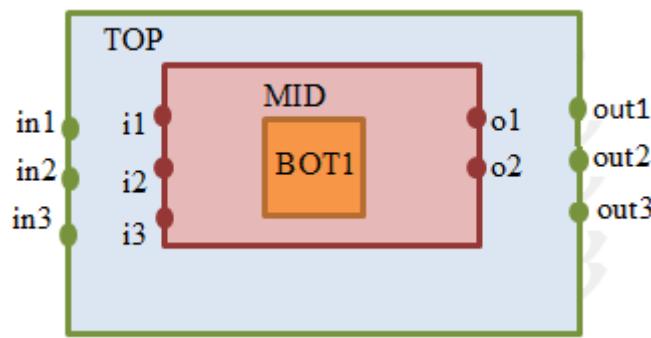
```
enable DISABLE_APPLIES_TO_WITH_ELEMENTS
```

When the DISABLE_APPLIES_TO_WITH_ELEMENTS variable is enabled, the -applies_to option is ignored in presence of -elements of the set_isolation command.

Support for Specifying Instances with `set_isolation -elements`

You can specify the design instances in the `-elements` list of the `set_isolation` command. The isolation strategy is applied only to the boundary elements of the specified instance in the reference domain. With this support, you can easily specify the target elements for the strategy.

Consider the following design structure:



The instance `MID` contains input ports `i1`, `i2`, `i3` and output ports `o1` and `o2`. The isolation strategy `ISO` is applied to all the input and output ports of `MID`.

```
set_scope /  
set_design_attributes -elements {.} -attribute  
lower_domain_boundary true  
create_power_domain PD_TOP  
create_power_domain PD_MID -elements { MID }  
set_isolation ISO -domain PD_TOP -elements { MID } ...
```

You can specify wildcards for the instances in `-elements`. When wildcards are specified in the `-elements` option, it matches any ports or instances at the current level of hierarchy.

VCS NLP supports instances in the `-elements` list by default. For backwards compatibility, use the `DISABLE_INSTANCE_WITH_ELEM` variable in the `synopsys_bc.setup` file, as shown below:

```
enable DISABLE_INSTANCE_WITH_ELEM
```

This variable disables the instances support in the `-elements` option.

Fanout Isolation Instrumentation Support for Signals of SV Interface/Modport

VCS NLP supports isolation on the signals of SystemVerilog interface/modport with location fanout (`-location fanout`).

Generating Virtual Isolation Reports

You can use the `-power=write_mvinfo` compile-time option to generate the following reports:

- [Virtual Isolation Insertion Report](#)
- [Virtual Isolation Inference Report](#)

Following is the use model of the `-power=write_mvinfo` option:

```
% vcs -upf <upf_file> -sverilog file_name.v  
-power_top top -power=write_mvinfo
```

Virtual Isolation Insertion Report

VCS NLP generates an isolation insertion report at compile time which provides the details such as signal being isolated and its corresponding isolation control, isolation sense, and clamp value.

Reporting Format

```
{isolation_control  isolation_sense  clamp_value  
isolated_signal}
```

1. The report is dumped in the `mvsim_native_reports/isolation_insertion_info.txt` file as a Tcl list.
2. The `isolation_control` and `isolation_sense` are the isolation enable and the sense specified in UPF for the isolation strategy.
3. The `clamp_value` is “high” if 1 and “low” if 0 in the UPF. “Z” is printed as it is.
4. Hierarchy of the `isolation_control` and `isolated_signal` are relative to power top.
5. “/” is used as the hierarchical separator.
6. The file is sorted with `isolated_signal` name field.
7. If the isolation is already implemented in the design, it is not written into the insertion report.

Sample Report

```
set mvsim_iso_list {  
{p1/iso high low f2/f2d4/in1\[1\]}  
{p1/iso high high t22/d2/in1\[1\]}  
{p1/iso high high t42/d4/in1\[3\]}  
}
```

}

Virtual Isolation Inference Report

You can use the `-power=write_mvinfo` option to generate a report for the virtual isolation inference from the RTL based on the UPF. The generated report can be used to compare with the isolation inference report dumped by Formality on the same RTL and UPF.

The isolation inference report from VCS NLP is generated as `new_isolation_association.rpt` file in the `mvsim_native_reports` directory.

Following is the sample report from VCS NLP:

```
set mvsim_iso_list {  
    { /inst/X/Y/out[0] /Island_V1 V1_iso /inst/X/Y/out[0] L 1 }  
    { /inst/X/Y/out[1] /Island_V1 V1_iso /inst/X/Y/out[1] L 1 }  
    { /inst/X/Y/out[2] /Island_V1 V1_iso /inst/X/Y/out[2] L 1 }  
    { /inst/X/Y/out[3] /Island_V1 V1_iso /inst/X/Y/out[3] L 1 }  
    { /inst/X/Y/out[4] /Island_V1 V1_iso /inst/X/Y/out[4] L 1 }  
    { /inst/X/Y/out[5] /Island_V1 V1_iso /inst/X/Y/out[5] L 1 }  
    { /inst/X/Y/out[6] /Island_V1 V1_iso /inst/X/Y/out[6] L 1 }  
    { /inst/X/Y/out[7] /Island_V1 V1_iso /inst/X/Y/out[7] L 1 }
```

Isolation Inference Report Format

Each inferred isolation cell consists of the following fields:

```
<boundary_port_path> <power_domain_name>  
<isolation_strategy_applied> <location_port_path>  
<L|H> <order>
```

Where,

```
<boundary_port_path>
```

Path to the boundary ports for the specified domain in set_isolation.

<power_domain_name>

Path to the specified domain name in set_isolation.

<isolation_strategy_applied>

The specified strategy name from set_isolation.

<location_port_path>

The path to the port being isolated. This can be the same as the boundary port.

<L | H>

The isolation cell is located on the low or high side of the location port.

<order>

Distinguishes multiple isolation cells back to back. It starts from 1 and increments to the total number of back to back isolation cells at that location.

Number 1 represents the nearest isolation cell to the boundary port path, and the number increments for each back to back isolation cell. The largest number is the isolation cell farthest away from the boundary port path.

Isolation/Corruption on Ports with Logical Expressions in Port Map

You can use the `-power=portconnect_exprs` compile-time option to enable isolation/corruption on the logical expressions in the port map. By default, isolation/corruption is not enabled on the logical expressions in the port map.

For example, consider the following code:

```
child child_inst (.data_in(~data));
```

You can use the `-power=portconnect_exprs` option to enable isolation/corruption on `child_inst/data_in`.

Note:

Logical expressions that are part of a concatenation expression are handled for isolation/corruption by default.

Limitations

Following are the limitations of this feature:

- Expressions in the port map of a user-defined* type `port` are not supported for isolation/corruption.
*Refers to any type other than basic Verilog types (reg, wire, logic, and so on). For example: struct, union, enum, and so on.
- Expressions containing streaming and assignment pattern operators are not supported.

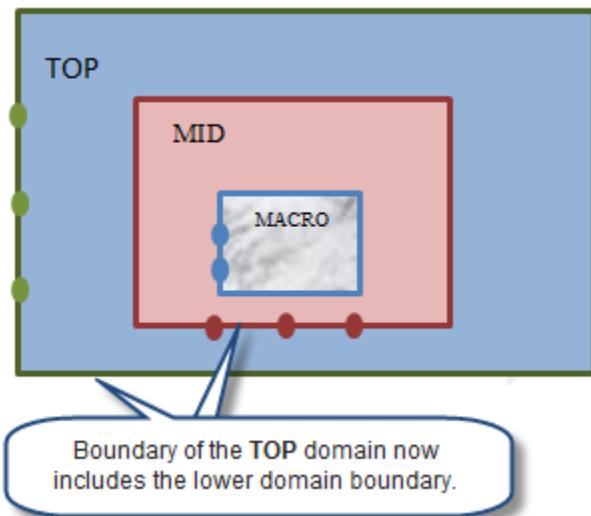
Supporting Lower Domain Boundary for Isolation

You can use the `lower_domain_boundary` design attribute to extend the definition of a power domain boundary to include the lower domain boundary ports/pins, as shown in [Figure 9-9](#). In this figure, the boundary `TOP` includes both the interface colored green and interface colored red.

However, the boundary of `TOP` does not extend to the interface colored blue of the domain `MACRO`. The boundary of the domain `MID` extends to the boundary of the nested domain `MACRO`.

For hierarchical models, this feature allows you to insert isolation cells in the parent domain.

Figure 9-9 Boundary of TOP which Includes the Lower Domain Boundary



Use Model

Use the `lower_domain_boundary` design attribute to enable this feature. Following is the syntax:

```
set_design_attributes -elements {element_list} -  
attribute lower_domain_boundary TRUE/true/FALSE/false
```

This feature is enabled for all domains scoped at and below the current scope where the command is executed. For example,

```
set_design_attributes -elements {.} -attribute  
lower_domain_boundary TRUE
```

Key Points to Note

- The default value of `lower_domain_boundary` is FALSE. The definition of the lower boundary is enabled or disabled for the entire design irrespective of the scope specified in the `set_design_attributes` command.
- The `lower_domain_boundary` attribute can be set on the top scope before defining any power domains for the design and the value matches with any explicit setting elsewhere in the design.
- The `lower_domain_boundary` attribute can be set on the lower scope with FALSE value when the top scope is set to TRUE. For more information, see “[Block-Level Control of Lower Domain Boundary](#)” .
- If there are no power domains and explicit settings in the design, the `lower_domain_boundary` attribute is allowed to change on the top scope.

Examples

Example-1

Consider the following code:

```
set_scope /
set_design_attributes -elements { mid_inst } -attribute
lower_domain_boundary FALSE
```

In this example, attribute is set on `mid_inst` because the value `false` matches with the default value on the top scope.

Example-2

Consider the following code:

```
set_scope /
set_design_attributes -elements { . } -attribute
lower_domain_boundary FALSE
set_design_attributes -elements { mid_inst } -attribute
lower_domain_boundary FALSE
```

In this example, attribute will be set on `mid_inst` because the value `false` matches with the explicit setting on the top scope.

Example-3

Consider the following code:

```
set_scope /
set_design_attributes -elements { . } -attribute
lower_domain_boundary TRUE
create_power_domain TOP
set_scope mid_inst

set_design_attributes -elements { . } -attribute
lower_domain_boundary TRUE
```

```
create_power_domain MID  
set_scope /
```

In this example, the `lower_domain_boundary` attribute is set on both the top scope and `mid_inst` because the attributes are set before creating the power domains at the respective scopes.

Example-4

Consider the following code:

```
set_scope /  
set_design_attributes -elements { . } -attribute  
lower_domain_boundary TRUE  
set_design_attributes -elements { mid_inst } -attribute  
lower_domain_boundary TRUE  
create_power_domain MID -elements { mid_inst } -scope {  
mid_inst }  
set_design_attributes -elements { . } -attribute  
lower_domain_boundary FALSE
```

In this example, you cannot change the attribute value at the top scope to `FALSE` as there is a power domain below the top scope. Even if there is no power domain `MID`, attribute value does not change as there is a setting on `mid_inst`.

Block-Level Control of Lower Domain Boundary

VCS NLP supports block-level control of lower domain boundary. The following sections describe this feature in detail:

- [Support for OFF-Within-ON Integration](#)
- [Support for ON-Within-OFF Integration](#)

Support for OFF-Within-ON Integration

VCS NLP allows you to set the `lower_domain_boundary` attribute on the lower scope with `FALSE` value when the top scope is set to `TRUE` (OFF-within-ON block).

Use Model

Following is the use model of the OFF-within-ON block configuration:

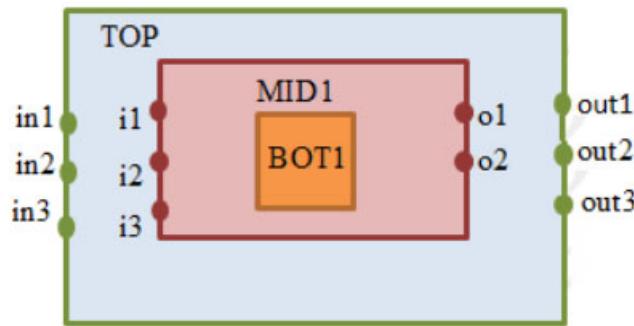
```
set_design_attributes -elements { . } -attribute  
lower_domain_boundary TRUE  
  
set_design_attributes -elements <scope> -attribute  
lower_domain_boundary FALSE
```

Key Points to Note

- A scope without the `lower_domain_boundary` attribute setting inherits the setting from its attributed ancestor.
- The `lower_domain_boundary` attribute setting applies to all the power domains scoped at or below the design or the scope having the setting.
- The `lower_domain_boundary` attribute is not allowed to change once it is set.

Examples

Consider the following design structure:



Example-1

```
set_scope /  
set_design_attributes -elements { . } -attribute  
lower_domain_boundary TRUE  
set_design_attributes -elements { MID1/BOT1 } -attribute  
lower_domain_boundary FALSE  
create_power_domain PDT -include_scope  
create_power_domain PDB -elements { MID1/BOT1 } -scope {  
MID1/BOT1 }
```

In this example, MID1/BOT1 is a valid OFF-within-ON block because the domain PDB is scoped at MID1/BOT1 and all the scopes above MID1/BOT1 have implicit/explicit TRUE setting.

Support for ON-Within-OFF Integration

VCS NLP allows you to set the lower_domain_boundary attribute on the lower scope with TRUE value when the top scope is set to FALSE (ON-within-OFF block).

Use Model

Following is the use model of the ON-within-OFF block configuration:

```
set_design_attributes -elements { . } -attribute  
lower_domain_boundary FALSE  
  
set_design_attributes -elements <scope> -attribute  
lower_domain_boundary TRUE
```

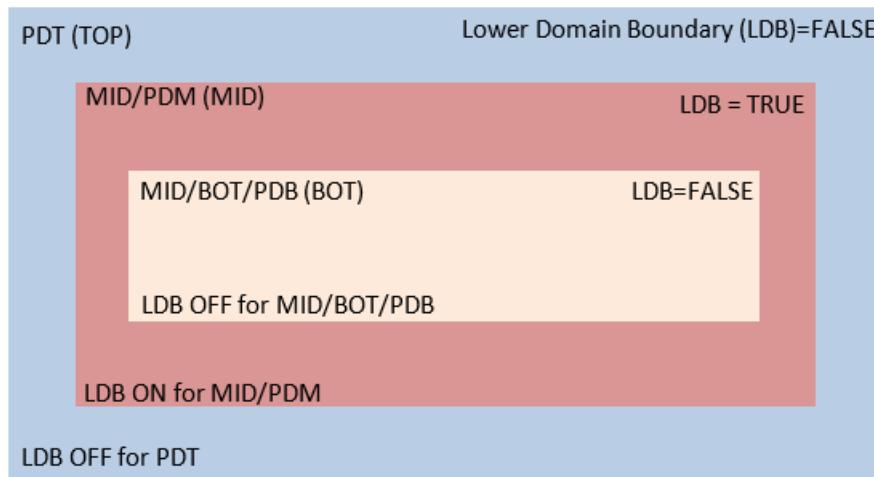
Example

Consider the following test case and design structure:

Test case:

```
set_design_attributes -elements { . } -attribute  
lower_domain_boundary FALSE  
set_design_attributes -elements {MID} -attribute  
lower_domain_boundary TRUE  
set_design_attributes -elements {MID/BOT} -attribute  
lower_domain_boundary FALSE  
create_power_domain PDT -include_scope  
create_power_domain PDM -elements {MID} -scope MID  
create_power_domain PDB -elements {MID/BOT} -scope MID/BOT
```

Design Structure:



In this example, `MID` is a valid ON-within-OFF block because the domain `PDM` is scoped at `MID` and the scope above `MID` have explicit `FALSE` setting.

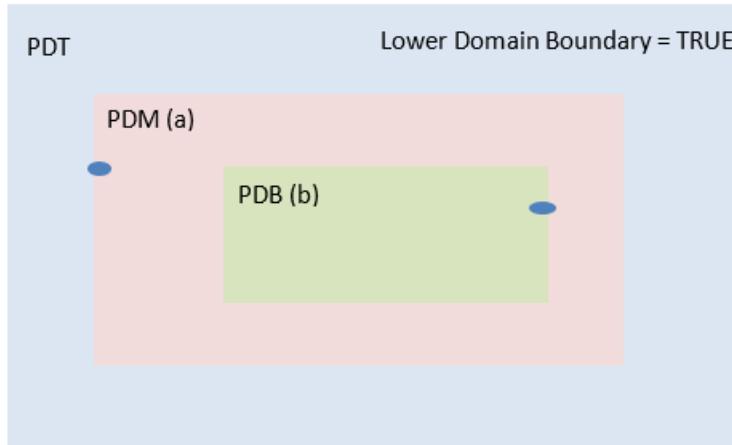
Location Parent Strategy Support on Lower Domain Boundary

Location parent (`-location parent`) strategy is supported on blocks with the `lower_domain_boundary` attribute set to `TRUE`. However, if this strategy is specified on a lower domain boundary port, it is ignored. Therefore, these strategies play no role in isolation precedence. The following example describes the ignore case:

Example:

```
set_design_attributes -elements {.} -attribute
lower_domain_boundary TRUE
create_power_domain PDT
create_power_domain PDM -elements {a}
create_power_domain PDB -elements {a/b}
set_isolation ISO1 -domain PDM -applies_to inputs -location
parent
```

Design Structure:



Power domain PDB is not a parent domain of PDM, so inserting isolation on the lower domain boundary is not supported. Therefore, ISO1 is implemented only at the upper boundary and ignored from implementation on the lower boundary.

Restricting Application of Filters to Specified Boundary

VCS NLP supports the `-applies_to_boundary` option for the `set_isolation` command. This option restricts the application of filters to the specified boundary. Following is the syntax:

```
-applies_to_boundary <lower | upper | both>
```

- Default value is `upper` for a domain with lower domain boundary OFF (`lower_domain_boundary FALSE`) or with no lower domain boundary setting.
- Default value is `both` for a domain with lower domain boundary ON (`lower_domain_boundary TRUE`).

This option has no impact on strategy precedence.

VCS NLP issues the

`UPF_INVALID_ELEMENT_VALUE_WITH_LOCATION` warning message if `-applies_to_boundary lower` is specified and there is no lower boundary for domain or no lower boundary elements are specified in the element list. Similarly, VCS NLP issues this warning message when `-applies_to_boundary upper` is specified and there are no upper boundary elements specified in the element list.

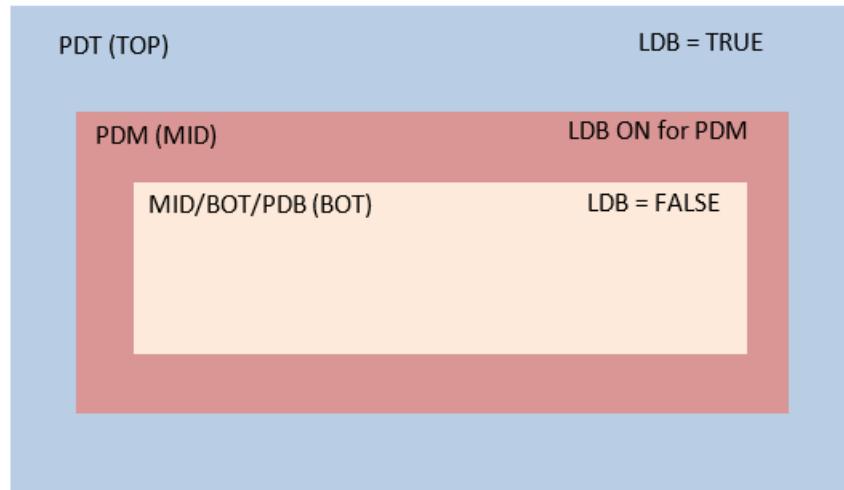
Example

Consider the following test case and design structure:

```
set_design_attributes -elements { . } -attribute
lower_domain_boundary TRUE
set_design_attributes -elements { MID/BOT } -attribute
lower_domain_boundary FALSE
create_power_domain PDT -include_scope
create_power_domain PDM -elements { MID }
create_power_domain PDB -elements { MID/BOT } -scope MID/BOT
set_isolation ISO1 -domain PDT -applies_to output
set_isolation ISO2 -domain PDM -applies_to input -
applies_to_boundary lower
set_isolation ISO3 -domain MID/BOT/PDB -applies_to both
```

- ISO1 applies to both lower and upper domain boundary (highconn and lowconn)
- ISO2 applies to lower domain boundary (highconn)
- ISO3 applies to upper domain boundary (lowconn)

Design Structure:



The following table describes how VCS NLP treats location parent strategy in the presence of `-applies_to_boundary` option and lower domain boundary:

lower_domain_boundary	location	-applies_to_boundary	Expected Behavior
TRUE	parent	upper	Implement strategy
		lower or both (default)	<p>Domain-level strategy:</p> <ul style="list-style-type: none"> Warn and ignore on lower domain boundary. <p>Element-level strategy:</p> <ul style="list-style-type: none"> Drop the lower domain elements, implement for the valid elements.
FALSE (default)	parent	upper (default)	Implement strategy
		lower or both	<p>Domain-level strategy:</p> <ul style="list-style-type: none"> Warn and ignore on lower domain boundary. <p>Element-level strategy:</p> <ul style="list-style-type: none"> Drop the lower domain elements, implement for the valid elements.

Limitation

- VCS NLP does not support `-location fanout` on the lower domain boundary elements.

Behavior of the Isolation Commands

The `set_isolation` command identifies the specified pin and port elements at the lower boundary interface of the specified domain. Also, the `-applies_to` option considers the lower boundary pins of the domain while filtering the objects of the domain-level strategies.

Commands Supported for Delayed Error Check

Domain boundary error checks on elements specified in the `set_isolation` command are performed at the end of reading UPF. This allows you to specify the lower domain boundary ports in the isolation element list even before the lower power domain is defined.

Consider the following example:

```
create_power_domain TOP
set_isolation out_iso -domain TOP -elements { BOT/* } ...
create_power_domain BOT -elements { BOT }
```

In this example, pins of `BOT` are mentioned as lower domain boundary pins of `TOP` before the creation of the domain `BOT`. VCS NLP does not issue an error message due to the delayed error check.

Impact on Isolation Engine

This section describes the following topics:

- General isolation strategies (without –elements)
- Specific Isolation Strategies (with –elements)
- Inserting Back-to-Back Isolation Cells in the Lower Domain Boundary
- Order of Precedence of Isolation Strategies

General isolation strategies (without –elements)

Isolation strategies of a power domain apply to its boundary pins. This feature expands the definition of the domain boundary pins to the lower domain-boundary pins. This means that the isolation strategies of a domain also apply to its lower boundary.

Output pins of a nested power domain are the input pins for the domain surrounding it, and the input pins of the nested domain are the output pins of the surrounding domain.

Therefore, any isolation strategies that apply to the input pins of a domain also applies to the output pins of the root cells of the domains nested inside it. The isolation strategies that apply to the output pins of a domain also applies to the input pins of the root cells of the domains nested inside it.

Figure 9-10 Output Isolation Strategy of TOP Applies to the Input of BOT

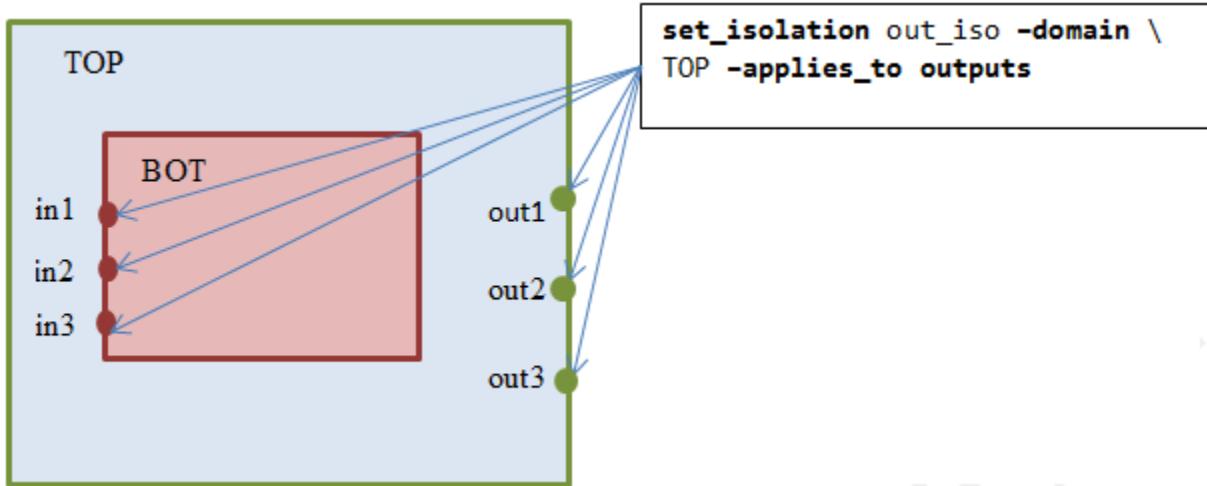
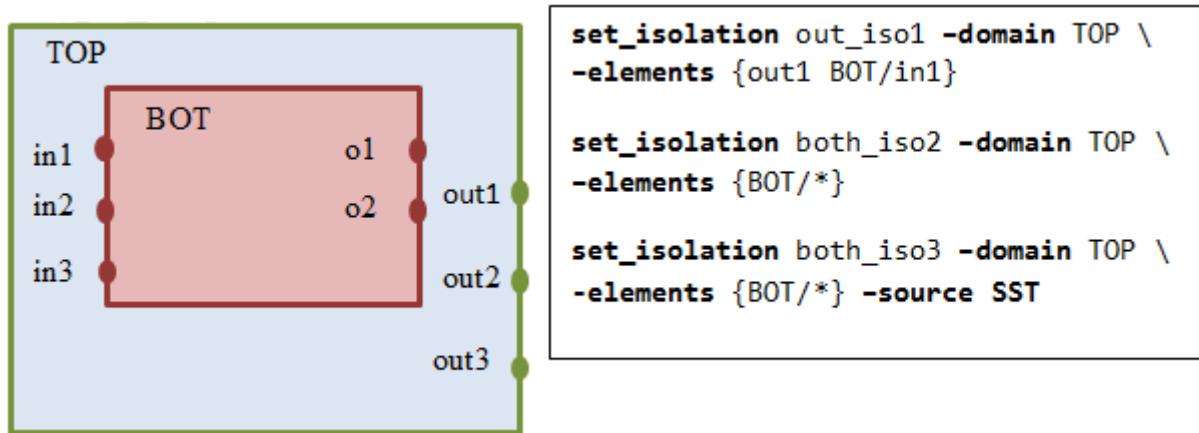


Figure 9-10 shows that the isolation strategy `out_iso` of the power domain `TOP` also applies to the input pins of the `BOT` block. Those pins are considered as outputs of the `TOP` domain. This means that with `lower_domain_boundary` enabled, `out_iso` results in 6 isolation cells for domain `TOP`, whereas without `lower_domain_boundary`, `out_iso` results in 3 isolation cells only.

Specific Isolation Strategies (with `-elements`)

Isolation strategies that specifically choose the pins to which they are applied do not change. They continue to apply to the pins and ports in the `-elements` option of the strategy. However, the isolation strategies are able to refer to the pins on the lower boundary of the domain in their `-elements` option. This enables you to target an isolation strategy specifically to the lower boundary pin(s).

Figure 9-11 Isolation Strategies of TOP Refer to the Lower Boundary Pins



In Figure 9-11, `out_isol1` shows how the pin `BOT/in1` can be specified as one of the elements of the isolation strategies of the `TOP` power domain.

`both_iso2` shows how all the pins of the `BOT` domain can be specified as the lower boundary of the domain `TOP` by specifying all the pins of the cell `BOT` in `-elements`. This strategy applies to all the pins in the element list.

`both_iso3` is a `-source`, `-sink`, or `-diff_supply_only` based strategy with no `-applies_to`. The target objects for this strategy are all the pins (both input and output) of `BOT`, and the strategy finally applies to only those pins which satisfy the `-source`, `-sink`, or `-diff_supply_only` filter.

Inserting Back-to-Back Isolation Cells in the Lower Domain Boundary

You can specify isolation strategies such that two isolation cells can be inserted on the same power domain boundary pin. One of these cells is inserted in the power domain and the other is inserted in its surrounding domain.

In Figure 9-12, the boundary pin `BOT/in1` of power domain `BOT` is subjected to two isolation strategies. One is the isolation strategy `bot_iso` specified on the domain `BOT`, and the other is the isolation strategy `top_iso` specified for `TOP`. The isolation strategy `top_iso` applies to `BOT/in1` because `BOT/in1` lies on the lower boundary of the `TOP` domain.

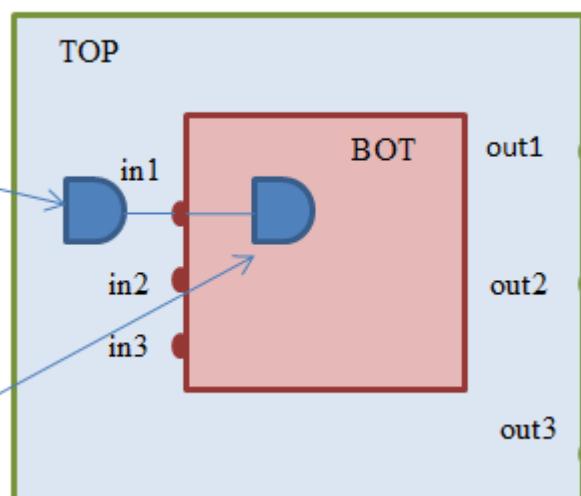
Figure 9-12 Inserting Back-to-Back Isolation Cells

```
set_isolation top_iso \
-domain TOP -elements BOT/in1

set_isolation_control top_iso \
-domain TOP -location self

set_isolation bot_iso -domain \
BOT -elements BOT/in1

set_isolation_control bot_iso \
-domain BOT -location self
```



Order of Precedence of Isolation Strategies

This feature does not affect the precedence order. The precedence order is applied to the strategies on each of HighConn and LowConn sides of a domain boundary.

Limitations

- Isolation strategies with only `-location self` are supported.
- VCS NLP does not support placing the isolation cells of the modport type ports on the lower domain boundary.

Implementing Assertion Checks for Isolation Cells

The following compile time options allow you to implement assertion checks to ensure stable output value from an isolation cell when its enable signal is asserted or deasserted:

`-power=clamp_toggle_check_on_iso_enable`

Adds an assertion that checks for a stable value on output of an isolation cell only when isolation is enabled. VCS NLP issues a warning message when there is a toggle on the output of the isolation cell. Following is the sample warning message:

```
[100000 ps] [WARNING] [LP_ISO_OUT_TOGGLE] Output of
isolation cell toggled from 'St0' to 'HiZ' when isolation
enable was asserted. Isolation output:
top.I1.var2_UPF_ISO, Enable: top/isoen, Strategy: top/
PD_M/isolation/isol (self).
```

`-power=clamp_toggle_check_on_iso_disable`

Adds an assertion that checks for a stable value on output of an isolation cell only when isolation is disabled. VCS NLP issues a warning message when there is a toggle on the output of the isolation cell. Following is the sample warning message:

```
[200000 ps] [WARNING] [LP_ISO_OUT_TOGGLE] Output of
isolation cell toggled from 'St0' to 'St1' when isolation
enable was de-asserted. Isolation output:
top.I1.var2_UPF_ISO, Enable: top/isoen, Strategy: top/
PD_M/isolation/isol (self).
```

`-power=clamp_toggle_check`

If you use this option, assertion check happens when isolation is enabled/disabled. This ensures stable output value from an isolation cell when its enable signal is asserted or deasserted.

Updating Isolation Supplies

The `-isolation_power_net`, `-isolation_ground_net`, and `-isolation_supply_set` options allow you to specify the supply net or supply set for a given `set_isolation` command invocation.

VCS NLP supports the `-update` option with the following `set_isolation` command options:

- `-isolation_power_net`
- `-isolation_ground_net`
- `-isolation_supply_set`

The following sections describe the usage of the `-update` option with the above isolation options:

- “[Using Isolation Power and Ground Nets With the -update Option](#)”
- “[Using `-isolation_supply_set` With the -update Option](#)”

Using Isolation Power and Ground Nets With the -update Option

The `-isolation_power_net` and `-isolation_ground_net` options for a given isolation strategy can be specified incrementally by using the `-update` option. Following is the use model:

```
set_isolation isolation_name -elements  
element_list
```

```
set_isolation isolation_name -elements  
element_list -isolation_power_net net_name -  
isolation_ground_net net_name -update
```

Example:

```
set_isolation ISO -elements {signal1 signal2}  
  
set_isolation ISO -elements {signal1 signal2} -  
isolation_power_net VDD -isolation_ground_net VSS  
-update
```

For a given isolation strategy, VCS NLP issues error messages in the following cases:

- If the `-isolation_power_net` or `-isolation_ground_net` option is specified more than once with or without the `-update` option.

Example:

```
set_isolation ISO -elements {signal1 signal2} -  
isolation_power_net VDD -isolation_ground_net  
VSS  
set_isolation ISO -elements {signal1 signal2} -  
isolation_ground_net VSS -update
```

Following is the sample error message:

```
Error-[UPF_IDCO] Invalid duplicate command option  
supply.upf, 56  
Specifying option '-isolation_power_net' more than once  
is not allowed. Please correct the UPF file and re-  
compile.
```

- If the `-isolation_power_net` option is specified along with the `-isolation_supply_set` and `-update` options.

Example:

```
set_isolation ISO -elements {signal1 signal2} -  
isolation_power_net VDD -isolation_ground_net  
VSS  
set_isolation ISO -elements {signal1 signal2} -  
isolation_supply_set {SSA} -update
```

Following is the sample error message:

```
Error-[UPF_ATMOST_ONEOF] Incorrect command options  
supply.upf, 56  
It is not legal to specify more than one of '(-  
isolation_supply_set, ATLEAST_ONE_OF(-  
isolation_power_net, -isolation_ground_net))' in command  
'set_isolation'.
```

Using **-isolation_supply_set** With the **-update** Option

The **-isolation_supply_set** option for a given isolation strategy can be specified incrementally by using the **-update** option.

Following is the use model:

```
set_isolation isolation_name -elements  
element_list  
  
set_isolation isolation_name -elements  
element_list -isolation_supply_set supply_set_name  
-update
```

Example:

```
set_isolation isol -domain PD_A  
  
set_isolation isol -domain PD_A -  
isolation_supply_set {SSA} -update
```

For a given isolation strategy, VCS NLP issues error messages in the following cases:

- If the `-isolation_supply_set` option is specified more than once with or without the `-update` option.

Example:

```
set_isolation isol -domain PD_A -isolation_supply_set {SSA}  
set_isolation isol -domain PD_A -isolation_supply_set {SSB} -update
```

Following is the sample error message:

```
Error-[UPF_IDCO] Invalid duplicate command option  
supply.upf, 56  
Specifying option '-isolation_supply_set' more than once  
is not allowed.  
Please correct the UPF file and re-compile.
```

- If multiple supply sets in `-isolation_supply_set` are specified with the `-update` option.

Example:

```
set_isolation isol -domain PD_A  
set_isolation isol -domain PD_A -isolation_supply_set {SSA SSB} -update
```

Following is the sample error message:

```
Error-[UPF_IMIES] Invalid multiple iso enables/supplies  
supply.upf, 52  
Isolation 'top/PD_A/isolation/isol' is defined with 2  
supplies, but isolation model is not specified for  
multiple supplies.  
Please use map_isolation_cell UPF command to specify a  
model for mapping the isolation.
```

Specifying Isolation Strategies Without Defining Isolation Enable

The `set_isolation/set_isolation_control` command allows you to specify isolation strategies without defining isolation enable (`-isolation_signal/-isolation_sense`). However, this is supported only when the corresponding `map_isolation_cell` command is specified.

This enhancement allows you to use isolation models or cells without isolation enable in a simulation by mapping them to a strategy.

Example

This example shows a latch based isolation model mapped with isolation policy without isolation enable pin specified.

Example 9-1 Verilog model for the isolation cell: design.v

```
module no_en_ISO(input D, output Z, input DVDD, input DVDB,
input GND);
reg DO;
always@(*)
begin
if ((DVDD === 1`b1) & ((DVDB === 1`b1))
    DO = DI;
else if ((DVDB !== 1`b1))
    DO = 1`bx;
end
end
endmodule
```

Example 9-2 test.upf

```
create_supply_set ssT -function {power VDD} -function
{ground VSS}
create_supply_set ss1 -function {power VDD1} -function
```

```

{ground VSS}

set_isolation iso_1 \
    -domain PD1 \
    -elements {t1/out} \
    -isolation_supply_set {ssT ss1} \
    -location parent

map_isolation_cell iso_1 \
    -domain PD1 \
    -lib_model_name DMLSDBF12V \
    -port {DVDD VDD1} \
    -port {DVDDB VDD} \
    -port {GND VSS}

```

Following is the simulation output:

DI (Data Input)	DVDD (Primary Power)	DVDDB (Backup Power)	DO (Data Output)
any	ON	ON	DI
any	OFF	ON	Latch
any	OFF	OFF	X
any	ON	OFF	X

10

Handling of Constants for Corruption and Isolation

VCS NLP supports the following behaviors for corruption and isolation of constants. Literals `1'b1` and `1'b0` in port maps or RHS of continuous assignments are treated as constants. The following topics describe this feature:

- [Default Behavior](#)
- [Skip Corruption on Constants](#)

Default Behavior

All literal constants in port map or all variables that are assigned to the literal constants using the continuous assign statements are powered by the power domain where the declaration or assignment is made.

All constants are corrupted like any other logic when either Power or Ground of the domain goes OFF.

The constants are corrupted like any other logic, and hence need isolation. Isolation strategy can be specified on all constant driven paths which cross the domain boundary. Constants in the port map of cell mapped (macro) instances are not corrupted unless there is isolation on that port.

Skip Corruption on Constants

The `-power=pass_thru_const` compile-time option allows you to skip corruption on constants.

Note:

Any isolation strategy defined is applied on constants.

11

State Retention Support

The retention commands specify the strategy for inserting retention cells inside switched (power-down) domains. VCS NLP supports the `set_retention` and `set_retention_control` commands to model the state retention schemes.

This chapter describes the state retention support in the following sections:

- [Introduction](#)
- [Using set_retention command](#)
- [Supported Retention Schemes](#)
- [Conditions](#)
- [Mutexes](#)
- [Parameters](#)

- Creating Retention List
- Automated Assertions Supported for Retention
- Generic Handles
- Hierarchical Flop Inference Report
- Limitations of Retention Support

Introduction

The `set_retention` command specifies the registers in the domain that are implemented as retention registers and identifies the save and restore signals for the retention functionality. Only the registers implied in the elements are given retention capabilities.

Following is the syntax of the `set_retention` command:

```
set_retention retention_name
  -domain domain_name
  [-elements element_list]
  [-retention_power_net net_name] [-retention_ground_net net_name]
  [-retention_supply_set ret_supply_set] [-no_retention]
  [-save_signal {{logic_net <high | low | posedge | negedge>}}]
  [-restore_signal {{logic_net <high | low | posedge | negedge>}}]
  [-save_condition {{boolean_function}}]
  [-restore_condition {{boolean_function}}]
  [-retention_condition {{boolean_function}}]
  [-use_retention_as_primary]
  [-parameters {<<RET_SUP_COR | NO_RET_SUP_COR> |
    <SAV_RES_COR | NO_SAV_RES_COR> >}]
  [-instance1 {{instance_name [signal_name]}]}
  [-transitive <TRUE | FALSE>]
  [-update]
```

¹This command argument is parsed and ignored by VCS NLP.

Using `set_retention` command

The `set_retention` command specifies which registers in the domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality.

If a design instance is specified, all registers within the design element acquire the specified retention strategy. If a process is specified, all registers inferred by the process acquire the specified retention strategy.

If a reg, signal, or variable is specified, and the object is a sequential element, then the implied register acquires the specified retention strategy. Any specified reg, signal, or variable that does not infer a sequential element is not changed by this command.

If `-elements` is specified, the element names outside the extent of `domain_name` are excluded. When `-elements` is not specified, this is equivalent to using the elements list that defines the power domain.

If the retention power and retention ground nets are specified, an implicit retention supply set is created and used with the specified strategy.

The retention power net serves the power function in the retention supply set and the retention ground net serves the ground function in the retention supply set.

If the retention power net is specified, but the retention ground net is not specified, then the primary supply set ground function of the domain is used as the retention ground. If the retention ground net is specified without the retention power net, then the primary supply set power function of the domain is used as the retention power.

VCS NLP generates an error message, if a retention supply set is specified and a retention supply net is individually defined.

Support for the -update Option

VCS NLP supports the `-update` option with the `set_retention` command.

The `-update` option provides additional information for a previous command with the same strategy name, domain name, and executed in the same scope.

Use Model

The `-update` option adds information to the base command executed in the same scope of the power domain for which the inferred cells are defined.

Syntax:

```
set_retention retention_name
  -domain domain_name
  [-elements element_list]
  [-update]
```

Example:

```
set_retention test_ret
```

```

-domain PD1
-elements i_core/q_dff
-save_signal {en posedge}
-restore_signal {en negedge}

set_retention test_ret
-domain PD1
-update
-elements i_core/q_tff

```

The `-update` option adds two elements (`i_core/q_dff` and `i_core/q_tff`) to the retention strategy `test_ret`.

Using `-no_retention` Option

The `-no_retention` option does not apply retention on the specified elements. This option allows you to skip retention on a subset of registers in a power domain.

```

set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP

set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level high} -restore_signal
{restore_level high}

set_retention NO_RET -domain Island_Vdd -elements
{flop_inst3} -no_retention

```

Using `-use_retention_as_primary` Option

The `-use_retention_as_primary` option specifies that the storage element and its output are powered by the retention supply. As a result, simstate for the retention supply set is applied to the register's output.

```

set_retention TWO_PIN_RET_EDGE -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst2/outp1} -use_retention_as_primary

set_retention_control TWO_PIN_RET_EDGE -domain Island_Vdd
-save_signal {save_posedge} -restore_signal {restore
posedge}

```

Supported Retention Schemes

VCS NLP supports the following retention schemes:

1. Two-Pin Retention
 2. Single-Pin Retention
 3. Zero-Pin Retention
-

Two-Pin Retention

Separate save and restore signals are used to control save and restore operations. Both level-sensitive and edge-sensitive save/restore are supported.

```

set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd -
retention_power_net VDD_TOP -retention_ground_net VSS_TOP -
elements {flop_inst1/outp1}

set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level_high} -restore_signal
{restore_level_high}

set_retention TWO_PIN_RET_EDGE -domain Island_Vdd -
retention_power_net VDD_TOP -retention_ground_net VSS_TOP -
elements {flop_inst2/outp1}

set_retention_control TWO_PIN_RET_EDGE -domain Island_Vdd -

```

```
save_signal {save posedge} -restore_signal {restore posedge}
```

Semantics

1. Level-sensitive save/restore

Save operation is performed as long as the save signal is at active level.

Restore operation is performed as long as the restore signal is at active level.

2. Edge-sensitive save/restore

Save event is the specified edge of the save signal.

Restore event is the specified edge of the restore signal.

Example: outp1 is the primary register, save_outp1 is the shadow register

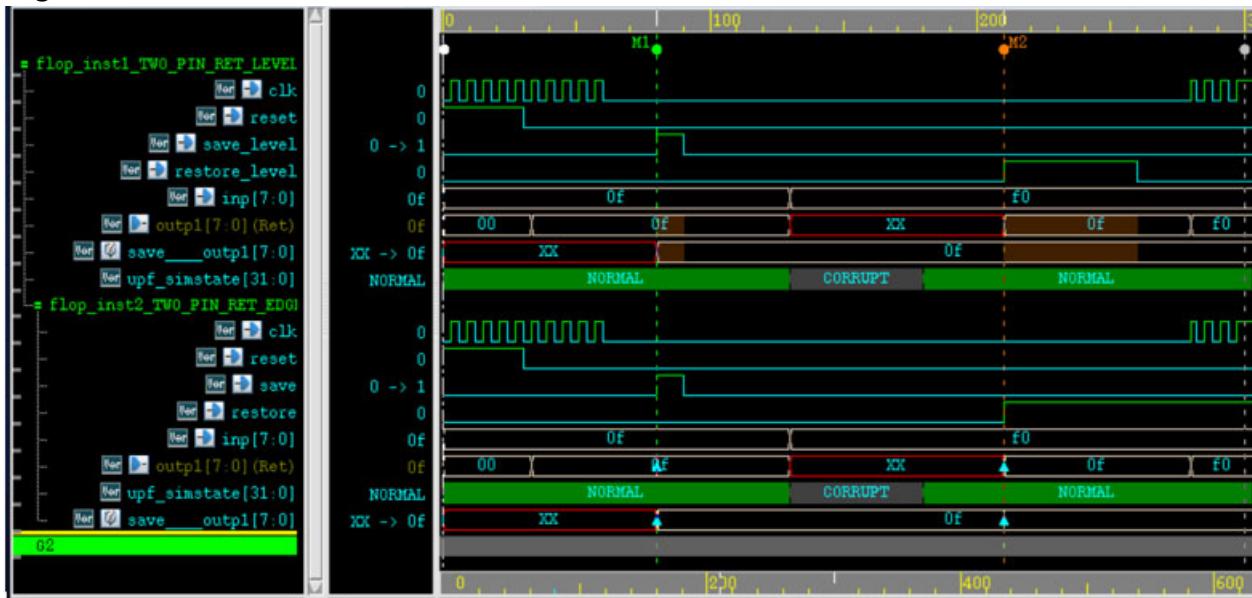
TWO_PIN_RET_LEVEL

- Save operation from time 80-90, since `save_level` is high.
- Restore operation from time 210-260, since `restore_level` is high.

TWO_PIN_RET_EDGE

- Save event at time 80, posedge on save.
- Restore event at time 210, posedge on restore.

Figure 11-1 Two-Pin Retention



Single-Pin Retention

Same signal is used as save and restore signal for controlling save and restore operations. Both level-sensitive and edge-sensitive save/restore are supported.

```
set_retention SINGLE_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst3/outp1}

set_retention_control SINGLE_PIN_RET_LEVEL -domain
Island_Vdd -save_signal {save_restore_level high}
-restore_signal {save_restore_level low}

set_retention SINGLE_PIN_RET_EDGE -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst4/outp1}

set_retention_control SINGLE_PIN_RET_EDGE -domain
Island_Vdd -save_signal {save_restore_posedge}
```

```
-restore_signal {save_restore negedge}
```

Semantics

1. Level-sensitive save/restore

Save operation is performed on the trailing edge specified with `-save_signal`. This indicates negedge for high and posedge for low.

Restore operation is performed as long as the restore signal is at active level.

2. Edge-sensitive save/restore

Save event is the specified edge of `-save_signal`.

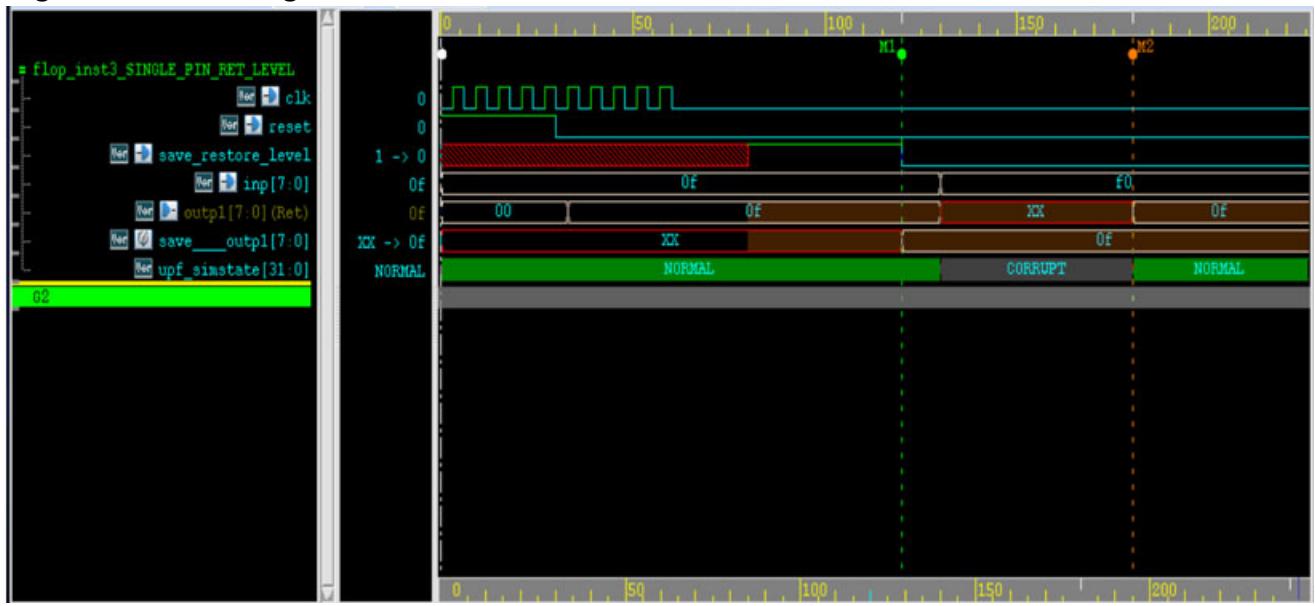
Restore event is the specified edge of `-restore_signal`.

Example: `outp1` is the primary register, `save_outp1` is the shadow register

`SINGLE_PIN_RET_LEVEL`

- Save event at time 120, trailing edge of `save_restore_level`.
- Restore operation from time 120-260 since `save_restore_level` is low.

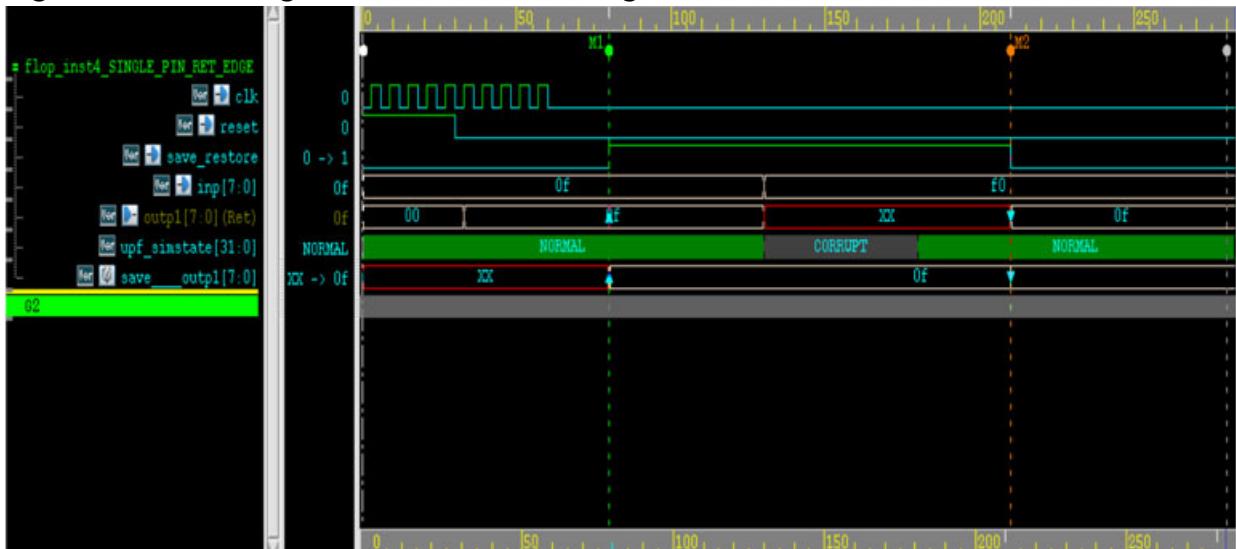
Figure 11-2 Single-Pin Retention Level



SINGLE_PIN_RET_EDGE

- Save event at time 80, posedge on `save_restore`
- Restore event at time 210, negedge on `save_restore`

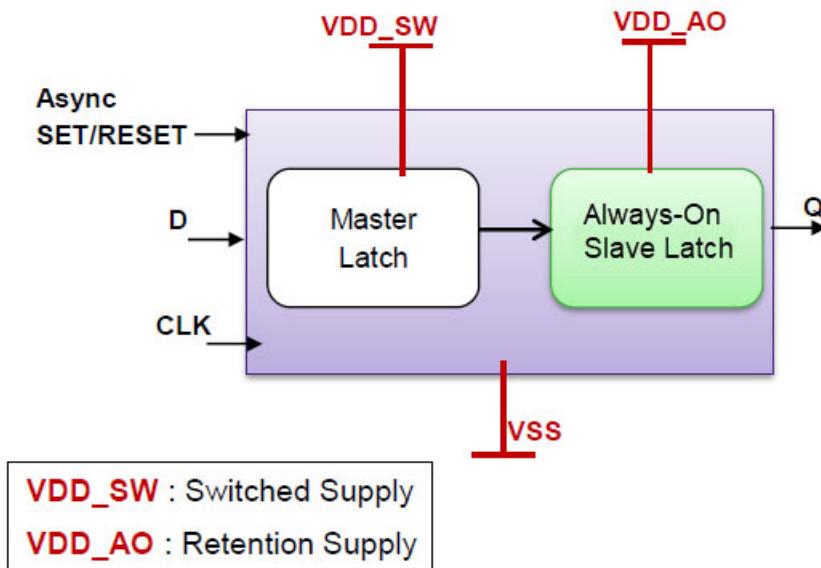
Figure 11-3 Single-Pin Retention Edge



Zero-Pin Retention

VCS NLP supports zero-pin retention registers. A zero-pin retention register, also known as a live-slave retention register, has no control pins. This type of retention register uses a master-slave structure in which the slave latch is a low-leakage block with an always-on power supply. [Figure 11-4](#) shows the internal structure of a zero-pin retention register.

Figure 11-4 Live-Slave Retention Register Model



The slave maintains the data during both normal operation and shutdown, so no save or restore pin is needed. This type of retention register is also known as a live-slave retention register because the slave portion of the register remains alive during shutdown to maintain the data.

Retention occurs when an isolation circuit holds the clock signal in an inactive state. For a rising edge clock, the inactive state is low or zero.

Figure 11-5 shows the sample zero-pin library model.

Figure 11-5 Zero-Pin Library Model

```

/* 0-pin Library Model - Example */

cell (ZPR_CLK_LOW) {
    retention_cell : @_pin_clk_low_retention ;
}

pg_pin (VDD_SW) {
    pg_type : primary_power;
    voltage_name : VDD_SW;
}
pg_pin (VDD_AO) {
    pg_type : backup_power;
    voltage_name : VDD_AO;
}
pg_pin (VSS) {
    pg_type : primary_ground;
    voltage_name : VSS;
}
pin (Q) {
    direction : output;
    function : "IQ2";
    related_ground_pin : VSS;
    related_power_pin : VDD_SW;
    power_down_function : "(CLK * !VDD_SW) +
    !VDD_AO + VSS";
}
}

pin (D) {
    related_power_pin : VDD_SW;
    related_ground_pin : VSS;
}
pin (RST) {
    related_power_pin : VDD_AO;
    related_ground_pin : VSS;
}
pin (CLK) {
    related_power_pin : VDD_AO;
    related_ground_pin : VSS;
}
clock_condition() {
    clocked_on : "CLK";
    hold_state : "L";
}
retention_condition() {
    required_condition : "!CLK * RST";
    power_down_function : "!VDD_SW + VSS";
}

```

UPF Example

Following is the UPF example for Zero-pin retention register:

Example 11-1 Zero-Pin Retention UPF Example

```
set_retention zpr_ret -domain PD_TOP \
    -retention_supply_set SS_RET \
    -elements {*RDFF*} \
set_retention_control zpr_ret -domain PD_TOP \
    -save_signal {retain high} \
    -restore_signal {retain low}
map_retention_cell zpr_ret -domain PD_TOP \
    -lib_cell_type {0_pin_ret_clk_low}
```

Conditions

This section describes the following topics:

- [-save_condition/-restore_condition](#)
 - [-retention_condition](#)
-

-save_condition/-restore_condition

You can further specify gating conditions for the save/restore events using the `-save_condition` and `-restore_condition` options. Restore event has highest priority over any other operation of the retention element (clock/set/reset). Restore condition can be used to give priority to clock/set/reset.

Condition must be a Boolean expression and save/restore events are gated if the Boolean expression evaluates to FALSE. Default value is TRUE.

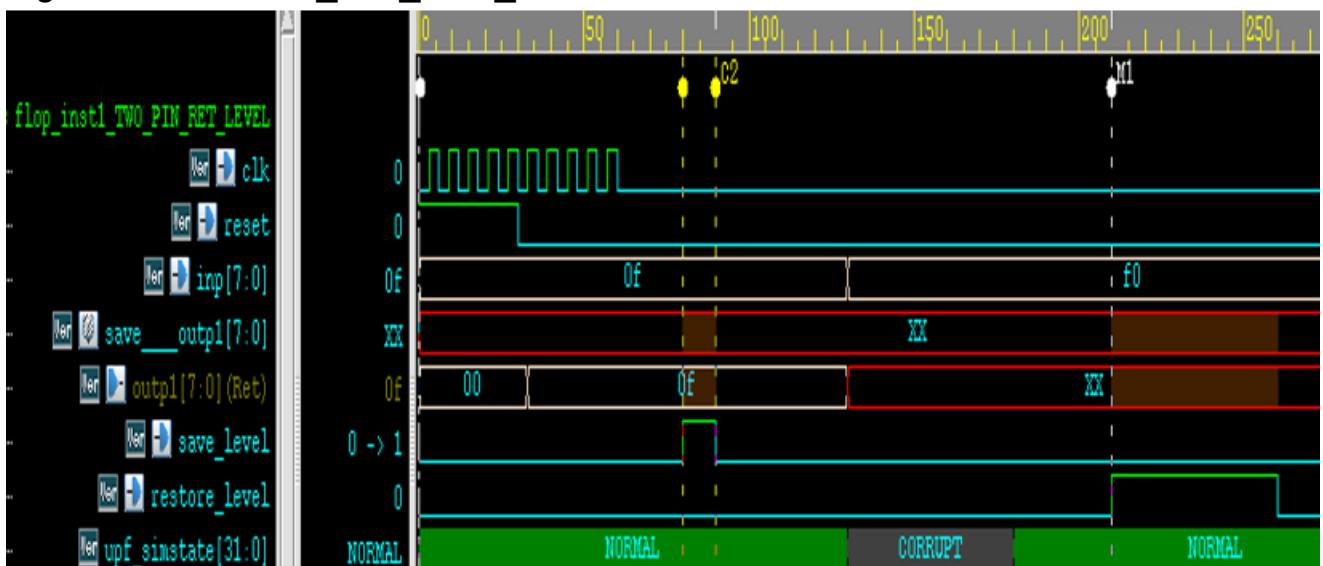
```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd  
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst1/outp1} -save_condition {clk}  
-restore_condition {!reset}  
  
set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd  
-save_signal {save_level high} -restore_signal  
{restore_level high}
```

Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_LEVEL

- No save operation between time 80-90 as the `save_condition` is FALSE. Shadow register retains its old value `8'hxx`.
- Restore operation between time 210-260 as `restore_condition` is TRUE. This operation restores the content of the shadow register.

Figure 11-6 TWO_PIN_RET_LEVEL



```
set_retention TWO_PIN_RET_EDGE -domain Island_Vdd
-retention_power net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst2/outp1} -save_condition {!clk}
-restore_condition {reset}
```

```
set_retention_control TWO_PIN_RET_EDGE -domain Island_Vdd -
-save_signal {save posedge} -restore_signal {restore posedge}
```

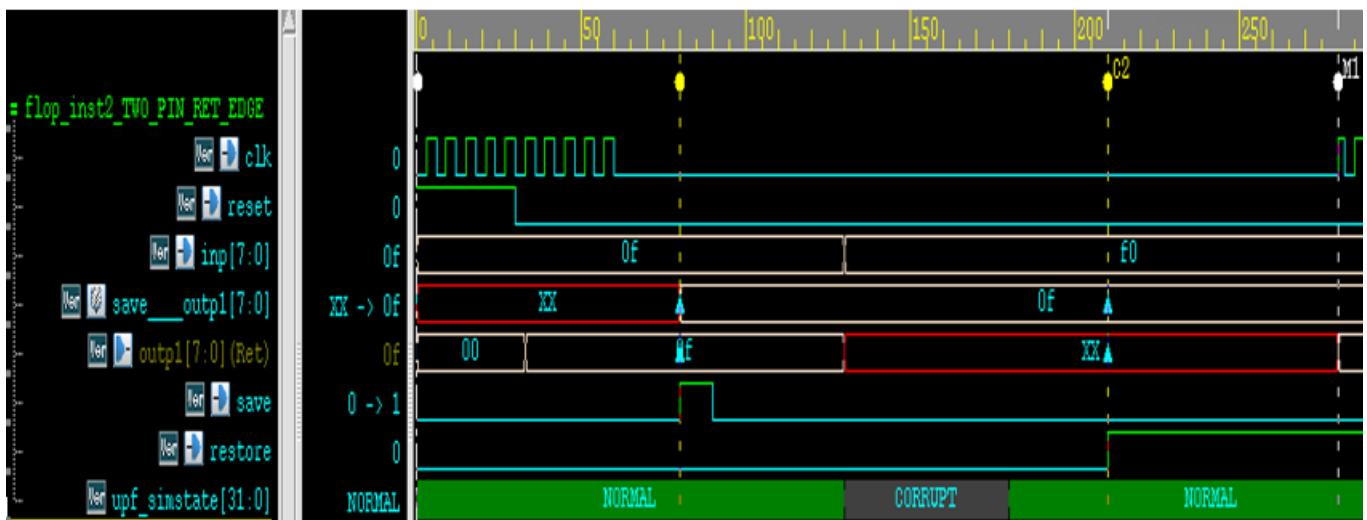
Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_EDGE

- Save operation at time 80 as the `save_condition` is TRUE.

- No Restore operation at time 210 as the `restore_condition` is FALSE. This operation corrupts the value on the primary register till the active clock edge.

Figure 11-7 TWO_PIN_RET_EDGE



-retention_condition

Boolean expression can be specified and the shadow register contents are corrupted, if the expression evaluates to FALSE when the primary supplies are OFF. The default value is TRUE.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst1/outp1} -retention_condition {clk}

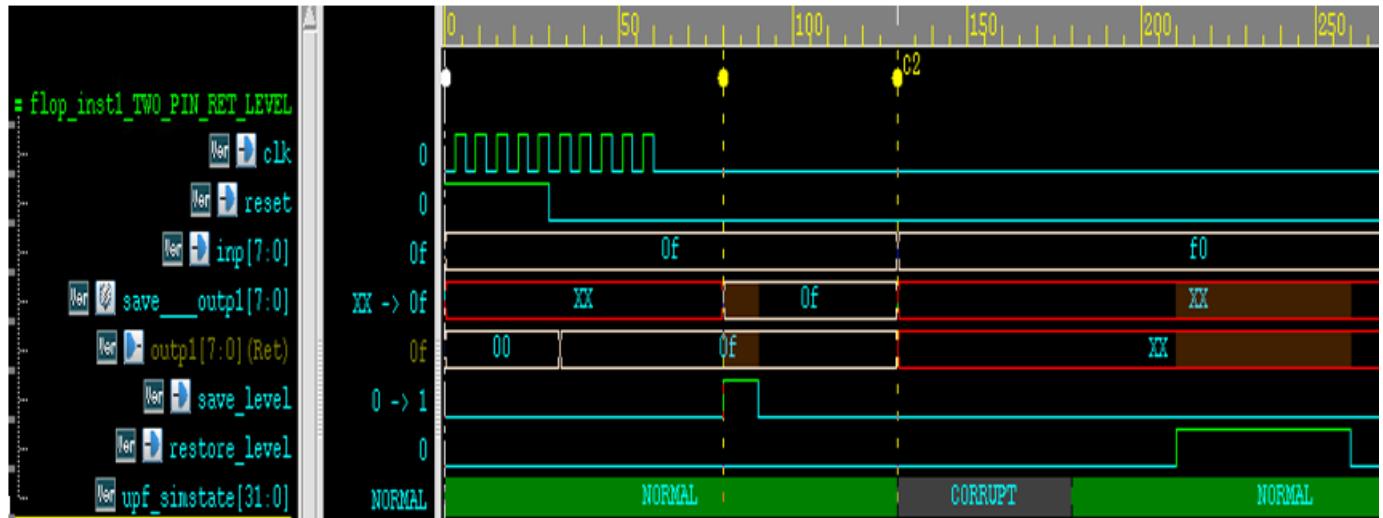
set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level high} -restore_signal
{restore_level high}
```

Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_LEVEL

- Save operation from time 80-90, since `save_level` is high.
- Shadow register corrupted on power down at time 130 as the `retention_condition` is FALSE.

Figure 11-8 TWO_PIN_RET_LEVEL



Mutexes

This section contains the following topics:

- [assert_s_mutex](#)
- [assert_r_mutex](#)
- [assert_rs_mutex](#)

assert_s_mutex

1. Defines mutual exclusion between save event and an active level/edge of other signal.

2. Assertion is reported.
3. In case of level sensitive save_signal, shadow register is corrupted if the mutual exclusion fails.
4. No corruption of shadow register in case of edge-sensitive save_signal.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst1/outp1}

set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level high} -restore_signal
{restore_level high} -assert_s_mutex {clk low}
```

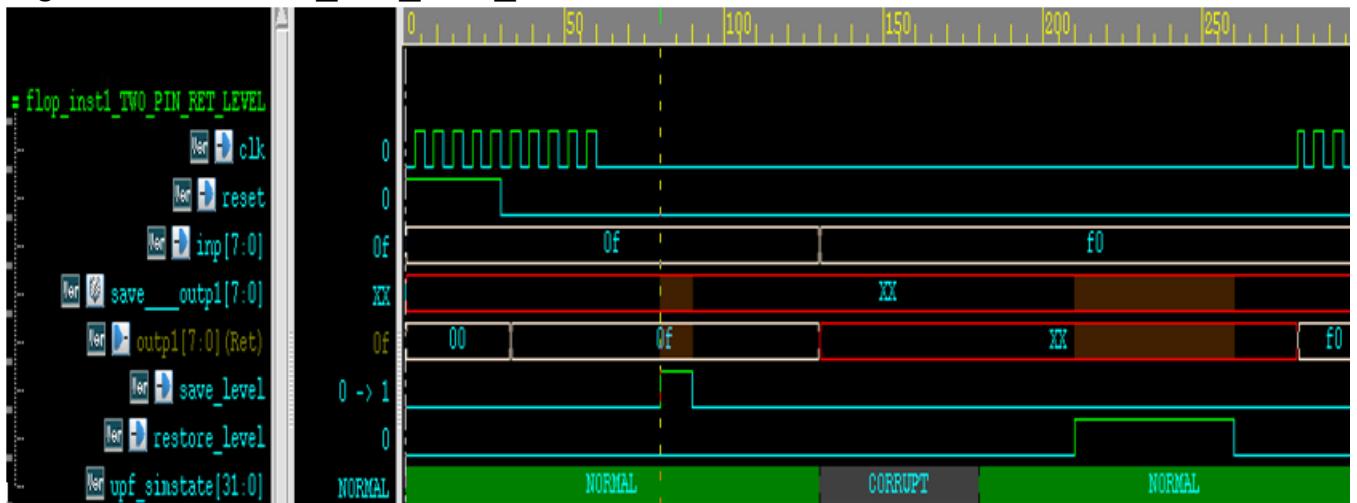
Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_LEVEL

- assert_s_mutex failure at time 80, shadow register corrupted
- Assertion reported at time 80 indicating mutex failure

```
[80] [ERROR] [LP_ASSERT_S_MUTEX] 'assert_s_mutex' defined
for retention strategy 'TWO_PIN_RET_LEVEL' of power domain
'testbench/top/Island_Vdd' failed. Save signal 'testbench/
top/save_level' ('high') and signal 'testbench/top/clk'
('low') are not mutually exclusive.
```

Figure 11-9 TWO_PIN_RET_LEVEL



```

set_retention TWO_PIN_RET_EDGE -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst2/outp1}

set_retention_control TWO_PIN_RET_EDGE -domain Island_Vdd
-save_signal {save posedge} -restore_signal {restore
posedge} -assert_s_mutex {clk low}

```

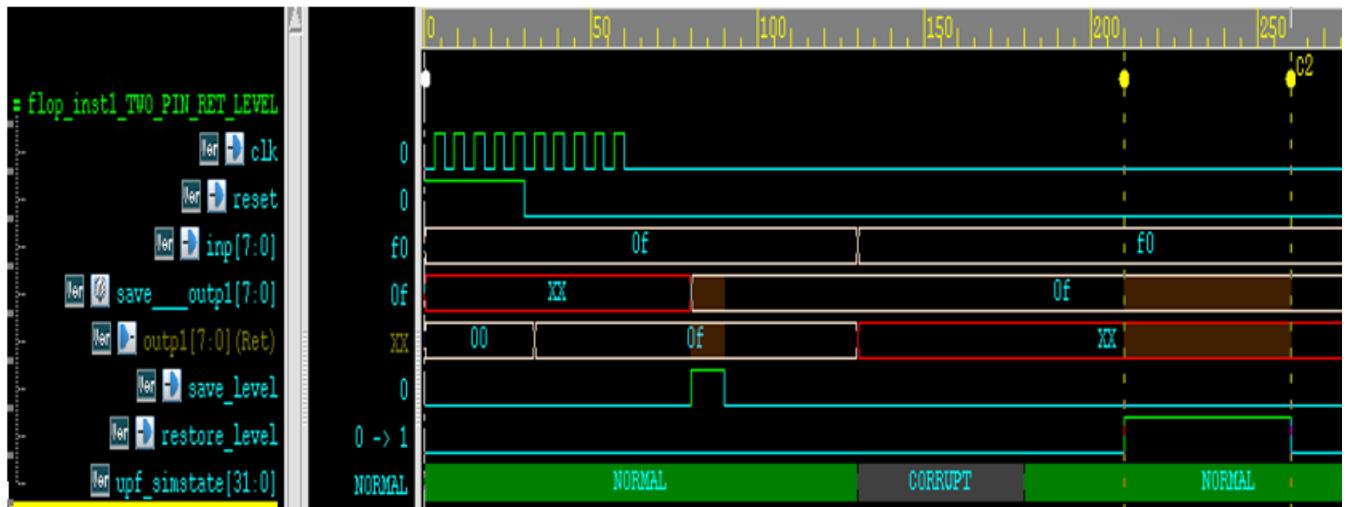
Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_EDGE

- assert_s_mutex failure at time 80, primary register content saved to shadow register.
- Assertion reported at time 80 indicating mutex failure.

```
[80] [ERROR] [LP_ASSERT_S_MUTEX] 'assert_s_mutex' defined
for retention strategy 'TWO_PIN_RET_EDGE' of power domain
'testbench/top/Island_Vdd' failed. Save signal 'testbench/
top/save' ('posedge') and signal 'testbench/top/clk' ('low')
are not mutually exclusive.
```

Figure 11-10 TWO_PIN_RET_EDGE



assert_r_mutex

1. Defines mutual exclusion between restore event and an active level/edge of other signal.
2. Assertion is reported.
3. In case of level sensitive `restore_signal`, primary register is corrupted if the mutual exclusion fails.
4. No corruption in case of edge-sensitive `restore_signal`.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst1/outp1}

set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level high} -restore_signal
{restore_level high} -assert_r_mutex {clk low}
```

Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_LEVEL

- Save operation from time 80-90 since save_level is high.
- assert_r_mutex failure at time 210, primary register corrupted.
- Assertion reported at time 210 indicating mutex failure.

```
[210] [ERROR] [LP_ASSERT_R_MUTEX] 'assert_r_mutex' defined  
for retention strategy 'TWO_PIN_RET_LEVEL' of power domain  
'testbench/top/Island_Vdd' failed. Restore signal  
'testbench/top/restore_level' ('high') and signal  
'testbench/top/clk' ('low') are not mutually exclusive.
```

Figure 11-11 TWO_PIN_RET_LEVEL



```
set_retention TWO_PIN_RET_EDGE -domain Island_Vdd  
-retention_power net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst2/outp1}
```

```
set_retention_control TWO_PIN_RET_EDGE -domain Island_Vdd  
-save_signal {save posedge} -restore_signal {restore  
posedge} -assert_r_mutex {clk low}
```

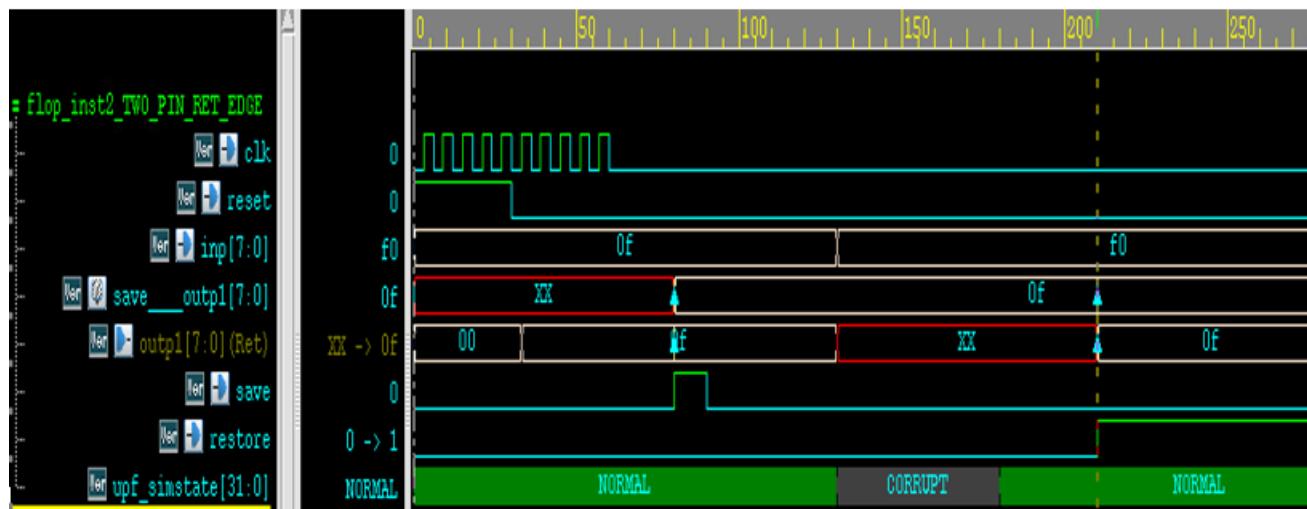
Example: `outp1` is the primary register, `save_outp1` is the shadow register

TWO_PIN_RET_EDGE

- Save operation from time 80-90, posedge on save.
- `assert_r_mutex` failure at time 210, restore operation performed.
- Assertion reported at time 210 indicating mutex failure.

```
[210] [ERROR] [LP_ASSERT_R_MUTEX] 'assert_r_mutex' defined  
for retention strategy 'TWO_PIN_RET_EDGE' of power domain  
'testbench/top/Island_Vdd' failed. Restore signal  
'testbench/top/restore' ('posedge') and signal 'testbench/  
top/clk' ('low') are not mutually exclusive.
```

Figure 11-12 TWO_PIN_RET_EDGE



assert_rs_mutex

1. Defines mutual exclusion between save and restore events and an active level/edge of other signal.

2. Assertion is reported for mutual exclusion failures on either/both save or/and restore.
3. In case of level-sensitive `save_signal`, shadow register is corrupted if the mutual exclusion fails with respect to the save event.
4. In case of level sensitive `restore_signal`, primary register is corrupted if the mutual exclusion fails with respect to the restore event.

```
set_retention SINGLE_PIN_RET_LEVEL -domain Island_Vdd
-retention_power net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst3/outp1}

set_retention_control SINGLE_PIN_RET_LEVEL -domain
Island_Vdd -save_signal {save_restore_level high}
-restore_signal {save_restore_level low} -assert_rs_mutex
{clk low}
```

Example: outp1 is the primary register, save_outp1 is the shadow register

SINGLE_PIN_RET_LEVEL

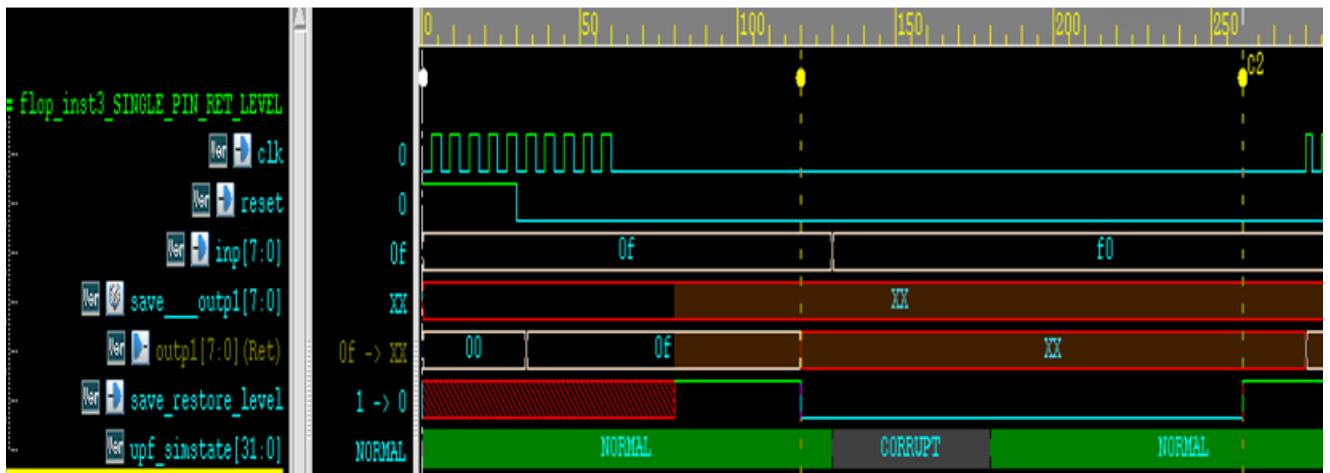
- `assert_s_mutex` failure at time 120 on trailing edge of `save_restore_level`, shadow register is corrupted.
- Assertion reported at time 120 indicating `assert_s_mutex` failure.
- `assert_r_mutex` failure from time 120-260, primary register is corrupted.
- Assertion reported at time 120 indicating `assert_r_mutex` failure.

```
[120] [ERROR] [LP_ASSERT_S_MUTEX] 'assert_s_mutex' defined
for retention strategy 'SINGLE_PIN_RET_LEVEL' of power
```

domain 'testbench/top/Island_Vdd' failed. Save signal 'testbench/top/save_restore_level' ('negedge') and signal 'clk' ('low') are not mutually exclusive.

[120] [ERROR] [LP_ASSERT_R_MUTEX] 'assert_r_mutex' defined for retention strategy 'SINGLE_PIN_RET_LEVEL' of power domain 'testbench/top/Island_Vdd' failed. Restore signal 'testbench/top/save_restore_level' ('low') and signal 'clk' ('low') are not mutually exclusive.

Figure 11-13 SINGLE_PIN_RET_LEVEL



Parameters

This section describes the following topics:

- [Supported Parameters with set_retention_control](#)
- [Supported Parameters with set_retention](#)

Supported Parameters with `set_retention_control`

`COR_AUTO`

Set by default. Enables corruption on primary and shadow registers due to mutex failures on level-sensitive restore and save signals respectively.

`COR_NONE`

Disables corruption on the primary and shadow registers due to mutex failures on level-sensitive restore and save signals respectively.

```
set_retention SINGLE_PIN_RET_LEVEL -domain Island_Vdd  
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst3/outp1}  
set_retention_control SINGLE_PIN_RET_LEVEL -domain  
Island_Vdd -save_signal {save_restore_level high}  
-restore_signal {save_restore_level low} -assert_rs_mutex  
{clk low} -parameters COR_NONE
```

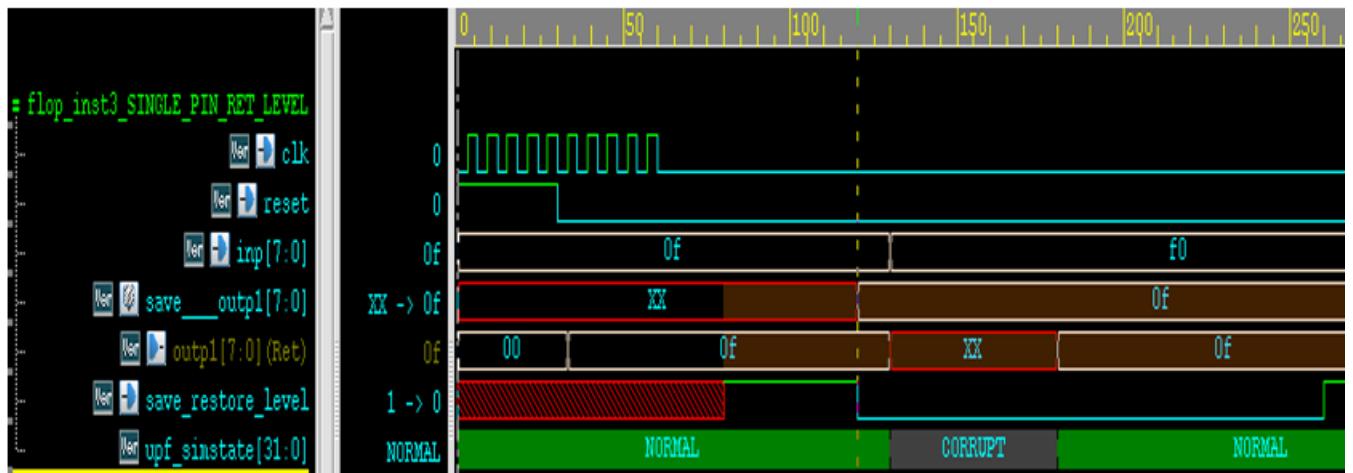
Example: `outp1` is the primary register, `save_outp1` is the shadow register

`SINGLE_PIN_RET_LEVEL`

- `assert_rs_mutex` failure at time 120 on trailing edge of `save_restore_level`. Value is saved to the shadow register, and no corruption happens on the shadow register.
- Assertion reported at time 120 indicating `assert_rs_mutex` failure.
- `assert_rs_mutex` failure from time 120-260, shadow register content restored to the primary register, and no corruption happens on the primary register.

- Assertion reported at time 120 indicating assert_rs_mutex failure.

Figure 11-14 SINGLE_PIN_RET_LEVEL



COR_RET

Enables corruption on the shadow register due to mutex failures on level-sensitive save signal.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP
-elements {flop_inst1/outp1}

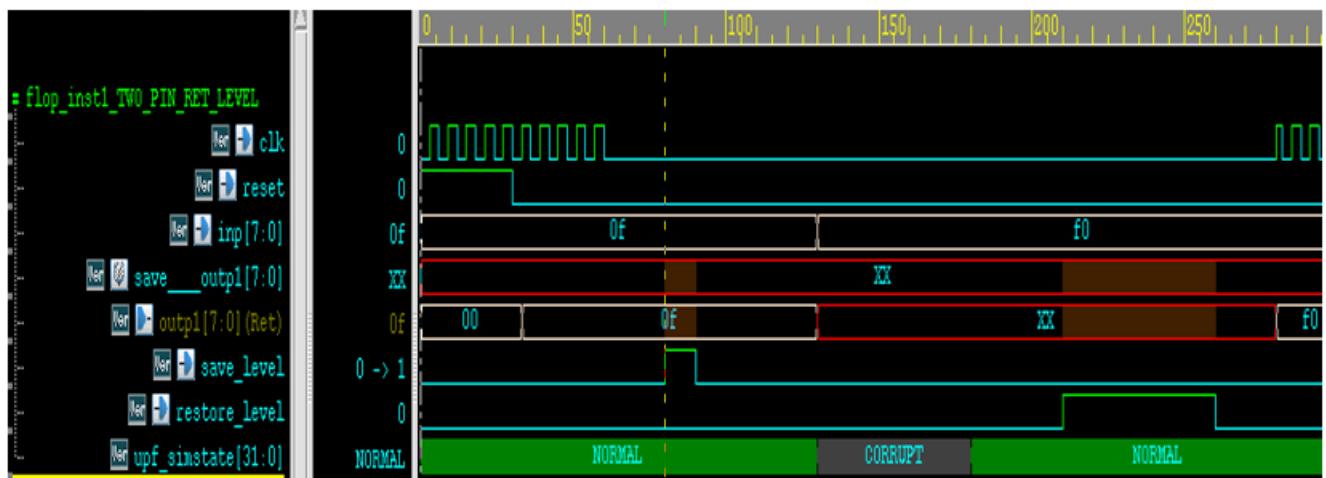
set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
-save_signal {save_level high} -restore_signal
{restore_level high} -assert_rs_mutex {clk low} -parameters
COR_RET
```

Example: outp1 is the primary register, save_outp1 is the shadow register

TWO_PIN_RET_LEVEL

- assert_s_mutex failure at time 80, shadow register is corrupted.
- Assertion reported at time 80 indicating mutex failure.

Figure 11-15 TWO_PIN_RET_LEVEL



COR_NORMAL

Enables corruption on primary register due to mutex failures on level-sensitive restore signal.

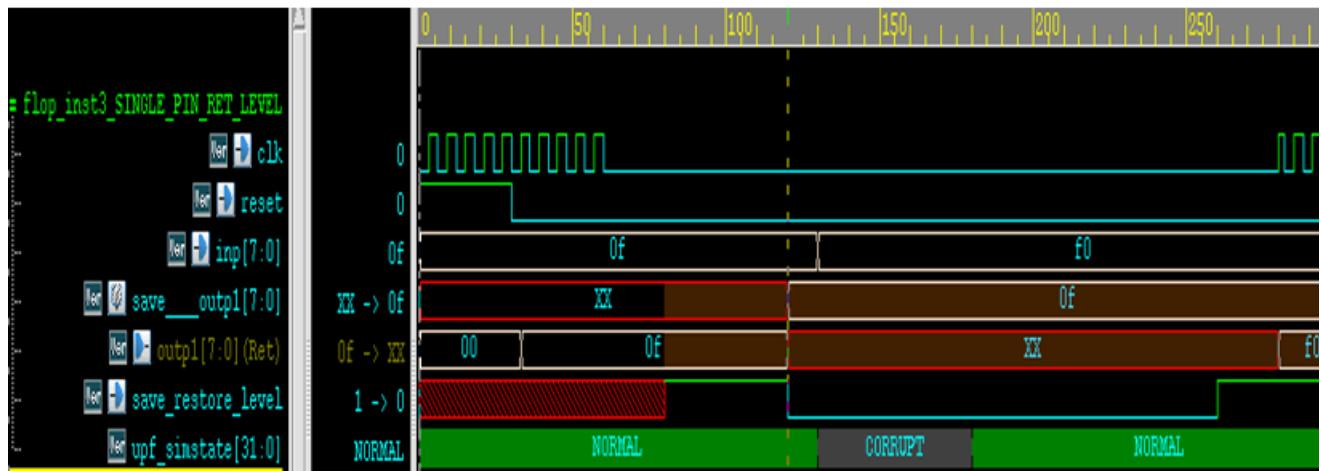
```
set_retention SINGLE_PIN_RET_LEVEL -domain Island_Vdd  
-retention_power net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst3/outp1}  
  
set_retention_control SINGLE_PIN_RET_LEVEL -domain  
Island_Vdd -save_signal {save_restore_level high}  
-restore_signal {save_restore_level low} -assert_rs_mutex  
{clk low} -parameters COR_NORMAL
```

Example: outp1 is the primary register, save_outp1 is the shadow register

SINGLE_PIN_RET_LEVEL

- assert_s_mutex failure at time 120 on trailing edge of save_restore_level. Value is saved to the shadow register, and no corruption happens on the shadow register.
- Assertion reported at time 120 indicating assert_s_mutex failure.
- assert_r_mutex failure from time 120-260, primary register corrupted.
- Assertion reported at time 120 indicating assert_r_mutex failure.

Figure 11-16 SINGLE_PIN_RET_LEVEL



COR_BOTH

Enables corruption on both primary and shadow registers due to mutex failures on level-sensitive restore and save signals.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd  
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst1/outp1}  
  
set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd
```

```

-save_signal {save_level high} -restore_signal
{restore_level high} -assert_s_mutex {clk low} -parameters
COR_BOTH

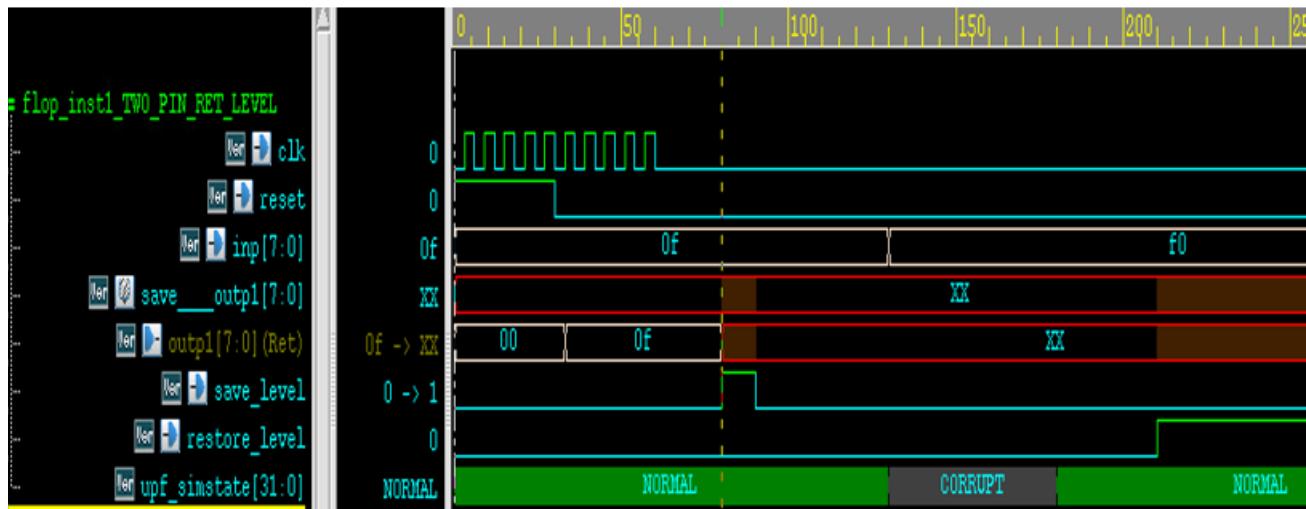
```

Example: `outp1` is the primary register, `save_outp1` is the shadow register

TWO_PIN_RET_LEVEL

- `assert_s_mutex` failure at time 80, shadow register and primary register corrupted.
- Assertion reported at time 80 indicating `assert_s_mutex` failure.

Figure 11-17 TWO_PIN_RET_LEVEL



Supported Parameters with `set_retention`

SAV_RES_COR/NO_SAV_RES_COR

`SAV_RES_COR` enables corruption of primary register during concurrent assertion of `save`, `save_condition`, `restore`, or `restore_condition`. It is default.

`NO_SAV_RES_COR` disables corruption.

```
set_retention TWO_PIN_RET_LEVEL -domain Island_Vdd  
-retention_power_net VDD_TOP -retention_ground_net VSS_TOP  
-elements {flop_inst1/outp1} -parameters SAV_RES_COR

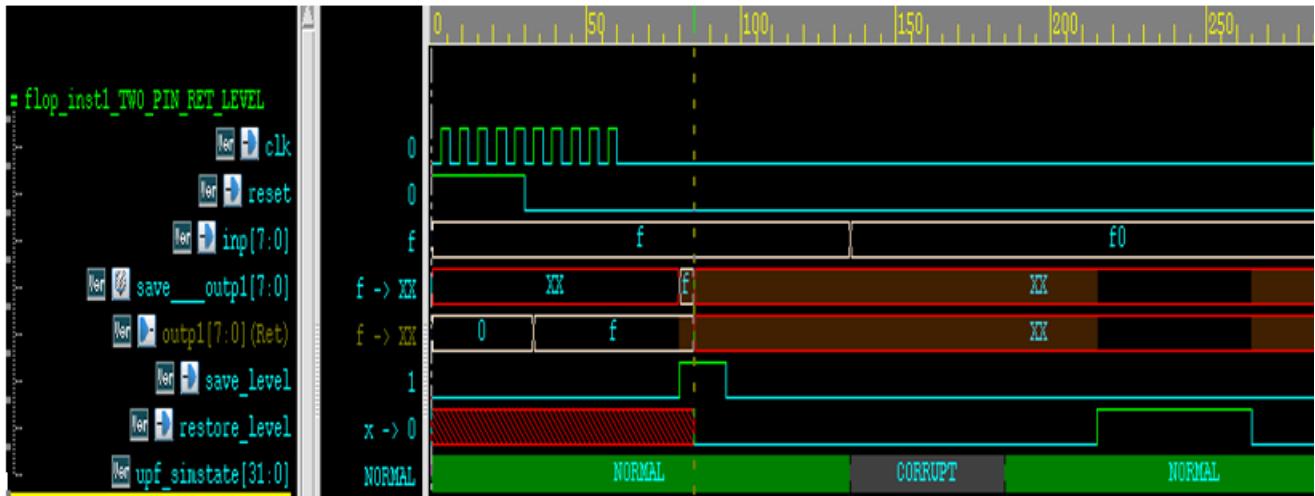
set_retention_control TWO_PIN_RET_LEVEL -domain Island_Vdd  
-save_signal {save_level high} -restore_signal  
{restore_level low}
```

Example: `outp1` is the primary register, `save_outp1` is the shadow register

TWO_PIN_RET_LEVEL

- Both save and restore events at time 85, primary register corrupted.

Figure 11-18 TWO_PIN_RET_LEVEL



RET_SUP_COR/NO_RET_SUP_COR

`RET_SUP_COR` enables corruption of the primary registers when the retention supplies are OFF.

By default, `NO_RET_SUP_COR` disables the primary register when the retention supplies are OFF.

```
set_retention TWO_PIN_RET_LEVEL_P -domain Island_top
-retention_power_net V1SW -retention_ground_net VSS_TOP
-elements {flop_inst8/outp1} -parameters RET_SUP_COR

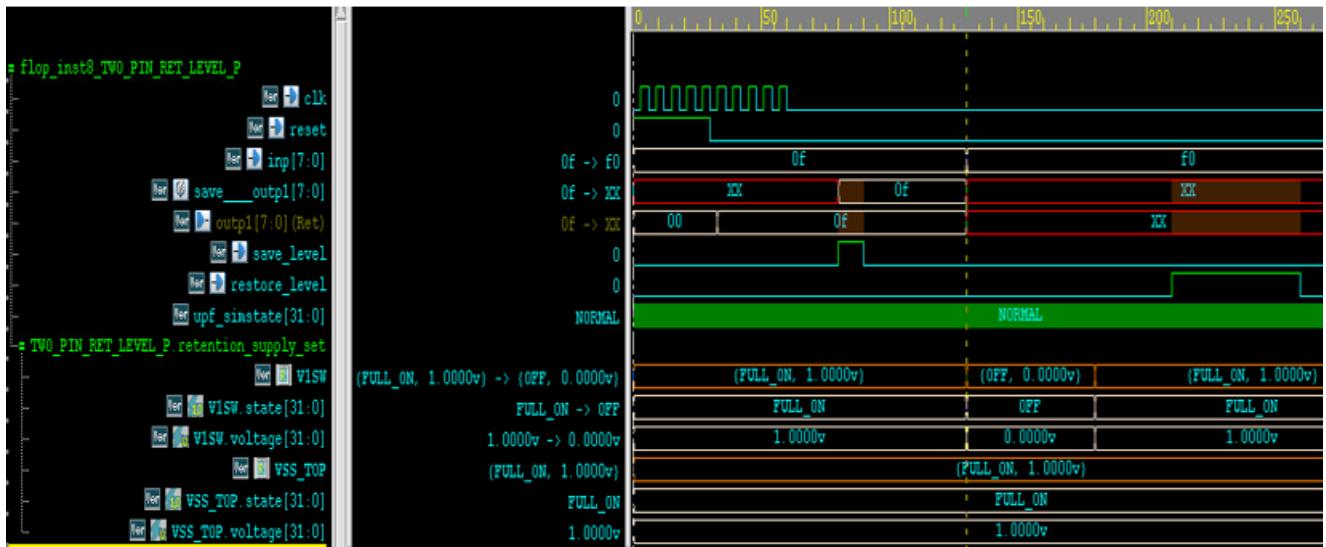
set_retention_control TWO_PIN_RET_LEVEL_P -domain
Island_top -save_signal {save_level high} -restore_signal
{restore_level high}
```

Example: `outp1` is the primary register, `save_outp1` is the shadow register

TWO_PIN_RET_LEVEL_P

- Retention supplies turning OFF at time 130, primary register corrupted even though the primary supplies are ON.

Figure 11-19 TWO_PIN_RET_LEVEL_P



Creating Retention List

The `set_retention_elements` command (see “[set_retention_elements](#)” for syntax) creates a named list of elements to be used in the `set_retention` command.

VCS NLP supports the `-elements` option of the `set_retention_elements` command.

This feature allows you to create a named list of elements to be used in the `set_retention` command. This feature helps you to create the list of registers which need to be retained during power shutdown.

Following is the syntax:

```
set_retention_elements <retention_list_name>
-elements <element_list>
```

Where,

- `retention_list_name` is a simple name; this shall be unique within the current scope.
- `element_list` is a list of rooted names: instances, named processes, sequential registers, or signal names.

Example:

```
set_retention_elements RET_LIST1 -elements {u1}
set_retention RET2 -domain PD2 -elements {RET_LIST1 u2}
```

Key Points to Note

- Retention list should be defined before it is used in the `set_retention` command.
- Wildcards are supported in the `-elements` option.
- You can use the same retention list in multiple `set_retention` commands.
- The behavior of the `set_retention` command matches as if the retention list has been written in the `-elements` option. For example, consider the following command:

```
set_retention_elements rlist1 -elements {u1 u2}
set_retention ret -domain PD1 {rlist1 u3}
```

This command is equivalent to the following:

```
set_retention ret -domain PD1 {u1 u1 u3}
```

Usage Examples

- Using `retention_element_list` along with other elements in `set_retention -elements`.

```
set_retention_elements RET_LIST1 -elements {u1}
set_retention RET2 -domain PD2 -elements {RET_LIST1 u2}
```

- Using elements of `set_retention_elements` individually.

```
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
set_retention RET1 -domain PD1 -elements {R1}
set_retention RET2 -domain PD2 -elements {u1/R2 u2}
```

- Specifying `set_retention_elements` in multiple `set_retention -elements`.

This example also shows that not all the elements in the retention list belong to the same power domain.

```
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
set_retention RET1 -domain PD1 -elements {RET_LIST1}
```

u1/R2, u2 do not belong to power domain PD1.

```
set_retention RET2 -domain PD2 -elements {RET_LIST1}
```

R1 does not belong to the power domain PD2.

VCS NLP issues a warning message in this case. Following is the sample warning message:

```
Warning-[UPF_RET_ELEM_ODS] Retention element outside
domain scope
ret_list_contains_inst_sig_process_dpr_sc1.upf, 41
Ignoring retention on element 'tb/top_inst/u1' specified
with retention strategy 'tb/top_inst/PD1/retention/RET1'
as it is outside the extent of power domain 'tb/top_inst'
```

```
PD1'. Element is in the extent of power domain 'tb/  
top_inst/PD2'.  
Please specify only elements that are in the extent of  
the domain specified with '-domain' option.
```

Invalid Usage Scenarios

VCS NLP issues error message for the following invalid usage scenarios:

- **Multiple retention_element_lists at a given scope with a matching name.**

```
set_scope /  
set_retention_elements RET_LIST1 -elements {u2}  
set_retention_elements RET_LIST1 -elements {u1}
```

Following is the sample error message:

```
Error-[UPF_SNNU] Strategy/Object name not unique  
../Source/mvsim_nlp/nlp_pv/set_retention_elements/  
set_ret_ele_neg_scl/set_ret_ele_neg_scl.upf, 40  
'tb/top_inst/RET_LIST1' is already used as 'retention  
elements list' name in  
UPF/design scope 'tb/top_inst'.  
Strategy/Object names must be unique in one UPF/design  
scope.
```

- **Using retention_element_list in another set_retention_elements command.**

```
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}  
set_retention_elements RET_LIST2 -elements {RET_LIST1}
```

- **Using retention_list_name before definition.**

```
set_retention RET1 -domain PD1 -elements {RET_LIST1 u1}
```

- Referring to `retention_element_list` defined outside the current scope.

```
set_scope /
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
set_scope u1
set_retention RET1 -domain PD1 -elements {RET_LIST1 R1}
```

- Defining multiple retention strategies on an element.

```
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
set_retention RET1 -domain PD1 -elements {RET_LIST1}
set_retention RET2 -domain PD1 -elements {R1}
```

- Defining `-no_retention` strategy for an element in `retention_element_list`.

Case-1:

```
set_retention RET1 -domain PD1 -elements {R1} -
no_retention
set_retention RET2 -domain PD2
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
```

Case-2:

```
set_retention_elements RET_LIST1 -elements {R1 u1/R2 u2}
set_retention RET1 -domain PD1 -elements {R1} -
no_retention
```

Limitations

VCS NLP does not issue an error message in the following cases:

- Element that belongs to `retention_element_list` is not retained when any other element in the same `retention_element_list` is retained.

- Element that belongs to `retention_element_list` is not retained when any other element in the same power domain extent is retained.

Ignoring Latches From Retention

By default, VCS NLP considers flops and latches for retention. You can use the `-power=ignore_latch_retention` compile-time option to ignore latches from retention. This option allows granular control on type of logic (flop, latch) that needs to be retained.

Automated Assertions Supported for Retention

For the list of automated assertions supported for retention, refer to the Retention context in [Table 21-1](#).

Generic Handles

VCS NLP supports `UPF_GENERIC_CLOCK` and `UPF_GENERIC_ASYNC_LOAD` generic handles for accessing the clock and the asynchronous load per retention register respectively.

These generic handles can be used to specify the retention functionality by using them in `boolean_expression` for `-save_condition`, `-restore_condition`, or `-retention_condition` of the `set_retention` command.

Along with specifying the retention functionality, generic handles can also be used in the `bind_checker` command to write the clock or asynchronous load related assertions on the retention registers.

For more information, refer to “[Generic Handles Support](#)” .

Hierarchical Flop Inference Report

VCS NLP generates the `hierarchical_flop_inference.rpt` compile-time static report for Verilog in the `mvsim_native_reports` directory. This report captures the details of the inferred retention registers. This report is dumped by default. The data in the report is dumped as Tcl list as follows:

```
set sr_elements {  
    {retention_register clock reset save_signal save_sense  
     restore_signal restore_sense primary_power_net  
     primary_ground_net retention_power_net  
     retention_ground_net}\\"  
    {retention_register clock reset save_signal save_sense  
     restore_signal restore_sense primary_power_net  
     primary_ground_net retention_power_net  
     retention_ground_net}\\"  
    { .. }\\"  
}
```

Following is the example:

```
set sr_elements { {testbench.top.flop_inst1.outp1 {{7 0}}}  
    testbench.top.flop_inst1.clk  
    testbench.top.flop_inst1.reset testbench/top/save_level  
    high testbench/top/restore_level high testbench/top/V1SW  
    testbench/top/VSS_TOP testbench/top/VDD_TOP testbench/top/  
    VSS_TOP }\\"  
  
    {testbench.top.flop_inst2.outp1 {{7 0}}}  
    testbench.top.flop_inst2.clk  
    testbench.top.flop_inst2.reset testbench/top/save_level
```

```
high testbench/top/restore_level high testbench/top/V1SW  
testbench/top/VSS_TOP testbench/top/VDD_TOP testbench/top/  
VSS_TOP }\  
}
```

Similarly, VCS NLP generates the `hierarchical_flop_inference_vhdl.rpt` compile-time static report for VHDL in the `mvsim_native_reports` directory.

Limitations of Retention Support

Following are the limitations of the state retention support:

- Part-selects are not allowed in -elements of `set_retention`. This is a Synopsys Low Power Flow restriction, and no tool in the flow supports part-selects in -elements.
- No parser warnings for invalid input for `set_retention` -parameters.
- Only clean-edges ($0 \rightarrow 1, 1 \rightarrow 0$) are considered as active edges for save/restore signals.
- Corruption for mutex failures is applied only for level-sensitive cases. No corruption happens for edge-sensitive.

Example

```
typedef struct { reg ret_reg; reg no_ret_reg; } my_str;  
  
my_str q_str;  
reg [2:0] out, q_vec1, q_vec2;  
  
always @ (posedge clk) begin  
    q_vec1      <= d;  
    q_vec2      <= d;
```

```

        q_str.ret_reg <= d[0];
        q_str.no_ret_reg <= d[1];
    end

# Partial Retention -- Supported

set_retention RETAIN_DFF_1 -domain DFF_PD -elements
{q_vec1}

set_retention RETAIN_DFF_2 -domain DFF_PD -elements
{q_vec2}

# Part/Bit-level retention for a vector -- Not supported

set_retention RETAIN_DFF_1 -domain DFF_PD \
-elements {q_vec1[0] q_vec2[2:1] }

set_retention RETAIN_DFF_2 -domain DFF_PD -elements
{q_vec1[1]}

# -no_retention on structure member -- Not supported

set_retention RETAIN_DFF_1 -domain DFF_PD -elements
{q_str}

set_retention NO_RETAIN -domain DFF_PD -no_retention \
-elements {q_str.no_ret_reg}

```

12

Unified Low Power Messaging

This chapter describes the VCS NLP low power messaging in the following sections:

- Message Format
- Message Control
- Wildcard Support
- User-Defined Messages in LP_MSG Format
- Power Aware Verification Environment
- Compile-Time UPF Parser Checks

Message Format

Following is the format of the low power runtime messages that capture the power events during the simulation:

```
[time] [severity] [msg_id] <Message>
```

Example 12-1 Low Power Runtime Message

```
[0] [INFO] [LP_PD_INIT_STATE] Power domain 'testbench/inst/  
Island_VDD' started in 'NORMAL' state.  
[500] [INFO] [LP_PD_STATE_CHANGE] Power domain 'testbench/  
inst/Island_VDD' state changed from 'NORMAL' to 'CORRUPT'.
```

See "[List of Automated Low Power Messages](#)" section for the list of all low power runtime messages.

Message Control

The following two modes of message control are supported:

- [Runtime Configuration File](#)
- [Function Calls from Testbench](#)

The options specified through the runtime configuration file are used for initialization. The function calls from the testbench overrides these initial settings.

The following message control options are supported:

- Choice of message logging modes
- Turn ON/OFF all/selected messages

- Override default severity of messages
 - Filter messages based on severity
-

Runtime Configuration File

Use the `-power <config_file>` runtime option to specify the runtime configuration file. The following topics describe the options supported in the configuration file:

- [Logging Mode](#)
- [Turn ON/OFF Messages](#)
- [Override the Default Severity of Messages](#)
- [Filter Messages Based on Severity](#)

Logging Mode

```
set_lp_msg_log_mode <DEFAULT/CUSTOM/  
DEFAULT_CUSTOM> -file {filename}
```

- **DEFAULT** — All the messages are logged into the simulation log file specified with the `-l` runtime option. Without `-l`, messages are displayed on the console.
- **CUSTOM** — All the messages are logged into the log file specified using `-file` with `set_lp_msg_log_mode`.
- **DEFAULT_CUSTOM** — All the messages are logged into the simulation log file specified with the `-l` runtime option and `-file` with `set_lp_msg_log_mode`.

Note:

-file is optional for CUSTOM and DEFAULT_CUSTOM. If -file is not specified, all messages are logged in the vcs_lpmsg.log file.

Example:

```
set_lp_msg_log_mode CUSTOM -file {test.log}
```

All the low power messages are dumped in the test.log file.

Turn ON/OFF Messages

```
set_lp_msg_onoff <ON/OFF> -msg {msgList}
```

- Default value is ON for all the messages.
- -msg is optional. Not specifying -msg results in the ON/OFF value being applied to all the messages.
- msgList is the list of space separated messages.

Example:

```
Turn OFF all the messages:  
          set_lp_msg_onoff OFF  
Turn ON specific message ID:  
          set_lp_msg_onoff ON -msg { LP_SS_INIT }  
Turn OFF specific message ID:  
          set_lp_msg_onoff OFF -msg { LP_ISOGN_INIT }  
Turn OFF multiple message IDs:  
          set_lp_msg_onoff OFF -msg {LP_SS_INIT  
LP_PST_INIT_ILLEGAL LP_PSW_CTRL_INIT_INVALID}
```

Override the Default Severity of Messages

```
set_lp_msg_severity <INFO/WARNING/ERROR/FATAL> -  
msg { msgList }
```

Severity must be one of INFO/WARNING/ERROR/FATAL.

Example:

```
set_lp_msg_severity ERROR -msg {LP_SS_INIT}
set_lp_msg_severity INFO -msg {LP_ISOEN_INVALID}
set_lp_msg_severity WARNING -msg {LP_ISOGN_OFF }
```

Filter Messages Based on Severity

```
set_lp_msg_log_severity <INFO/WARNING/ERROR/FATAL>
```

All the messages with severity less than the specified severity value are filtered.

- INFO — Default severity level. No messages are filtered.
- WARNING — Messages with severity INFO are filtered.
- ERROR — Messages with severity INFO and WARNING are filtered.
- FATAL — Messages with severity FATAL are ON and the rest are filtered.

Example:

To turn OFF all the INFO messages:

```
set_lp_msg_log_severity WARNING
```

Function Calls from Testbench

You must import the SNPS_LP_MSG package before using the message control functions.

```
import SNPS_LP_MSG::*;


```

Override the Default Severity of Messages

```
set_lp_msg_severity(input string msgIds, input  
string sev);
```

- `msgids` - string type, space separated message IDs
- `sev` - string type, one of INFO/WARNING/ERROR/FATAL

Example:

```
set_lp_msg_severity(" LP_ISOGN_INIT", "WARNING");
```

Turn ON/OFF Messages

```
set_lp_msg_onoff(input string msgIds, input string  
value);
```

- `msgids` - string type, space separated message IDs
- `value` - string type, must be either ON or OFF

Example:

```
set_lp_msg_onoff( "LP_SS_INIT", "OFF");
```

Filter Messages Based on Severity

```
set_lp_msg_log_severity(input string sev);
```

`sev` - string type, must be one of INFO/WARNING/ERROR/FATAL

Example:

```
set_lp_msg_log_severity("WARNING");
```

Wildcard Support

The * and ? wildcard characters are supported with the following control options to list the message IDs:

- set_lp_msg_onoff
- set_lp_msg_severity

Example

Testbench:

```
set_lp_msg_onoff("LP_ISO*", "OFF");  
set_lp_msg_severity("LP_*ILLEGAL", "FATAL");
```

Configuration file:

```
set_lp_msg_onoff OFF -msg {LP_ISO*}  
set_lp_msg_severity FATAL -msg {LP_*ILLEGAL}
```

User-Defined Messages in LP_MSG Format

You can choose to register the custom or user-defined low power messages in the automated assertions format to utilize the benefits of the assertion control options.

The following APIs are supported to register and print the custom messages. You must call these APIs from the testbench and import the SNPS_LP_MSG package.

Once the message is registered and printed using the following APIs, the low power message summary includes these custom messages.

The format of the message is same as automated assertions:

[Time] [Severity] [ID] Message

Register Custom Message

API Prototype:

```
void lp_msg_register(input string msgId, input string  
severity, input string onOff, input string msgText);
```

Example:

```
lp_msg_register("LP_MY_MSG", "WARNING", "OFF", "Assertion  
failure: Primary ground net "%s" for power domain '%s' turned  
OFF.");
```

Printing the Registered Messages

API Prototype:

```
void lp_msg_print(input string msgId, input string text);
```

Example:

```
lp_msg_print("LP_MY_MSG",  
$sformatf(lp_msg_get_format("LP_MY_MSG"), "VSS", "PD1"));
```

Checking if the Registered User-Defined Message is ON/OFF

API Prototype:

```
bit lp_msg_is_enabled(input string msgId);
```

Example:

```
if(!lp_msg_is_enabled("LP_MY_MSG"))
    set_lp_msg_onoff("LP_MY_MSG");
```

Power Aware Verification Environment

Power Aware Verification Environment (PAVE) is an infrastructure that enables accessing the UPF objects, monitor low power events, and write power-aware assertions.

PAVE uses the powerful UPF query commands to query the power intent and UPF bind_checker command to bind the checker modules to the UPF objects like power domains, power switch, isolation strategy, and retention strategy.

You can use the PAVE infrastructure to write custom assertions.

Note:

PAVE is not supported for pure VHDL designs.

Custom Assertions

PAVE enables writing power aware assertions combining the UPF and design variables. For the custom assertions, UPF variables can be populated with the help of UPF query commands. The `bind_checker` command helps bind these custom assertions to the appropriate target.

Custom checker module is written in SystemVerilog and passed as a regular design file for compilation. The UPF query commands and `bind_checker` are written in a custom bind file (Tcl syntax), and passed to VCS with the `-lpa_bind` compile-time option. VCS considers the last occurrence if you specify multiple `-lpa_bind` options on the same command line.

Note:

The query commands are supported only in the Tcl file passed with `-lpa_bind`. VCS NLP generates an error message, if you use query commands in the UPF file.

Use Model

The following steps describe the use model for custom assertions in detail:

Two-step Flow

Compile the user-written custom SV checker modules along with the `-sverilog` option.

Pass the custom bind file using the

`-lpa_bind <custom_bind.tcl>` compile-time option.

Where, `custom_bind.tcl` is the user-written custom bind file.

Note:

- The `query_*` and `bind_checker` commands must be used only in the custom bind file and not in the UPF.
- The `query_*` and `bind_checker` commands do not fully comply with the IEEE 1801 standard.

For information on creating custom bind file, see “[Writing Custom Low Power Assertions](#)” .

Example:

```
% vcs <options> -upf <upf_file> -lpa_bind  
<custom_bind_file> -sverilog
```

Specifying File List with Multiple Bind Checker Tcl Files During Compilation

The `-lpa_bind` option accepts only one Tcl file as input. VCS NLP allows you to specify a file list listing multiple custom bind checker Tcl files using the `-lpa_bind_filelist` compile-time option.

Use Model

Following is the use model of the `-lpa_bind_filelist` compile-time option:

```
% vcs <options> -sverilog -upf <upf_file> file_name.sv  
-lpa_bind_filelist file_list
```

Where,

`file_list`

User-defined text file which contains the path to the Tcl files. The path can be absolute or relative.

Lines starting with # are treated as comments. Blank space or new line is treated as a file separator.

Environment variable usage is supported.

```
# This is a comment  
file1.tcl  
file2.tcl  
file3.tcl file4.tcl  
$PROJ_BIND_FILES/lp/file5.tcl
```

Note:

- Only one -lpa_bind_filelist <file_list> option must be specified at compile/elaboration. VCS considers the last occurrence in case multiple -lpa_bind_filelist options are specified.
- Duplicate Tcl files in the filelist results in the Tcl procedures in the files getting executed multiple times.

Writing Custom Low Power Assertions

Perform the following steps to write custom low power assertions in VCS NLP:

1. Create Custom Bind File

This file is written in Tcl. Do the following in the custom bind file:

- Use UPF query commands to collate the necessary low power data.

Note:

Output from the `query_upf` commands may need post-processing in Tcl to generate an appropriate set of ports and parameters that are bound to the checker module.

- Use `bind_checker` command to bind the low power objects and to pass the appropriate low power data to the checker module.
- Bind the dynamically changing data as ports (for example, save signal, power switch control signal).
- Bind the static constant data as parameters (for example, domain name, supply net name).

2. Create SV Checker Module and Register Custom Assertion

The SV checker module is written in SystemVerilog. You must code the necessary assertion based on the data passed from the custom bind file. The ports monitor the low power events, and parameters provide the static information. See “[User-Defined Messages in LP_MSG Format](#)” section for information to register custom assertion.

3. Compile the Custom Bind File and the Checker Module

Use the `-lpa_bind` compile-time option to pass the custom bind file. Specify the custom checker file as any other regular SV file for compilation.

The following section describes the above three steps in detail with examples.

Example

Scenario: It is assumed that you have implemented power gating through footer cells, and want an assertion to ensure that primary power net of any power domain is never OFF.

1. Create Custom Bind File

Collate the necessary low power data using relevant `query_upf` commands. For the requirement mentioned above, you must have the power domain name and its primary power net. You can use `query_power_domain *` and `query_power_domain <domain> -detailed` to get the required data.

Do the following in the custom bind file (`lpa_custom.tcl`):

- Use the following command to get all the power domains defined:
`query_power_domain *`
- Use the following command to iterate over each power domain and to get the domain name and primary power net:
`query_power_domain <domain> -detailed`
- Bind the dynamically changing object (primary power net) as a port.
- Bind the static constants (domain name and primary power net name) as parameters.

Consider the following example:

```
proc primary_power_off {} {  
    foreach PD [query_power_domain *] {  
        array set pd_detail [query_power_domain $PD
```

```

-detailed]
    set PD_NAME [list PD_NAME $pd_detail(domain_name) ]
        set PPN_NAME [list PPN_NAME
$pd_detail(primary_power_net) ]
            set PPN [list PPN $pd_detail(primary_power_net) ]
                set ports_list {}
                lappend ports_list $PPN
                set params_list {}
                lappend params_list $PD_NAME $PPN_NAME
                bind_checker primary_power_off_msg \
                    -module primary_power_off \
                    -elements [list @$PD ] \
                    -parameters $params_list \
                    -ports $ports_list
                array unset pd_detail
            }
}
primary_power_off

```

In the above example, checker is binded to a power domain (extension of bind_checker), therefore you must use UPF name alias @ with -elements.

Note:

- The names of power domain and the primary power net are constants. Therefore, PD_NAME and PPN_NAME are passed as parameters while binding to the checker module.
- Since the PPN (primary power net) dynamically varies during simulation, it must be bound as a port.

2. Creating SV Checker Module and Registering Custom Assertion

Following is the example code to register custom assertion:

testbench.sv

```
module testbench;
    import UPF::*;
    import SNPS_LP_MSG::*;

    .....
    initial
        lp_msg_register("LP_PPN_OFF", "ERROR", "ON",
    "Assertion failure: Primary power net '%s' for power domain
    '%s' turned OFF.");
    .....
endmodule
```

The checker module defined below checks the state of primary power net of each power domain.

custom_checker.sv

```
module primary_power_off(PPN);
    input PPN;
    import UPF::*;
    import SNPS_LP_MSG::*;

    supply_net_type PPN;
    string PPN_STATE = "";
    parameter string PD_NAME = "";
    parameter string PPN_NAME = "";

    always @ (PPN.state)
        begin : my_lp_checker
            PPN_STATE = PPN.state.name;
            if (lp_msg_is_enabled("LP_PPN_OFF"))
                my_assert : assert (PPN_STATE != "OFF") else
                    lp_msg_print("LP_PPN_OFF",
```

```

$formatf(lp_msg_get_format("LP_PPN_OFF"), "PPN_NAME",
"PD_NAME");
end : my_lp_checker
endmodule

```

3. Compile the Custom Bind File and the Checker Module

Use the following command to compile the custom bind file `lpa_custom.tcl` and the checker module `custom_checker.sv`, along with other design and testbench files.

```
% vcs <options> <design_files> -upf <upf_file>
custom_checker.sv testbench.sv -lpa_bind
lpa_custom.tcl -sverilog
```

Following is the simulation output when primary power net of `PD_TOP` power domain goes to OFF state:

```
"custom_checker.sv", 18:
testbench.top.PD_TOP.primary_power_off_msg.my_lp_checker.check.my_assert: started at 5s failed at 5s

Offending '(PPN_STATE !== "OFF")'

[5] [ERROR] [LP_PPN_OFF] Primary power net 'testbench/top/VDD' of power domain 'testbench/top/PD_TOP' turned OFF.
```

Compile-Time UPF Parser Checks

The following sections describe the compile-time UPF parser checks:

- [Lint Check for Explicit Primary Supplies for a Domain](#)

- Error Checks for Non-Existing Design Objects Referred in UPF
- Lint Check for Mismatch Between Verilog and Liberty Definitions of a Cell

Lint Check for Explicit Primary Supplies for a Domain

The `set_domain_supply_net` and `create_power_domain -supply` commands specify the primary power and ground nets of a power domain explicitly.

As the designs get more and more complex with larger number of power domains, it is useful to identify power domains that do not have explicit specification of primary power or ground nets in the UPF.

You can use the `-power=upf_lint` compile-time option to generate a warning message for every power domain without explicit specification of power or ground nets in the UPF.

Following is the sample warning message:

```
Warning-[UPF_PDWS] Power Domain without supply
top.upf, 7
Power Domain 'testbench/TOP/PD_top' does not have any
'primary' supply specified.
Use 'create_power_domain -supply {primary <sset>}' or
'set_domain_supply_net' to add a supply to the Power Domain.
```

Error Checks for Non-Existing Design Objects Referred in UPF

VCS NLP generates the UPF_ONF and UPF_SPNF error messages at compile time when the design objects referred in the UPF are not found. Multiple scenarios are listed below with corresponding messages:

- **Case-1:** Error message for non-existing design object referred in the UPF

For example, consider the following UPF syntax:

```
create_power_domain Island_V1 -elements
{INST_COUNT_VH1 INST_COUNT_VL}
```

VCS NLP generates the following error message, if INST_COUNT_VH1 does not exist in the design:

```
Error-[UPF_ONF] Object not found
top.upf, 7
'testbench/top_u/INST_COUNT_VH1' specified with '-elements' option of
'create_power_domain' command could not be resolved to
any valid design element.
No instance found with name 'INST_COUNT_VH1' in scope
'testbench/top_u'
(Entity: 'TOP', top.vhd, 17).
Please specify a valid design object.
```

- **Case-2:** Error message for non-existing object specified with connect_supply_net

For example, consider the following UPF syntax:

```
connect_supply_net VDD1 -ports {VDD1 CORE2/VDD1  
CORE3/CORE22/VDD11}
```

VCS NLP generates the following error message, if CORE3 does not exist in the design:

```
Error-[UPF_ONF] Object not found  
top.upf, 39  
'tb/TOP/CORE3/CORE22/VDD11' specified with '-ports'  
option of 'connect_supply_net' command could not be  
resolved to any valid design element.  
No element found with name 'CORE3' in scope 'tb/TOP'  
(Module: 'top', top.v, 1).  
Unresolved Path: 'CORE3/CORE22/VDD11'  
Please specify a valid design object.
```

- **Case-3:** Error message for non-existing supply port

For example, consider the following UPF syntax:

```
connect_supply_net VDD1 -ports {VDD1 CORE2/VDD1  
CORE2/VCC11}
```

VCS NLP generates the following error message, if VCC11 does not exist in the design:

```
Error-[UPF_SPNF] Supply Port not found  
top.upf, 39  
'tb/TOP/CORE2/VCC11' specified with '-ports' option of  
'connect_supply_net'  
command could not be resolved to any valid design element.  
No Supply Port found with name 'VCC11' in scope 'tb/TOP/  
CORE2' (Module: 'core2', module2.v, 1).  
Please specify a valid design object.
```

Lint Check for Mismatch Between Verilog and Liberty Definitions of a Cell

You can use the `+lint=MV_DB_CELL_MISMATCH` compile-time option to generate a warning message when there is a mismatch between the Verilog definition and liberty definition of a cell.

Following is the sample warning message:

```
Lint-[MV_DB_CELL_MISMATCH] Cell and verilog module mismatch
Library cell for verilog module 'foomem' in library
'MACRO1_LIB' does not match.
Reason: Unmatched Port in the Model : temp.
```

You can use the `-error=MV_DB_CELL_MISMATCH` compile-time option to convert the `MV_DB_CELL_MISMATCH` warning message to error message.

Following is the sample error message:

```
Error-[MV_DB_CELL_MISMATCH] Cell and verilog module mismatch
Library cell for verilog module 'foomem' in library
'MACRO1_LIB' does not match.
Reason: Unmatched Port in the Model : temp.
```


13

Debugging Low Power Designs

You can debug low power designs using Verdi Automated Debug Platform. See the following guides for information on using Power-Aware Debug functionality in the Verdi Automated Debug Platform:

- *Verdi Power-Aware Debug User Guide*
- *Verdi and Siloti Command Reference Guide*

14

Coverage Support for Low Power Objects

The following sections describe coverage support for low power objects:

- [Creating Covergroups for the Low Power Objects](#)
- [Enabling Coverage on Low Power Objects](#)
- [Runtime Control for Low Power Coverage](#)
- [Excluding UPF Objects from Low Power Coverage](#)
- [Constraining Low Power Coverage Using the `describe_state_transition` Command](#)
- [Post-Processing Options for Low Power Functional Coverage](#)
- [Coverage Reports](#)
- [Dumping Coverage Exclusion Files for Low Power Objects](#)

Creating Covergroups for the Low Power Objects

You can use the `-power=coverage` compile-time option to automatically create covergroups for the low power objects based on the power intent. The covergroups are created for the following low power objects:

- PST states
- Power switch states/transitions
- Supply net/port states/transitions (Root Supply)
 - States defined through `add_port_state` of connected port
 - States inferred from `supply_expr` of `add_power_state` for the connected supply set
- Supply set power states/transitions
 - primary
 - `isolation_supply_set`
 - `ret_supply`
- Supply set simstate states/transitions
 - primary (domain simstate)
 - `isolation_supply_set`
 - `ret_supply`
 - `default_isolation`
 - `default_retention`
- Isolation enable signal

- Retention SAVE/RESTORE signals
- Power switch control and ACK ports

Note:

For the PST state coverage, final PST state change is sampled if there are multiple PST state changes in the same simulation time stamp.

Enabling Coverage on Low Power Objects

You can use the compile-time options listed in [Table 14-1](#) to selectively enable coverage on the low power objects in a design. This allows you to enable low power coverage from the `vcs` command line without the need to change the UPF frequently.

Table 14-1 Compile-time Options to Enable Coverage

Option	Description
-power=cov_pd_simstate	Enable the power domain simstate coverage.
-power=cov_port_state	Enables port state coverage.
-power=cov_implicit_supply_set	<p>Enables coverage on the implicit supply sets only (simstate and power state).</p> <p>Simstate coverage is enabled only for a power domain. Power state coverage is enabled only for the supply sets if the power states are defined using <code>add_power_state</code>.</p> <p>Coverage is disabled for implicit supply sets <code>default_isolation</code> and <code>default_retention</code>.</p>
-power=cov_explicit_supply_set	<p>Enables power state coverage for the explicit supply sets.</p> <p>Power state coverage is enabled only for the supply sets if the power states are defined using <code>add_power_state</code>.</p> <p>Simstate coverage is disabled for the explicit supply sets.</p>
-power=cov_supply_set	Enables all types of supply set coverage.
-power=cov_psw	Enables power switch coverage on the following low power objects: <ul style="list-style-type: none"> • Power switch state (on/off/partial_on) • ACK port • Control port

Option	Description
-power=cov_pst_state	Enables only the state coverage of PST.
-power=cov_pst_transition	Enables only transition coverage of PST.
-power=cov_pst	Enables Power State Table (PST) coverage. This option enables both the state and transition coverage for PST.
-power=cov_ret	Enables coverage on the retention save and restore signals.
-power=cov_iso	Enables coverage on the isolation enable signal.

You can specify more than one option on the command line using a plus (+) as a delimiter. For example, as shown below:

```
% vcs -power=cov_psw+cov_implicit_supply_set -upf
<upf_file> -sverilog file_name.v
```

The above command enables coverage on the power switch and implicit supply sets.

Note:

The deselect_coverage_object design attribute is given higher priority than the command line option. Therefore, if a specific type of low power coverage is enabled using a command line option, and if deselect_coverage_object is specified in the UPF file on the same low power object, then VCS NLP do not report coverage on that low power object. For example, consider the following command:

```
% vcs -power=cov_supply_set -upf <upf_file>
-sverilog file_name.v
```

If the UPF file contains the following:

```
set_design_attributes -attribute  
deselect_coverage_object default_supply_set
```

then VCS NLP reports coverage only on the explicit supply sets.

Runtime Control for Low Power Coverage

The following options allow you to disable low power coverage at runtime:

Table 14-2 Runtime Options to Disable Low Power Coverage

Option	Description
-covg_disable_cg[=ALL]	Default behavior. Same as using <code>-covg_disable_cg</code> . This option disables all covergroups (low power and user-defined).
<code>-covg_disable_cg=LP</code>	Disables only low power covergroups.
<code>-covg_disable_cg=USER</code>	Disables only user-defined covergroups.

Excluding UPF Objects from Low Power Coverage

You can use the `deselect_coverage_object` design attribute to specify the low power objects that you want to exclude from the coverage collection.

Use Model

Use the following command in the UPF file to disable the coverage collections for various UPF objects:

```
set_design_attributes -attribute  
deselect_coverage_object <Attribute value>
```

[Table 14-3](#) lists the supported attribute values:

Table 14-3 Supported Attribute Values

Attribute Value	Description
power_switch	Turn off power switch coverage
power_state_table	Turn off both PST and port state coverage
supply_port_state	Turn off port state coverage
supply_set	Turn off default and explicit supply set coverage for power state and simstate
default_supply_set	Turn off default supply set coverage for power state and simstate
supply_set_power_state	Turn off coverage for both default and explicit supply set power state
default_supply_set_power_st ate	Turn off default supply set power state
supply_set_simstate	Turn off coverage for both default and explicit supply set simstate
default_supply_set_simstate	Turn off coverage for default supply set simstate
save_signal	Turn off coverage for save signal
restore_signal	Turn off coverage for restore signal
isolation_enable	Turn off coverage for isolation enable
control_port	Turn off coverage for control port switch
ack_port	Turn off coverage for ack_port switch

Example

The following example code disables the coverage of the default supply set, but not the explicit supply set:

```
set_design_attributes -attribute deselect_coverage_object  
default_supply_set
```

Following is the syntax to deselect single coverage object:

```
set_design_attributes -attribute deselect_coverage_object  
{value}
```

Example:

```
set_design_attributes -attribute deselect_coverage_object  
{control_port}
```

Following is the syntax to deselect multiple coverage objects:

```
set_design_attributes -attribute deselect_coverage_object  
{value1} -attribute deselect_coverage_object {value2}
```

Example:

```
set_design_attributes -attribute deselect_coverage_object  
{isolation_enable} -attribute deselect_coverage_object  
{save_signal}
```

Limitations

Following is the limitation of this feature:

- Instance-based disabling of the low power coverage objects is not supported.

Constraining Low Power Coverage Using the `describe_state_transition` Command

You can use the `describe_state_transition` command to describe the desired state transitions or sequences.

By default, VCS NLP assumes that all state transitions between legal and user-defined states are valid transitions. You can also use the `describe_state_transition` command to constraint the set of valid transitions and to define illegal transitions.

Note:

If `describe_state_transition` is specified on an object, the transitions are not generated automatically.

Following is the syntax of the `describe_state_transition` command:

```
describe_state_transition transition_name
-object object_name
[-from from_list -to to_list]
[-paired {{from_state to_state}*} ]
[-through through_list]
[-legal | -illegal]
```

Options

`transition_name`

Simple name of transition.

`Object`

Simple name of any UPF object which contains the power state.
VCS NLP supports the following objects:

- PowerStateTable
- PowerSwitch — on/on_partial/error/off states are considered.
- SupplySet — For supply sets, VCS NLP allows transitions to be captured between named power states defined using add_power_state and simstates. A transition can be from one named power state to another named power state or from one simstate to another simstate.
- SupplyNet — These are the states defined using the add_port_state command of a connected port or inferred from supply_expr of the add_power_state command issued for the supply set to which this supply net is connected.
- SupplyPort

`-from, -to`

These options take a list of states.

`from_list` is an unordered list of power state names active before a state transition.

`to_list` is an unordered list of power state names active after a state transition.

The final transitions are cross of the states in `from_list` and `to_list`.

If `from_list` is {} and `to_list` has valid states, then the command is expanded to all possible transitions to the states in `to_list`.

Similarly, if `from_list` contains valid states and `to_list` is {}, then the command is expanded to all possible transitions from the states in `from_list`.

-through

List of intermediate states. This option is applicable only when the `-from` and `-to` options are specified. If the `-from` and `-to` options are used, then `-through` can be used to define intermediate states between `-from` and `-to` states to create a sequence. This is a VCS NLP specific option to capture a state sequence.

-paired

List of `from-state name` and `to-state name` pairs.

-illegal | -legal

These options specify the legality of the transition being defined as either legal or illegal. The default option is `-legal`. Illegal transitions are marked in illegal bins.

Key points to note

- Transitions/Sequences can be generated using `-from`, `-to`, `-through`, or `-paired`
- For `-from/-to`, transitions are generated from each `from_state` to `to_state`
- `-from {} -to {to_states}` or `-from {from_states} -to {}` is supported
- Use `-paired` to specify pair of states for transition
- Use `-through` to specify sequence (from -> through -> to)

- A transition involving `ILLEGAL` cannot be tagged legal using –
`legal`
- Wildcards are allowed wherever the state name is referred
- All objects with legal/illegal states/transitions being covered are captured in the `mvsim_native_reports/upf_state_machine.rpt` file. You can use this file to verify the output of the `describe_state_transition` command after compile
- The functional coverage on the control ports/signals of a power switch, isolation strategy, and retention strategy is captured in the `upf_control_port_coverage.rpt` file generated in the `mvsim_native_reports` directory. This file contains the details on all the cover group instances for low power functional coverage of the control ports. For more information, see “[Low Power Functional Coverage Report Example](#)” section.

Limitations

- VCS NLP issues error message when the object is a `PowerStateGroup` or `PowerDomain` with no simstates in the options.
- Wildcards are not supported.

Examples

Consider MYPST has states `s1, s2, s3, s4, s5`

```
describe_state_transition pst_tran1 -object MYPST
-from {s1 s2} -to {s4 s5}
```

Expanded Transitions: `{{s1, s4} {s1 s5} {s2 s4} {s2 s5}}`

```
describe_state_transition pst_tran2 -object MYPST
```

```
-from {} -to {s4}
```

Expanded Transitions: {{s1, s4} {s2 s4} {s3 s4} {s5 s4}}

Error Conditions

VCS NLP issues an error message for the following:

- `object_name` does not exist or is not of specified type
- If no `-paired` or `(-from, -to)` is specified
- State referred in the options `-from`, `-to`, and `-paired` is not defined on the object
- If `-through` is specified and `-from`, `-to` is not specified
- If `-legal` is specified for a transition where final state is illegal
- If the same state is specified as both the `-from` state and the `-to` state

Example of `describe_state_transition`

Consider the following PST description:

Supplies:

```
{VDD  VDD_LOW  VDDS_p0  VDDS_p1  VDD_LOWS_p2  
VDD_LOWS_p3  VDDS_misc  VSS}
```

PST States:

```
INIT, P0_BOOT, P0_P1_BOOT, P0_P1_P3_BOOT, ALL_ON,  
HIGH_PERF, LOW_PERF, P3_DROWSY, P2_DROWSY,  
P1_DROWSY, P0_DROWSY, ULTRA_DROWSY
```

PST states hit during simulation:

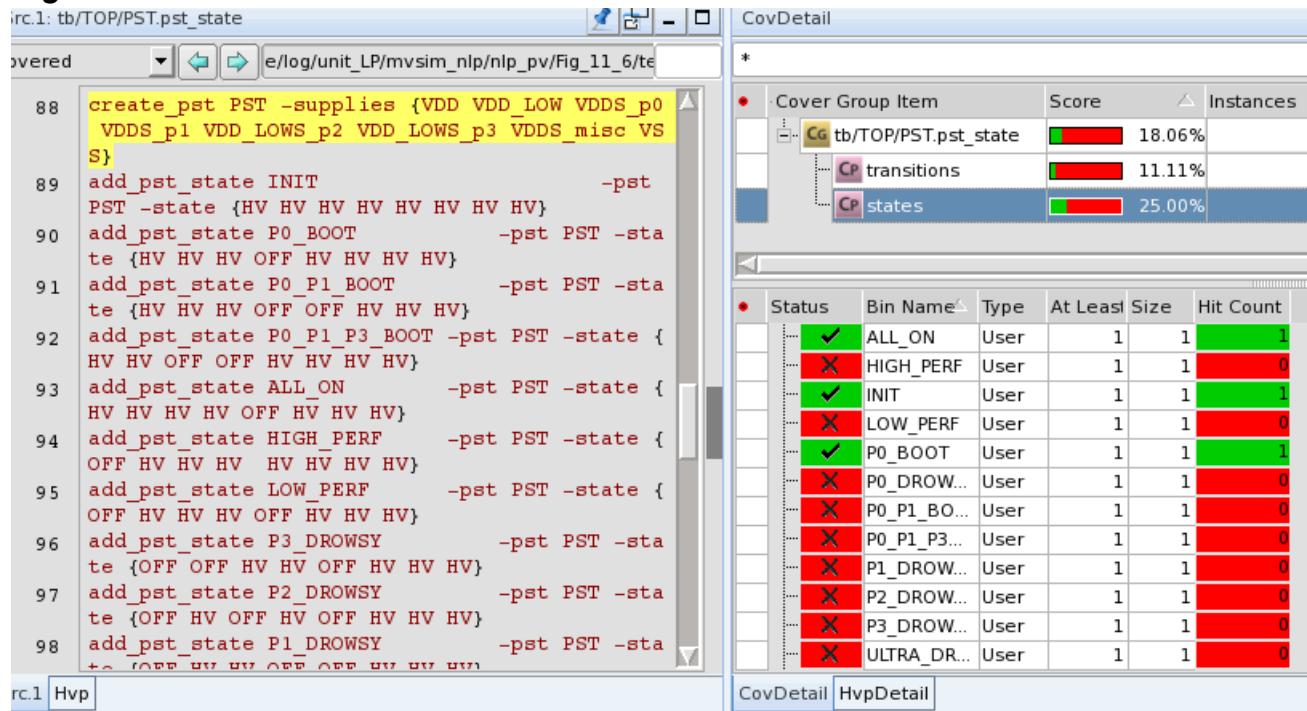
INIT, ALL_ON, P0_BOOT

PST state transitions hit during simulation:

INIT->P0_BOOT

Figure 14-1 shows PST states for the above PST description.

Figure 14-1 PST States



Marking a Transition Illegal

You can use `describe_state_transition` to mark a legal state transition as illegal. For example, you can use the following commands to mark the transition INIT->P0_BOOT (transition highlighted in Figure 14-2) as illegal.

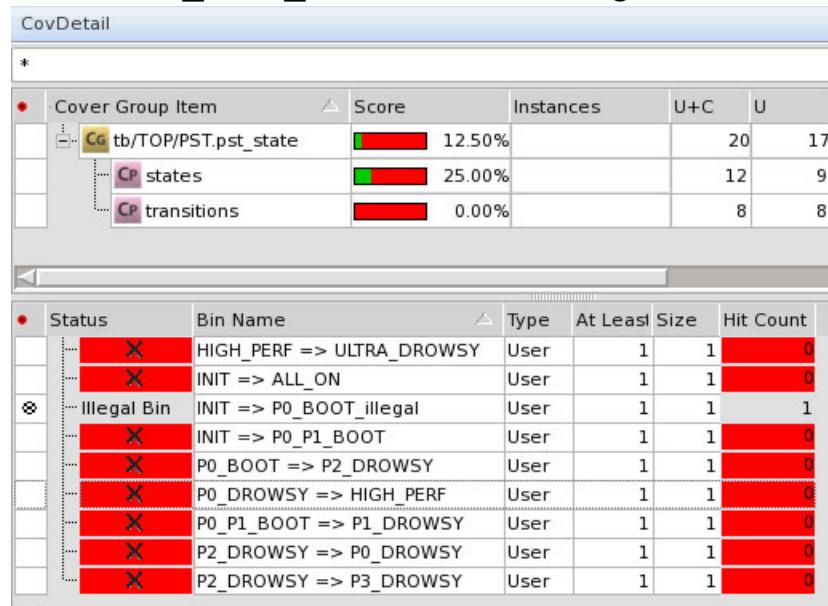
```
describe_state_transition pst_trans_ill \
```

```
-object LEON3_MP_PST \
-paired {{INIT P0_BOOT}} \
-illegal
```

or

```
describe_state_transition pst_trans_ill \
-object LEON3_MP_PST \
-from {INIT} \
-to {P0_BOOT} \
-illegal
```

Figure 14-2 describe_state_transition – Marking a Transition Illegal



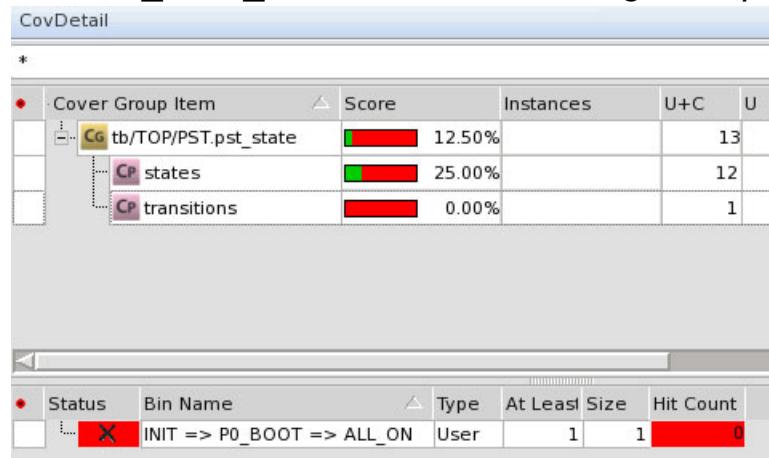
Generating a Sequence

You can use `describe_state_transition` to cover a sequence.

For example, you can use the following commands to cover the `INIT->P0_BOOT->ALL_ON` sequence, as shown in [Figure 14-3](#).

```
describe_state_transition pst_seq \
-object LEON3_MP_PST \
-from {INIT} -to {ALL_ON} -through {P0_BOOT} \
-legal
```

Figure 14-3 describe_state_transition – Generating a Sequence



Post-Processing Options for Low Power Functional Coverage

VCS NLP low power coverage feature has resulted in a new classification for the covergroups as follows:

- Covergroups that represent testbench coverage
- Covergroups instrumented for low power coverage

The testbench coverage groups (user groups) are user written while the low power groups (LP groups) are tool generated. Currently, VCS saves both user groups and LP groups to a single coverage test database during simulation.

None of the post-processing tools recognize categorization, and displays all the covergroups as a single unit under the testbench coverage metric, with the metric score computed from all of the covergroups.

The following sections describe the filtering mechanism for processing covergroups in the post-processing tools. Using the proposed options, coverage reports can be customized for low power coverage or testbench coverage.

Filtering Covergroups in URG

You can use the following URG options to filter covergroups:

- `-group show_lpgroups_only`
- `-group show_usergroups_only`

You can use these options to restrict the set of covergroups to one of the two categories: LP groups and user groups. By default, URG displays both LP and user groups present in the loaded set of databases.

The `-group show_lpgroups_only` option displays only the LP groups and disables all user groups. Similarly, the `-group show_usergroups_only` option displays only the user groups and disables all LP groups.

The disabled covergroups are removed from the universe, that is, the testbench metric score shown in the URG reports is computed only from the enabled covergroups. Similarly, when grading is applied along with any of the two options, only the enabled set of covergroups are graded and reported.

Use Model

Following are the use model scenarios:

Enable all covergroups:

```
urg -dir <list_of_databases>
```

Enable only low power covergroups:

```
urg -dir <list_of_databases> -group show_lpgroups_only
```

Enable only user written covergroups:

```
urg -dir <list_of_databases> -group show_usergroups_only
```

Complete customization of low power coverage, where only LP groups are considered for coverage and reporting with the rest of the metrics disabled, can be achieved by using the **-metric group** and **-group show_usergroups_only** options in conjunction, as follows:

```
urg -dir <list_of_databases> -metric group -group  
show_lpgroups_only
```

If URG encounters both options in the command line, an error indicating the incompatibility is issued. Following is the sample error message:

```
Error-[URG-ICF] Incompatible options  
The '-group show_lpgroups_only' option is not compatible  
with the '-group show_usergroups_only' option.  
You may use only one of them. Please refer to the URG user  
guide for information.
```

If the loaded database contains covergroup data, but has no group matching the specified filter option, a warning message such as the following is issued:

```
Warning-[URG-GF-NCGF] No covergroup found for the filter  
None of the specified coverage directories contain  
covergroups matching the filter 'show_lpgroups_only'.
```

If this is not expected, please check if the correct set of coverage directories are specified or change the filter option.

Filtering Covergroups in UCAPI

To support covergroup filtering in UCAPI, the existing API `covdb_configure` is enhanced by adding a new configuration item, `covdbGroupFilter`, to the enum `covdbConfigItemT` located in the UCAPI header file `covdb_user.h`.

This configuration accepts three values: `show_lpgroups_only`, `show_usergroups_only`, and `show_allgroups`. By default, all covergroups are enabled, which makes the last of the three values the default configuration; the first two values are already described in the previous section.

You can reconfigure the value any number of times. The configuration is overwritten and the subsequent iteration on the test handle for covergroups reflects the latest value. Following are the semantics of the API for the three configuration values:

```
covdb_configure(covdbGroupFilter, "show_lpgroups_only")
covdb_configure(covdbGroupFilter, "show_usergroups_only")
covdb_configure(covdbGroupFilter, "show_allgroups")
```

Alternatively, covergroups can be filtered using the `covdb_get_integer_annotation` API. This API can be invoked with the newly added key `is_lp_group` on any covergroup definition, variant, or instance handle. The API returns 1 if the handle is an LP covergroup, 0 otherwise.

```
covdb_get_integer_annotation(GrpHandle, "is_lp_group")
```

Coverage Reports

The following sections describe the low power coverage reports:

- [Low Power Coverage Report View in URG](#)
 - [Low Power Functional Coverage Report Example](#)
-

Low Power Coverage Report View in URG

You can use the `-power=dump_hvp` compile-time option along with `-power=coverage` to dump the coverage database. You can then use the `-lpcov` option to generate low power coverage reports in URG.

Use Model

To dump the necessary information into the coverage database, include the `-power=dump_hvp` compile-time option along with `-power=coverage` on the `vcs` command line, as shown below:

```
% vcs -sverilog -upf ./upf/top.upf \
-power_top top dut.v -power=coverage -power=dump_hvp \
-R
```

To view the low power coverage report in URG, include the `-lpcov` option in the `urg` command line, as shown below:

```
% urg -dir simv.vdb -lpcov
```

The revised dashboard is displayed as shown in [Figure 14-4](#):

Figure 14-4 URG Dashboard With Low Power Coverage

The screenshot shows a Mozilla Firefox browser window displaying the 'Unified Coverage Report :: Dashboard'. The URL in the address bar is `file:///remote/us01home17/mozilla/dashbo...`. The dashboard header includes links for 'dashboard', 'hierarchy', 'modlist', 'groups', 'tests', 'assets', and 'UPF LP' (which is circled in red). Below the header, it shows the date as 'Wed Aug 26 23:52:01 2015', the user as '#user1', and the version as '0.20150826.00'. The command line used is 'urg -dir simv.vdb -lpcov' (also circled in red), and the number of tests is 1. A section titled 'Scores for Low Power' (circled in red) contains a table:

NAME	SCORE	PD	ISO	RET	PSW	PST	SUPPLY_STATE
LP	45.21	40.00	43.75	100.00	62.50	0.00	25.00

Below this are sections for 'Total Hierarchical Coverage Summary' (with a 'SCORE' button) and 'Total Groups Coverage Summary' (with a 'SCORE' and 'WEIGHT' button, showing a value of 1). At the bottom right is a 'Done' button.

When the `-lpcov` option is used, URG generates the `urgReport/lphvp.LP.html` file. This file displays the low power objects in the Power Domain :: Power Scope format and in a hierarchical manner, as shown in [Figure 14-5](#). Click an object name to go to the Power Scope Detail page or the Power Domain Detail page. Each node can be expanded or collapsed to view the sub-node in the scope.

Figure 14-5 LP Hierarchy Page

The screenshot shows a Mozilla Firefox browser window with the title "Unified Coverage Report :: LP Hierarchy - Mozilla Firefox". The address bar displays "file:///remote/us01home17/.../dash". The main content area is titled "LP Hierarchy" and contains a table with the following data:

NAME	SCORE	PD	ISO	RET	PSW	PST	SUPPLY_STATE
TOP::/	45.21	40.00	43.75	100.00	62.50	0.00	25.00
GENPP::/	25.00	25.00	25.00				
GPRS::/	100.00	100.00	100.00	100.00			
INST::/	25.00	25.00	25.00				
MULT::/	25.00	25.00	25.00				

The Power Scope Detail page displays the coverage scores for all the objects in the respective power scope, as shown in Figure 14-6.

Figure 14-6 Power Scope Detail Page

The screenshot shows the Power Scope Detail Page with the following sections and data:

- Header:** dashboard | hierarchy | modlist | groups | tests | asserts | UPF LP
- Top Level:** Power Domain Subtree Metric: PD Metric: PSW Metric: PST Metric: SUPPLY_STATE
- Power Domain: TOP Scope : /**

SCORE	PD	ISO	RET	PSW	PST	SUPPLY_STATE
45.21	40.00	43.75	100.00	62.50	0.00	25.00
- Subtree:**

NAME	SCORE	PD	ISO	RET	PSW	PST	SUPPLY_STATE
INST:/	25.00	25.00	25.00				
GENPP:/	25.00	25.00	25.00				
MULT:/	25.00	25.00	25.00				
GPRS:/	100.00	100.00	100.00	100.00			
- Metric PD:**

NAME	SCORE	PD
TOP.simstate	25.00	25.00
- Metric PSW:**

NAME	SCORE	PSW
tb/dut/inst_SW /inst_on_PSW_CTRL	25.00	25.00
tb/dut /inst_sw.switch_state	25.00	25.00
tb/dut/gprs_SW /gprs_on_PSW_CTRL	100.00	100.00
tb/dut /gprs_sw.switch_state	100.00	100.00
tb/dut/mult_SW /mult_on_PSW_CTRL	25.00	25.00
tb/dut /mult_sw.switch_state	100.00	100.00
- Metric PST:**

NAME	SCORE	PST
tb/dut /chiptop_pst.pst_state	0.00	0.00
- Metric SUPPLY_STATE:**

NAME	SCORE	SUPPLY_STATE
tb/dut/inst_SW /out.supply_state	0.00	0.00
tb/dut/gprs_SW /out.supply_state	50.00	50.00
tb/dut/mult_SW /out.supply_state	25.00	25.00
tb/dut /VDDI.supply_state	25.00	25.00
tb/dut /VDDG.supply_state	25.00	25.00

The Power Domain Detail page displays the various metrics, such as PSW, PST, as shown in [Figure 14-7](#).

Figure 14-7 Power Domain Detail Page

The screenshot shows a web-based interface for managing power domains. At the top, there's a navigation bar with links to dashboard, hierarchy, modlist, groups, tests, asserts, and UPF LP. Below the navigation is a breadcrumb trail: Power Domain > Metric: PD > Metric: ISO. The main content area is divided into three sections:

- Power Domain: INST**: A table with columns SCORE, PD, ISO, RET, PSW, PST, and SUPPLY_STATE. The first row has values 25.00, 25.00, 25.00, empty, empty, empty, and empty.
- Metric PD:** A table with columns NAME, SCORE, and PD. It contains one row: INST.simstate with values 25.00 and 25.00.
- Metric ISO:** A table with columns NAME, SCORE, and ISO. It contains four rows:
 - tb/dut/INST/isolation /inst_iso_in /inst_iso_in_ISO_SIG with values 25.00, 25.00, and 25.00.
 - tb/dut/INST/isolation /inst_iso_out /inst_iso_out_ISO_SIG with values 25.00, 25.00, and 25.00.

Low Power Functional Coverage Report Example

Use the following command to dump the functional coverage on the control ports/signals of a power switch, isolation strategy, and retention strategy:

```
% vcs -upf <upf_file> -sverilog file_name.v  
-power=coverage
```

The coverage database is populated once you run the simulation.

Use the following command to generate the coverage reports in the text format:

```
% urg -dir simv.vdb -format text
```

The `grpinfo.txt` file is generated in the `urgReport` directory. This file contains the coverage information.

Examples for Different Scenarios

Following are the examples for different scenarios:

- Functional Coverage Report on Isolation Enable
- Functional Coverage Report on Retention SAVE/RESTORE Signals
- Functional Coverage Report on Power Switch Control Port

Functional Coverage Report on Isolation Enable

The coverage report displays the following:

- Hierarchical name of the isolation enable signal with a suffix `_ISO_SIG`.
- State information like `ACTIVE_LEVEL`, `INACTIVE_LEVEL`, `ACTIVE_X`, `ACTIVE_Z`

Transitions like `INACTIVE_TO_ACTIVE` and
`ACTIVE_TO_INACTIVE`

Following is the sample coverage report of the isolation enable:

```
=====
Group : tb/dut/GPRS/isolation/gprs_iso_in/
gprs_iso_in_ISO_SIG
=====
SCORE  WEIGHT  GOAL
100.00 1      100
-----
```

Summary for Group tb/dut/GPRS/isolation/gprs_iso_in/
gprs_iso_in_ISO_SIG

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	4	0	4	100.00

Variables for Group tb/dut/GPRS/isolation/gprs_iso_in/
gprs_iso_in_ISO_SIG

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT
states	2	0	2	100.00	100	1
transitions	2	0	2	100.00	100	1

Summary for Variable states

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for states

Excluded/Illegal bins

NAME	COUNT
ACTIVE_Z	0 Excluded
ACTIVE_X	0 Excluded

Covered bins

NAME	COUNT AT LEAST
INACTIVE_LEVEL	2 1
ACTIVE_LEVEL	1 1

Summary for Variable transitions

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for transitions

Bins

NAME	COUNT	AT LEAST
INACTIVE_TO_ACTIVE	1	1
ACTIVE_TO_INACTIVE	1	1

Functional Coverage Report on Retention SAVE/RESTORE Signals

The coverage report displays the hierarchical name of the retention SAVE/RESTORE signal with a suffix `__RET_SAVE` and `__RET_RESTORE` respectively.

The coverage report does not display information of the `-save_condition` and `-retention_condition`, but the coverage is computed considering these conditions. Other state information and transition information remains same as the isolation enable.

Following is the sample coverage report of the retention SAVE (`__RET_SAVE`) signal:

Group : tb/dut/inst_ret/inst_save__RET_SAVE

SCORE	WEIGHT	GOAL
25.00	1	100

Summary for Group tb/dut/inst_ret/inst_save__RET_SAVE

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
----------	----------	-----------	---------	---------

Variables	4	3	1	25.00		
Variables for Group tb/dut/inst_ret/inst_save__RET_SAVE						
VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT
states	2	1	1	50.00	100	1
transitions	2	2	0	0.00	100	1

Summary for Variable states

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	1	1	50.00

User Defined Bins for states

Uncovered bins

NAME	COUNT	AT LEAST	NUMBER
ACTIVE_LEVEL_0	1		1

Excluded/Illegal bins

NAME	COUNT
ACTIVE_Z_0	Excluded
ACTIVE_X_0	Excluded

Covered bins

NAME	COUNT	AT LEAST
INACTIVE_LEVEL_1	1	1

Summary for Variable transitions

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	2	0	0.00

User Defined Bins for transitions

Uncovered bins

NAME	COUNT	AT LEAST	NUMBER
INACTIVE_TO_ACTIVE	0	1	1
ACTIVE_TO_INACTIVE	0	1	1

Following is the sample coverage report of the retention RESTORE (__RET_RESTORE) signal:

=====

Group : tb/dut/gprs_ret/gprs_nrestore__RET_RESTORE

=====

SCORE	WEIGHT	GOAL
100.00	1	100

Summary for Group tb/dut/gprs_ret/
gprs_nrestore__RET_RESTORE

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	4	0	4	100.00

Variables for Group tb/dut/gprs_ret/
gprs_nrestore__RET_RESTORE

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT
states	2	0	2	100.00	100	1
transitions	2	0	2	100.00	100	1

Summary for Variable states

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for states

Excluded/Illegal bins

NAME	COUNT
ACTIVE_Z 0	Excluded
ACTIVE_X 0	Excluded

Covered bins

NAME	COUNT	AT LEAST
INACTIVE_LEVEL 2	1	
ACTIVE_LEVEL 1	1	

Summary for Variable transitions

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for transitions

Bins

NAME	COUNT	AT LEAST
INACTIVE_TO_ACTIVE 1	1	
ACTIVE_TO_INACTIVE 1	1	

Functional Coverage Report on Power Switch Control Port

The coverage report displays the following:

- State information like SIGNAL_1, SIGNAL_0, SIGNAL_X, SIGNAL_Z
- Transitions like LOW_TO_HIGH and HIGH_TO_LOW

Following is the sample coverage report of the power switch control port:

```

=====
Group : tb/dut/gprs_sw/gprs_on__PSW_CTRL
=====
SCORE  WEIGHT GOAL
100.00 1      100

-----
Summary for Group    tb/dut/gprs_sw/gprs_on__PSW_CTRL

CATEGORY EXPECTED UNCOVERED COVERED PERCENT
Variables 4          0          4        100.00

Variables for Group  tb/dut/gprs_sw/gprs_on__PSW_CTRL

VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT
states   2          0          2        100.00 100  1
transitions 2         0          2        100.00 100  1

-----
Summary for Variable states

CATEGORY           EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 2          0          2        100.00

User Defined Bins for states

Excluded/Illegal bins

NAME COUNT
SIGNAL_Z 0     Excluded
SIGNAL_X 0     Excluded

Covered bins

NAME COUNT AT LEAST
SIGNAL_1 1     1
SIGNAL_0 2     1

```

Summary for Variable transitions

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for transitions

Bins

NAME	COUNT	AT LEAST
HIGH_TO_LOW	1	1
LOW_TO_HIGH	1	1

Dumping Coverage Exclusion Files for Low Power Objects

You can use the `-dump full_exclusions group URG` option to dump the exclusion files with user-defined covergroups and automated low power covergroups. [Table 14-4](#) describes the exclusion files generated by `-dump full_exclusions group`.

Table 14-4 Exclusion Files Generated by -dump full_exclusions group

Exclusion File	Description
<code>fullexclude.tb_def</code>	Contains user-defined covergroups
<code>fullexclude.tb_inst</code>	Contains instances of the user-defined covergroups
<code>fullexclude_lp.tb_def</code>	Contains automated low power covergroups

For more information on the `-dump full_exclusions` option, see [Coverage Technology User Guide](#).

15

Partition Compile Support

You can apply incremental changes to the design source or IEEE 1801, also known as the Unified Power Format (UPF) files, or you can modify the low power options.

Note:

VCS installation is required for using Partition Compile.

Limitations

The following limitations apply in VCS NLP:

- Partitioning of instances in an instance array is not supported.
- The `bind_checker` command is not supported.
- Level Shifter inference and 30-70 corruption is not supported.
- Generic handles for `set_retention` is not supported.

- Generic handles for isolation/retention in `bind_checker` is not supported.

16

Gate-level Netlist Simulations (Simulating with Liberty (.db) Files)

In VCS NLP, gate-level netlist simulations are not the same as RTL simulations. In addition to the design (gates and models) and the UPF, VCS NLP can also read some special attributes about the cells from the .db representations. The following sections describe gate-level netlist simulations (GLS) with VCS NLP:

- Library Mapping
- Using Connect Supply Net
- DB Matching Criteria
- Supported .db Attributes
- Power Aware and Non-Power Aware Models
- Simulation Behavior for Power Aware Models

- Implicit Supply Net Connections
- UPF2SV/SV2UPF Connections
- Value Conversion Tables (VCTs)
- User-Defined Value Conversion Tables
- Multi-Rail Macro Simulation
- Association of Pre-Inserted Isolation Cells
- Support for Timing Simulation With SDF in Presence of UPF
- Specifying Default ON Voltage for Internal PG Pins of a Macro

Library Mapping

Library mapping is used to determine a matching library cell for a Verilog or VHDL module-based architecture. A module is considered as a cell, if a matching library cell for the module exists in a specified database.

Use Model

Use the following search path and link library command in the `-power_config` file:

```
db_search_path = {path}  
db_link_library = {.db files}
```

Where,

- {path} is a Tcl list where each element of the list is a valid path to the directory where DB files are present.
- { .db files} is a Tcl list where each element of the list is the name of a DB file.

Example:

```
db_search_path = {<path1>/IP1 <path2>/IP2}
```

```
db_link_library = {iso_pg.db ret_pg.db}
```

Specifying Library Path Using Environment Variable in the -power_config File

Following is the use model to specify library path using environment variable in the -power_config file:

```
db_search_path = "libs $env(path)"
```

Where, path is a valid path to the directory where DB file is present.

Example: db_search_path = "libs \$env(MYPATH) /DB"

To view the path VCS NLP is going to use, you can print the path before you assign it to db_search_path. For example:

```
set search_path "libs $env(MYPATH) /DB"  
puts "SEARCH_PATH = $search_path"  
db_search_path = $search_path
```

Using Connect Supply Net

The `connect_supply_net` command is used to connect the UPF supply nets to the HDL supply nets in the Verilog or PG cell models.

Example

The following example shows the usage of the explicit `connect_supply_net` command:

```
create_power_domain Island_Vdd -include_scope
create_power_domain Island_V1 -elements inst1

create_supply_port V1 -domain Island_V1
create_supply_port Vdd -domain Island_Vdd
create_supply_port VSS -domain Island_Vdd

create_supply_net V1 -domain Island_V1

connect_supply_net Vdd -ports { Vdd out_0_UPF_ISO/VDD
out_1_UPF_ISO/VDD out_5_UPF_ISO/VDD out_6_UPF_ISO/VDD
out_7_UPF_ISO/VDD }
connect_supply_net VSS -ports { VSS out_0_UPF_ISO/VSS
out_1_UPF_ISO/VSS out_5_UPF_ISO/VSS out_6_UPF_ISO/VSS
out_7_UPF_ISO/VSS }

create_power_switch V1_header_switch -domain Island_V1
```

In this example UPF, the `connect_supply_net` command has the explicit connection from the UPF supply port `Vdd` to the HDL cell supplies `out_0_UPF_ISO/VDD..out_1_UPF_ISO/VDD`, and so on. VCS NLP propagates the state and voltage values of the UPF supply net to the HDL supplies based on the connection.

DB Matching Criteria

DB matching in VCS NLP is stricter than MVSIM-PLI. The following conditions must be satisfied for the cell definition in the .db to be termed as matched to the RTL model.

- Name of the Verilog model and db cell must match
- All RTL ports must match the cell logic pins in name and width
- Any RTL port matching db PG pin must not have width greater than 1

Simulation Semantics for Cells

- No corruption semantics are applied on internals of the model
- Corruption semantics of pins is based on the liberty attributes read
- A cell is treated as a black-box for LP instrumentation
- Internal PG pins must be driven by the model in Verilog

Cell information is captured in the [`mvsim_native_reports/library_mapping`](#) report.

Matching/non-matching cells are listed in the [`mvsim_native_reports/library_match`](#) report.

Supported .db Attributes

VCS NLP supports the following .db attributes. For more information on these attributes, see the *Library Compiler LRM* (Liberty Format). These attributes help VCS NLP to accurately simulate the power-aware behavior of the netlist and UPF.

- **Logic Pins** — These are the input and output ports of the cells, same as the ports listed in the model of the cell. VCS NLP expects the logic pins of .db and the model to match exactly. This is how VCS NLP maps a cell model with the appropriate .db. If the module names match, but the logic pins do not, VCS NLP ignores the .db representation of the cell.
- **PG Pins** — These are the power and ground pins of the cell. These may be primary, backup, or internal PG pins. They may or may not present in the model.
- **Power Down Function** — This is a Boolean function of PG pins, one for each output pin of the cell.
- **Related PG Pin** — The `related_pg_pin` attribute links a logic pin with a PG pin. In effect, it indicates which PG pin controls the logic pin's shutdown behavior.
- **Cell Type** — Following are the special types of cells that VCS NLP supports:
 - `always_on` cells
 - isolation cells
 - level shifter cells
 - retention cells

- power switch cells
 - Nor-Style Isolation cells and Nor-Style Enabled Level Shifter (ELS) cells (see the following section).
-

Support for Nor-Style Cells

In multi-voltage designs, Enabled Level Shifter (ELS) cells are used to achieve the functionality of both level-shifting and isolation. ELS cells with NOR-style isolation are referred as NOR-style ELS cells.

NOR-style isolation is applicable to the isolation strategies with clamp value 0. For more information on NOR-style isolation, see [“Inserting NOR-Style Isolation Cells as per UPF 3.0 LRM”](#).

The following .db attribute supports both NOR-style isolation cells and NOR-style ELS cells. For more information on this attribute, refer to the *Library Compiler LRM* (Liberty Format).

```
alive_during_partial_power_down : true
```

This attribute can be present on the enable pin and output pin of an ELS cell.

If you use this attribute, VCS NLP does not perform input corruption based on `related_power_pin` and `related_ground_pin` of the enable pin.

Example

```
cell (ELS) {
    pin (EN) {
        related_power_pin : "VDDOUT";
        related_ground_pin : "VSS";
```

```

isolation_cell_enable_pin : true;
alive_during_partial_power_down : true;
direction : "input";
}
pin (Z) {
    related_power_pin : "VDDOUT";
    related_ground_pin : "VSS";
    direction : "output";
alive_during_partial_power_down : true;
power_down_function :"!VDDIN * (!VDDOUT*!EN)+VSS";
function :"!(!A + EN)";
} /* end cell */

```

Power Aware and Non-Power Aware Models

The model for a cell may or may not include PG pins. A model which does not contain PG pins is referred to as a “non-power aware” model. If the PG pins are present in a model, they must exactly match what is present in the .db representation. Otherwise, VCS NLP treats it as a non-power aware model.

In a power aware model, the PG pins can be:

- input/output ports of the model
- reg/wire
- supply_net_type
- supply0/supply1 type

VCS NLP uses standard VCTs, as defined in the LRM, for conversion between various types. The default VCS NLP behavior seeks to match the current MVSIM behavior. However, you can use other different VCTs. For more information on VCTs, see “[Value Conversion Tables \(VCTs\)](#)” section.

Simulation Behavior for Power Aware Models

If a model is recognized as power aware, it means that the model itself completely describes the power behavior. That is, VCS NLP does not corrupt anything inside the model. It only drives the appropriate supply values to the PG pins of the cell. The model takes care of corruption. For example, in the following power aware model, the output pin `Z` is corrupted to `x` when the PG pin `VDD` is turned off.

```
module BUFFD0HVT (I, Z, VDD, VSS);  
  input I, VDD, VSS;  
  output Z;  
  
  assign Z = (VDD) ? I : 1'bx;  
  
endmodule
```

Simulation Behavior for Non-Power Aware Models

If a model is recognized as non-power aware, VCS NLP corrupts the outputs of a cell based on the power-down function. If there is no power-down function, VCS NLP uses the `related_pg_pin` attribute to corrupt the ports.

Implicit Supply Net Connections

The UPF may have `connect_supply_net` (CSN) commands to the PG pins of various cells. If there is no CSN in the UPF, the PG pin is assumed to be connected to the primary power or primary ground of the domain that the cell is in.

The isolation cells (ISO/ELS) and retention cells are exceptions to this rule. If the UPF contains isolation strategies, VCS NLP tries to match isolation strategies with the implemented ISO and ELS cells in the netlist. Once an ISO cell is matched with a strategy, the `isolation_power_net` and the `isolation_ground_net` specified in the strategy are implicitly connected to the PG pins of the ISO cell.

In the Synopsys low power flow, explicit CSN is required for the level shifter cells, else the level shifter cells are treated as always-on cells, and no implicit connections are made to the PG pins.

If there is an explicit CSN, that connectivity is honored. Similarly, if there is a retention strategy, VCS NLP tries to associate the retention cell with a strategy. If there is no explicit CSN, the retention power/ground nets in the strategy implicitly drive the backup PG pins of the retention cell.

For the blocks that are not bias enabled using the `enable_bias` design attribute (see “[Enabling Bias Feature](#)”), unconnected bias PG pins (pwell/nwell) are tied to the domain supplies implicitly. pwell is tied to the primary ground, and nwell is tied to the primary power of the domain. To keep the unconnected bias PG pins ON, use the `-power=unconnected_bias_pins_on` compile-time option.

For FDSOI (Fully Depleted Silicon on insulator), both pwell and nwell are tied to the ground supply. To enable FDSOI, set the `is_soi` design attribute to `TRUE` in UPF, as follows:

```
set_design_attributes -models <model_name> -  
attribute is_soi TRUE
```

UPF2SV/SV2UPF Connections

In presence of the appropriate .db files, the information in .db is considered golden for determining the connection (sv2upf/upf2sv). The pg_type and/or direction attributes decide the drive of the supply connection.

SV2UPF Connection

- In case of DB matched model:
 - pg_type attribute is internal_power/internal_ground or
 - direction attribute is output
- In case of Verilog model with no corresponding .db:
 - For the ports in the same scope, the direction of the Verilog port is input
 - For the ports in the lower scope, the direction of the Verilog port is output
 - Connection to a non-port type in the same scope

UPF2SV Connection

- In case of DB matched model:
 - pg_type in .db is not internal_power/internal_ground
- In case of Verilog model with no corresponding .db:

- For the ports in the same scope, the direction of the Verilog port is output
 - For the ports in the lower scope, the direction of the Verilog port is input
 - Connection to a non-port type in the lower scope
-

Value Conversion Tables (VCTs)

A value conversion table (VCT) defines how values are mapped for the connections from the UPF to the HDL model or from the HDL model to the UPF.

You can use the `-vct` option of the `connect_supply_net` command to specify VCT.

UPF supply net can be connected to:

- `wire/reg/logic/UPF::supply_net_type` in Verilog
- `std_logic/std_ulogic/bit/UPF::supply_net_type` in VHDL
- VCT is not required for connection to `UPF::supply_net_type` port/net in HDL.

Supported VCTs

The following table describes the supported VCTs.

Table 16-1 Supported VCTs

Type	VCT	Table
HDL2UPF	SV_LOGIC2UPF *	1 - FULL_ON 0 - OFF z - UNDETERMINED x - UNDETERMINED
	SV_LOGIC2UPF_GNDZERO **	1 - OFF 0 - FULL_ON z - UNDETERMINED x - UNDETERMINED
	SV_LOGIC2UPF_MD	1 - FULL_ON 0 - OFF z - OFF x - UNDETERMINED
	SV_LOGIC2UPF_GNDZERO_MD	1 - OFF 0 - FULL_ON z - OFF x - UNDETERMINED
	SV_LOGIC2UPF_2S	1 - FULL_ON 0 - OFF z - OFF x - OFF
	SV_LOGIC2UPF_GNDZERO_2S	1 - OFF 0 - FULL_ON z - OFF x - OFF
UPF2HDL	SV_TIED_HI	FULL_ON - 1 OFF - X PARTIAL_ON - X UNDETERMINED - X
	SV_TIED_LO	FULL_ON - 0 OFF - X PARTIAL_ON - X UNDETERMINED - X

Table 16-1 Supported VCTs

	UPF2SV_LOGIC ^	FULL_ON - 1 OFF - $\bar{0}$ PARTIAL_ON - X UNDETERMINED - X
	UPF_GNDZERO2SV_LOGIC ^^	FULL_ON - 0 OFF - $\bar{1}$ PARTIAL_ON - X UNDETERMINED - X
	VHDL_TIED_HI	FULL_ON - 1 OFF - \bar{X} PARTIAL_ON - X UNDETERMINED - X
	VHDL_TIED_LO	FULL_ON - 0 OFF - \bar{X} PARTIAL_ON - 0 UNDETERMINED - X
	UPF2VHDL_SL	FULL_ON - 1 OFF - $\bar{0}$ PARTIAL_ON - X UNDETERMINED - X
	UPF_GNDZERO2VHDL_SL	FULL_ON - 0 OFF - $\bar{1}$ PARTIAL_ON - X UNDETERMINED - X

* **SV_LOGIC2UPF** is the default HDL2UPF VCT used for the power ports.

** **SV_LOGIC2UPF_GNDZERO** is the default HDL2UPF VCT used for the ground ports.

^ **UPF2SV_LOGIC** is the default UPF2HDL VCT used for the power ports.

^^ **UPF_GNDZERO2SV_LOGIC** is the default UPF2HDL VCT used for the ground ports.

Attributes Supported for Overriding the Default VCTs

Table 16-2 Attributes supported for overriding the default VCTs

Attribute	Type
SNPS_default_power_sv2upf_vct	SV2UPF for power ports
SNPS_default_ground_sv2upf_vct	SV2UPF for ground ports
SNPS_default_power_upf2sv_vct	UPF2SV for power ports
SNPS_default_ground_upf2sv_vct	UPF2SV for ground ports
SNPS_default_power_upf2vh_vct	UPF2VH for power ports
SNPS_default_ground_upf2vh_vct	UPF2VH for ground ports

Changing default VCT for the whole design:

```
set_design_attributes -attribute <attribute>
<vct_name>
```

Changing default VCT for a DB Cell:

```
set_design_attributes -models <model_name> -
attribute <attribute> <vct_name>
```

Changing VCT for a particular supply net connection:

```
connect_supply_net <net_name> -ports {<port_list>}
-vct {<vct_name>}
```

Precedence of VCT in High to Low Order

1. VCT in CSN
2. VCT of the cell, if defined
3. VCT of the design, if defined

4. Default VCT

Limitations

Following are the limitations of this feature:

- Defining custom VCTs is not supported
- For HDL2UPF, -vct with connect_supply_net cannot be used for the supply net connections to a DB matched cell port

User-Defined Value Conversion Tables

You can use the following IEEE Std 1801 UPF commands to create user-defined VCTs. You can use these VCTs for value conversion with the connect_supply_net command:

- create_upf2hdl_vct
- create_hdl2upf_vct

The following sections describe these commands in detail:

- [Using the create_upf2hdl_vct Command](#)
- [Using the create_hdl2upf_vct Command](#)
- [Syntax Checks](#)
- [Semantic Checks for the connect_supply_net -vct Command](#)

Using the `create_upf2hdl_vct` Command

The `create_upf2hdl_vct` command defines a VCT that can be used to convert the UPF `supply_net_type.state` values into the HDL logic values when the values are propagated from a UPF supply net to a logic port defined in HDL model.

Syntax

```
create_upf2hdl_vct vct_name
    -hdl_type {<vhdl | sv> [typename]}
    -table {{from_value to_value}*}
```

* Indicates that the item in curly braces can be repeated in the command.

Syntax examples:

```
create_upf2hdl_vct upf2vlog_vdd
    -hdl_type {sv}
    -table {{OFF X} {FULL_ON 1} {PARTIAL_ON 0}}
create_upf2hdl_vct upf2vhdl_vss
    -hdl_type {vhdl std_logic}
    -table {{OFF 'X'} {FULL_ON '1'} {PARTIAL_ON 'H'}}
```

Using the `create_hdl2upf_vct` Command

The `create_hdl2upf_vct` command defines a VCT from an HDL logic type to the state type of the supply net value when that value is propagated from HDL port to a UPF supply net.

Syntax

```
create_hdl2upf_vct vct_name
    -hdl_type {<vhdl | sv> [typename]}
```

```
-table {{from_value to_value}*}
```

Syntax examples:

```
create_hdl2upf_vct vlog2upf_vss
    -hdl_type {sv reg}
    -table {{X OFF} {0 FULL_ON} {1 OFF} {Z PARTIAL_ON}}
create_hdl2upf_vct stdlogic2upf_vss
    -hdl_type {vhdl std_logic}
    -table {{'U' OFF}
            {'X' OFF}
            {'0' OFF}
            {'1' FULL_ON}
            {'Z' PARTIAL_ON}
            {'W' OFF}
            {'L' OFF}
            {'H' FULL_ON}
            {'-' OFF}}
```

Syntax Checks

The following points describe the options of the `create_hdl2upf_vct` and `create_upf2hdl_vct` commands:

- `vct_name` provides a name for the value conversion table for later use with the `connect_supply_net` command. It must be a simple and unique name and can be referenced from anywhere in a power intent description.

Since the name of an user-defined VCT belongs to global namespace, it cannot conflict with any other objects defined in the global scope. Conflicts may arise with other VCT names, model names, and retention-element list.

For more information about these VCTs, see IEEE Std 1801-2013 LRM.

Following are the error scenarios:

Scenario-1:

```
create_hdl2upf_vct VCT1  
create_upf2hdl_vct VCT1
```

Result: Error since VCT1 already exists.

Scenario-2:

```
create_hdl2upf_vct UPF2VHDL_SL
```

Result: Error since UPF2VHDL_SL is the name of a pre-defined VCT.

- `-hdl_type` specifies the HDL type for which the value conversions are defined. If the `typename` is not one of the language's predefined types or one of the types specified in the next paragraph, it must be of the form `library.pkg.type`.

The following HDL types are the minimum set of types supported:

VHDL

- `Bit`, `std_[u]logic`, `Boolean`
- Subtypes of `std_[u]logic`

SystemVerilog

- `reg/wire`, `Bit`, `Logic`

VCS NLP issues an error message if there is a mismatch in the HDL type.

- `-table` defines the 1:1 conversions from UPF supply net states to an HDL logic value or vice versa. The values must be consistent with the HDL type values. For example:
 - When converting from/to SystemVerilog logic type, the legal values are 0, 1, X, and Z.
 - When converting from/to SystemVerilog or VHDL bit, the legal values are 0 or 1.
 - If the states are defined for illegal values, states defined for illegal values are ignored.
 - When converting from/to VHDL std_[u]logic, the legal values are U, X, 0, 1, Z, W, L, H, and -.

VCS NLP issues an error message for the following scenarios:

- If in the VCT table, either value is missing for a particular state or a state is missing for a particular value.
- If any UPF state other than OFF, PARTIAL_ON, FULL_ON, or UNDETERMINED is specified in the table when there is no 1:1 mapping.

Semantic Checks for the connect_supply_net -vct Command

The following semantic checks apply to the `connect_supply_net -vct <vct_name>` command:

- The VCT name must be one of the predefined VCT names or a user-defined VCT.

- A ground/nwell-type supply net cannot be connected with predefined VCT of type non-ground (`UPF2VHDL_SL`, `UPF2SV_LOGIC`, `VHDL_TIED_HI`, `SV_TIED_HI`, `VHDL_TIED_LO`, `SV_TIED_LO`, `VHDL_SL2UPF`, `SV_LOGIC2UPF`, `SV_LOGIC2UPF_MD`).
- The type of the VCT must match the type of the HDL port connection. It is an error to use a `HDL2UPF_VCT` when the Supply net drives the HDL port and vice versa.
- If multiple ports are defined in `connect_supply_net`, user-defined VCT matches only few ports, rest of the ports use the default VCT. VCS NLP issues a warning message in this case.

Limitations

- User-defined VCTs are not supported with the `set_design_attributes` command.

Multi-Rail Macro Simulation

Multi-Rail Macro (MRM) simulation associates different HDL supply rail connections to different PG ports. The PG cell liberty contains the related power pin and ground pin for input and output ports. The `connect_supply_net` command establishes connection from the UPF supply ports to the HDL supplies.

Use Model

The following are the inputs that you require to setup the Multi-Rail Macro simulation:

- The `-power_config` file.

Note:

For MRM, you must specify the link library and database (DB) path of the MRM cells in the `-power_config` file, which is similar to `connect_supply_net` and library mapping feature.

- UPF compliant liberty DB's for all Multi-Rail Macros.

Note:

Only liberty DB's are supported. Library files (.lib) are not supported.

- UPF with `connect_supply_net` statements for the power pins of all MRM instances.
- If required, use the `set_related_supply_net` command for the boundary port of the design.

You require these inputs only when the primary input/output ports are not associated with the same supply rail as the power domain, in which the design top is partitioned.

Usage Example

The following is the usage example of the MRM cell:

```
cell (mad3amux2x1) {
    area : 1316.044800;
    dont_use : true;
    dont_touch : true;
    interface_timing : true;
    timing_model_type : "abstracted";
    pg_pin (VDDIO) {
        pg_type : "primary_power";
        voltage_name : "VDDIO";
    }
    pg_pin (VDDR) {
        pg_type : "backup_power";
        voltage_name : "VDDR";
    }
    pg_pin (VSS) {
        pg_type : "primary_ground";
        voltage_name : "VSS";
    }
    pin ("In0_VIO_Q9") {
        direction : "input";
        capacitance : 238.046500;
        max_transition : 56.456710;
        related_power_pin : VDDR;
        related_ground_pin : VSS;
    }
}
```

```

        pin ("In1_VIO_Q9") {
            direction : "input";
            capacitance : 203.497300;
            max_transition : 56.545580;
            related_power_pin : VDDR;
            related_ground_pin : VSS;
        }
        pin ("LowPower_VIO") {
            direction : "input";
            capacitance : 10.103840;
            max_transition : 56.483670;
            related_power_pin : VDDR;
            related_ground_pin : VSS;
        }
        pin (Sel) {
            direction : "input";
            capacitance : 3.405169;
            max_transition : 56.473290;
            related_power_pin : VDDR;
            related_ground_pin : VSS;
        }
        pin ("Out0_VIO_Q9") {
            direction : "output";
            related_power_pin : VDDR;
            related_ground_pin : VSS;
        }
        pin ("Out1_VIO_Q9") {
            direction : "output";
            related_power_pin : VDDIO;
            related_ground_pin : VSS;
        }
    }
    default_operating_conditions : "BALANCED";
    /* Wire length (mm) for 8 fanouts */
    default_wire_load : "DEFAULT";
}

```

In the above example, all pins are connected to power pin VDDR, except for Out1_VIO_Q9, which is connected to VDDIO.

The following is the example UPF:

```
## CREATE POWER DOMAINS ##
create_power_domain Island_V1 -elements inst1
create_power_domain Island_V2 -elements inst2
create_power_domain Island_V3 -elements inst3
create_power_domain Island_multi_rail -elements
MULTI_RAIL_MACRO
create_power_domain Island_VTOP

# CREATE SUPPLY PORTS  ##
create_supply_port VTOP_VDD
create_supply_port VTOP_VSS
create_supply_port V1_VDD
create_supply_port V2_VDD
create_supply_port V3_VDD
create_supply_port MULTI_RAIL_PRIMARY_POWER

## CREATE SUPPLY NETS  ##
create_supply_net V1 -domain Island_V1
create_supply_net V2 -domain Island_V2
create_supply_net V3 -domain Island_V3
create_supply_net MULTI_RAIL_PRIMARY_POWER -domain
Island_multi_rail

create_supply_net VDD -domain Island_VTOP
create_supply_net VSS -domain Island_VTOP
create_supply_net VSS -domain Island_V1 -reuse
create_supply_net VDD -domain Island_V1 -reuse
create_supply_net VSS -domain Island_V2 -reuse
create_supply_net VSS -domain Island_V3 -reuse
create_supply_net VSS -domain Island_multi_rail -reuse

## CONNECT SUPPLY NETS  ##
connect_supply_net VDD -ports {VTOP_VDD MULTI_RAIL_MACRO/
VDDR}
connect_supply_net VSS -ports {VTOP_VSS MULTI_RAIL_MACRO/
VSS}
connect_supply_net V1 -ports V1_VDD
connect_supply_net V2 -ports V2_VDD
connect_supply_net V3 -ports V3_VDD
connect_supply_net MULTI_RAIL_PRIMARY_POWER -ports
```

```

{MULTI_RAIL_PRIMARY_POWER MULTI_RAIL_MACRO/VDDIO}

## DOMAIN SUPPLY NET      ##
set_domain_supply_net Island_V1 -primary_power_net V1 -
primary_ground_net VSS
set_domain_supply_net Island_V2 -primary_power_net V2 -
primary_ground_net VSS
set_domain_supply_net Island_V3 -primary_power_net V3 -
primary_ground_net VSS
set_domain_supply_net Island_VTOP -primary_power_net VDD -
primary_ground_net VSS
set_domain_supply_net Island_multi_rail -primary_power_net
MULTI_RAIL_PRIMARY_POWER -primary_ground_net VSS

```

In the above UPF, the MRM pin `Out1_VIO_Q9`'s power pin `VDDIO` is connected to a different UPF supply port, and pins `VDDR` and `VSS` are connected to a different UPF supply port.

For the Multi-Rail Macro simulation to work, you must define the DB search path and link library, as you do for `connect_supply_net` and library mapping.

Note:

DB matching is done for Verilog portion of the design only.

Configuring Port-Based Corruption on Macros

You can use the `SNPS_macro_port_based_corruption` design attribute at compile time to force/prevent port-based corruption (by setting it to `ENABLE` or `DISABLE`) for a particular macro or for the full design.

By default, port-based corruption is enabled for the cells identified as non-power-aware, and disabled for the power aware cells.

Use Model

Use the following command to enable port-based corruption for all the cells in the design:

```
set_design_attributes -attribute  
SNPS_macro_port_based_corruption ENABLE
```

Use the following command to disable port-based corruption for all the cells in the design:

```
set_design_attributes -attribute  
SNPS_macro_port_based_corruption DISABLE
```

Use the following command to enable port-based corruption for specific macro(s):

```
set_design_attributes -models <list_of_models> -  
attribute SNPS_macro_port_based_corruption ENABLE
```

Use the following command to disable port-based corruption for specific macro(s):

```
set_design_attributes -models <list_of_models> -  
attribute SNPS_macro_port_based_corruption DISABLE
```

Note:

Any usage of the `set_simstate_behavior` command in UPF is ignored if the usage of `set_design_attributes SNPS_macro_port_based_corruption` is found.

Example

Example 16-1 Verilog Design Example

```
module MACRO_MR (A, CLK, SLEEP, RST, Q, VDDP, VDDA, GND);
    output Q;
    input A;
    input CLK;
    input SLEEP;
    input RST;
    input VDDP, VDDA, GND; // all PG pins defined in Verilog

    reg Q;
    reg Qint, A_wire;
    .....
    .....
```

UPF example (enabling port-based corruption on a power aware macro):

```
set_design_attributes -models {MACRO_MR} -attribute
SNPS_macro_port_based_corruption ENABLE
```

Refer to the **PortBasedCorruption** field of the cell in the mvsim_native_reports/library_mapping_after_upf.rpt report, to identify if the port-based corruption is ENABLED/ DISABLED.

```
mvsim_native_reports/library_mapping_after_upf.rpt

Cell: MACRO_MR
Library = MACRO1_LIB: Library Path = macro1.db

Cell Id = 1 Cell Type: Normal Cell
Cell Function: UNKNOWN
IsPowerAwareModuleCell: true
IsNameBasedMatchingDone: No
PortBasedCorruption: Enabled
```

Association of Pre-Inserted Isolation Cells

VCS NLP performs the following:

- Associates an existing isolation cell in the RTL/netlist with the corresponding isolation strategy.
- Generates an association summary report (`mvsim_native_reports/isolation_association_summary.rpt`) capturing the details of the associated isolation cell instances and their associated isolation strategies. This report also captures the list of isolation cells that could not be associated with any isolation strategy in the UPF.
- Inserts the virtual isolation cells for cases where pre-inserted isolation cell is not found or could not be associated.
- Skips the insertion of the virtual isolation cells when the `-power=accurate` compile-time option is specified.

Note:

- For association, liberty definition (.db) must be provided for the pre-inserted Isolation/ELS cells.
- For each boundary port in the report (`mvsim_native_reports/isolation_association.rpt`), `-instance` points to the associated isolation cell instance name.
- Unassociated cell PG pins are tied to the domain supplies.

- To treat the unassociated Isolation/ELS cells as always-on, you must pass the below design attribute at runtime through the power configuration file:

```
set_design_attributes -attribute
SNPS_keep_unassociated_iso_on TRUE
```

```
% simv -power config.tcl
```

- The following factors are considered as hard constraints for association:

-location

-applies_to

These factors should match between the isolation cell in the design and the isolation strategy in the UPF.

Examples

The following example design code contains isolation cells and isolation cells inferred by VCS NLP based on the UPF:

Example 16-2 Isolation Cells in the Low Power Design

```
top.v
//isolation cell already present in the rtl

DMISAB2 cnt1_0_iso(
    .A(cnt_out1_iso_out[0]),
    .B(iso_en),
    .Z(cnt_out1[0])
);

DMISAB2 cnt1_1_iso(
    .A(cnt_out1_iso_out[1]),
    .B(iso_en),
```

```

.Z(cnt_out1[1])
);

iso.upf

//isolation strategy specified in the upf

set_isolation iso_pd3 \
-domain PD3 \
-clamp_value 0 \
-applies_to both

set_isolation_control iso_pd3 \
-domain PD3 \
-isolation_signal iso_en \
-isolation_sense high \
-location self

```

Following is the sample output of the isolation_association.rpt report file:

```

// information on isolation cells present in the rtl

# Number of Locations for Instrumentation: 1
set ISO_SIGNET_PORT(3) { { SEQ/cnt_out1[1] -policy PD1/
isolation/iso_pd1 -domain PD1 }
{ -isolation_signal iso_en -isolation_sense high -
clamp_value 0 -location parent }
{ -instrumented_node cnt_out1_iso_out[1] -instance
testbench.DUT.cnt1_1_iso -driver_supply PD1_prim -
receiver_supply TOP_prim } }

# Number of Locations for Instrumentation: 1
set ISO_SIGNET_PORT(4) { { SEQ/cnt_out1[0] -policy PD1/
isolation/iso_pd1 -domain PD1 }
{ -isolation_signal iso_en -isolation_sense high -
clamp_value 0 -location parent }
{ -instrumented_node cnt_out1_iso_out[0] -instance
testbench.DUT.cnt1_0_iso -driver_supply PD1_prim -
receiver_supply TOP_prim } }

```

```

// information on the isolation cells inferred by VCS NLP
based on isolation strategy

# Number of Locations for Instrumentation: 1
set ISO_SIGNET_PORT(13) { { COMB/or2bit[0] -policy PD3/
isolation/iso_pd3 -domain PD3 }
{ -isolation_signal iso_en -isolation_sense high -
clamp_value 0 -location self }
{ -instrumented_node COMB/or2bit[0] -driver_supply PD3_prim
-receiver_supply TOP_prim } }

# Number of Locations for Instrumentation: 1
set ISO_SIGNET_PORT(14) { { COMB/and2bit[1] -policy PD3/
isolation/iso_pd3 -domain PD3 }
{ -isolation_signal iso_en -isolation_sense high -
clamp_value 0 -location self }
{ -instrumented_node COMB/and2bit[1] -driver_supply
PD3_prim -receiver_supply TOP_prim } }

```

Support for Timing Simulation With SDF in Presence of UPF

VCS NLP supports the following simulation semantics in presence of the UPF:

- Corruption is applied immediately on an element without any delay when the power domain changes its state from `NORMAL` to `CORRUPT`
- Timing checks on elements of a power domain are turned off when a domain simstate changes from `NORMAL` to `CORRUPT`.
- For the non-power-aware DB cells, the timing checks on the logic ports are turned off if the corresponding related power or ground pins are turned off

- For power-aware cells, timing checks are not turned off automatically. User must ensure that the PG pins are part of the timing check definition
- When a simstate transitions from CORRUPT to NORMAL, timing checks are turned on and all delays are re-evaluated, similar to the time 0 behavior
- All child instances within a DB cell are treated as non-power-managed, hence no specific action related to the timing checks is taken

You can use the `-power=dump_pd_tchk_rpt` compile-time option to capture the information related to the timing checks on elements of a power domain in the `pd_tcheck_mapping.rpt` file generated in the `mvsim_native_reports` directory.

Following is the message format for the disabled timing checks:

```
[Time] [INFO] [LP_TIMING_CHECKS_OFF] All timing checks on
elements inside power domain '<DOMAIN_NAME>' are turned OFF
as the domain simstate changed from NORMAL to CORRUPT.
```

For example,

```
[28 ns] [INFO] [LP_TIMING_CHECK_OFF] All timing checks on
elements inside power domain 'tb/inst/Island_Vdd' are turned
OFF as the domain simstate changed from NORMAL to CORRUPT.
```

Following is the message format for the enabled timing checks:

```
[Time] [INFO] [LP_TIMING_CHECKS_ON] All timing checks on
elements inside power domain '<DOMAIN_NAME>' are turned ON
as the domain simstate changed from CORRUPT to NORMAL.
```

For example,

```
[48 ns] [INFO] [LP_TIMING_CHECK_ON] All timing checks on elements inside power domain 'tb/inst/Island_Vdd' are turned ON as the domain simstate changed from CORRUPT to NORMAL.
```

Following is the report format for the timing checks. This report contains the mapping between instances having timing checks and power domain to which it belongs.

Power domain <Full path of power domain> has the following instances having timing checks.

<Instance name that has timing checks one per line>

For example,

Power domain "tb/pd" has the following instances having timing checks.

"tb/inst"

Specifying Default ON Voltage for Internal PG Pins of a Macro

Internal PG pins of a macro cell are driven from the primary PG pin specified in the pg_function liberty attribute in case of non-power aware model. The state and voltage are derived from the primary PG pin specified in the pg_function liberty attribute.

For power aware model, the internal PG pins are derived from the corresponding Verilog definition of a macro. If it is driven using the object of type UPF::supply_net_type, then the voltage is derived from that driver. If it is driven using 4-state logic (0 | 1 | x | z), then the default ON voltage of 1.0V is taken.

For scenarios where the ON voltage of an internal PG pin is different from that of the primary PG pin, you can define port state using the `add_port_state` command on the internal PG pin, and compile the design using the `-power=voltage_regulator_cell` compile time option.

If there is only one ON state defined for the internal PG pin in the UPF, the voltage value is taken from the port state definition. If multiple ON states are defined on the internal PG pin, you must set the default ON state using the `SNPS_default_supply_net_state` attribute in the UPF. Following is the syntax:

```
add_port_state port_name
{-state {name <nom | min max | min nom max | off>}}}

set_design_attributes -element <supply_port> -attribute
SNPS_default_supply_net_state <state_in_add_port_state>
```

Example

Consider two instances `reg1` and `reg2` of a macro (regulator) with `VDD` (internal Power) and `VCC` (primary power specified with the `pg_function` attribute in the liberty file).

Consider the following Liberty file, `test.lib`:

```
cell( regulator ) {
    switch_cell_type : coarse_grain;
    pg_pin( VCC ) {
        voltage_name : V_POWER;
        pg_type : primary_power;
        direction : input;
    }
    pg_pin( VDD ) {
        voltage_name : V_POWER;
        pg_type : internal_power;
```

```

        direction : internal;
        pg_function : VCC;
        switch_function : !C;
    }
...
}

```

By default, the ON state and voltage on VDD is determined by VCC.

Consider the following UPF file, test.upf:

```

add_port_state VCC -state {ON_3v 3.0} -state {OFF off}
add_port_state sub/reg1/VDD -state {ON_2v 2.0} -state {OFF off}
add_port_state sub/reg2/VDD -state {ON_1v 1.0} -state
{ON1_2v 2.0} -state {OFF off}
set_design_attributes -elements {sub/reg2} -attribute
SNPS_default_supply_net_state ON_1v

```

To meet the requirement to have ON voltage on reg1/VDD as 2.0v and reg2/VDD as 1.0v, compile the design using
-power=voltage_regulator_cell.

For reg1/VDD, only one ON state ON_2v is defined, hence the corresponding voltage 2.0v is taken for reg1/VDD.

If multiple ON states are defined on the internal PG pin, then you must set the default ON state using the SNPS_default_supply_net_state attribute in the UPF.

For reg2/VDD, there are two ON states ON_1V and ON_2v corresponding to 1.0v and 2.0v respectively, and the SNPS_default_supply_net_state attribute is set to ON_1V. Therefore, the corresponding voltage 1.0v is taken for reg2/VDD.

17

Connecting UPF Supply Network to HDL Ports

The following sections describe connecting UPF supply network to HDL ports:

- [Support for SPICE Blocks in UPF](#)
- [Connecting UPF Supply Network to Wreal Supply Ports](#)
- [Connecting UPF Supply Network to SystemVerilog Nettype With Real Base Type](#)
- [Driving UPF Supply Network Through SPICE](#)
- [Support for Multiple View Modules for SPICE with UPF](#)
- [Supporting UPF in the VCS and CustomSim Cosimulation Flow](#)

Support for SPICE Blocks in UPF

VCS NLP now allows references to SPICE blocks in the UPF. It automatically considers SPICE blocks in the UPF as a power black box and parses it without giving any error message.

All the properties of power black box are applicable for SPICE blocks in the UPF. There is no need to declare SPICE blocks as a black box explicitly.

Use Model

Following is the use model to run low power designs with analog or SPICE blocks in the UPF:

```
% vcs -adopt upf -ad=vcsAD.init -upf <upf_file.upf>  
-sverilog <design files>
```

Following is the sample code of `vcsAD.init` file:

```
choose xa -nspice top.spi;  
  
use_spice -cell inv_spi -inst tb.DUT.I1;
```

Where `top.spi` is input file of the command and contains the definition of SPICE module. The `use_spice` command indicates the module-instance to be considered as a SPICE module.

If `-cell` is specified, then all the instances of the cell are considered as SPICE modules, otherwise only the specified instance is considered as a SPICE module. If `-inst` is specified without `-cell`, then VCS NLP issues an error message.

If a cell is marked as SPICE cell, then all instances under it are treated as a SPICE cell.

VCS NLP reports auto black box SPICE module instances in the `mvsim_native_reports/spice_black_box_insts.rpt` file. Following is the sample `spice_black_box_insts.rpt` file:

```
Spice black box Instance Name = tb.DUT.I0 Module Name =  
inv_spi
```

Note:

This feature is supported only for the Verilog part of a mixed design.

Connecting UPF Supply Network to Wreal Supply Ports

VCS NLP allows you to connect wreal signals to the UPF supply network using the `connect_supply_net` UPF command.

This feature allows you to connect wreal netlist to the UPF supplies in a mixed-signal design environment.

The following sections describe the use model:

- [Connecting UPF Supply Nets to Wreal Input and Inout Ports](#)
- [Connecting UPF Supply Nets to Wreal Output Ports](#)

Connecting UPF Supply Nets to Wreal Input and Inout Ports

You can use the `connect_supply_net` command to connect UPF supply nets to wreal input and inout ports.

VCS NLP considers `connect_supply_net` to wreal inout connection as `connect_supply_net` to wreal input connection.

As per the UPF resolution semantics, resolved value of the UPF driver is propagated to the wreal port after being converted through CSN-TO-WREAL Value Conversion Table (VCT). Propagated value is resolved with other wreal drivers (if any) according to the wreal resolution semantics.

Figure 17-1 Connecting UPF supply Nets to Wreal Input Ports

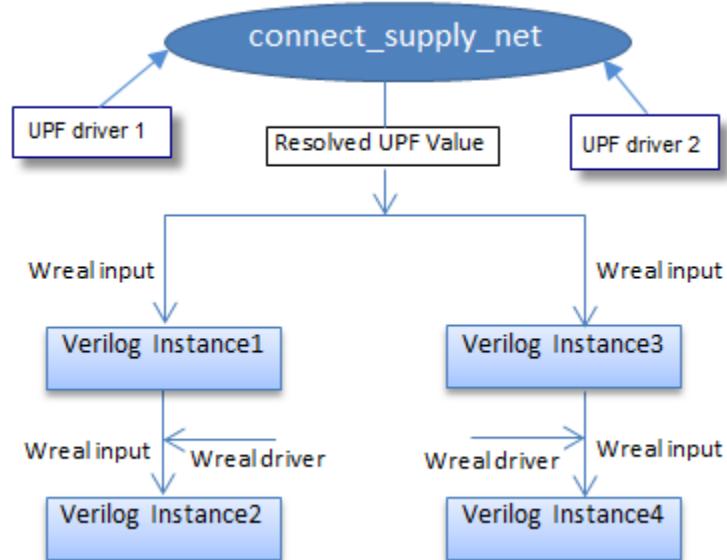
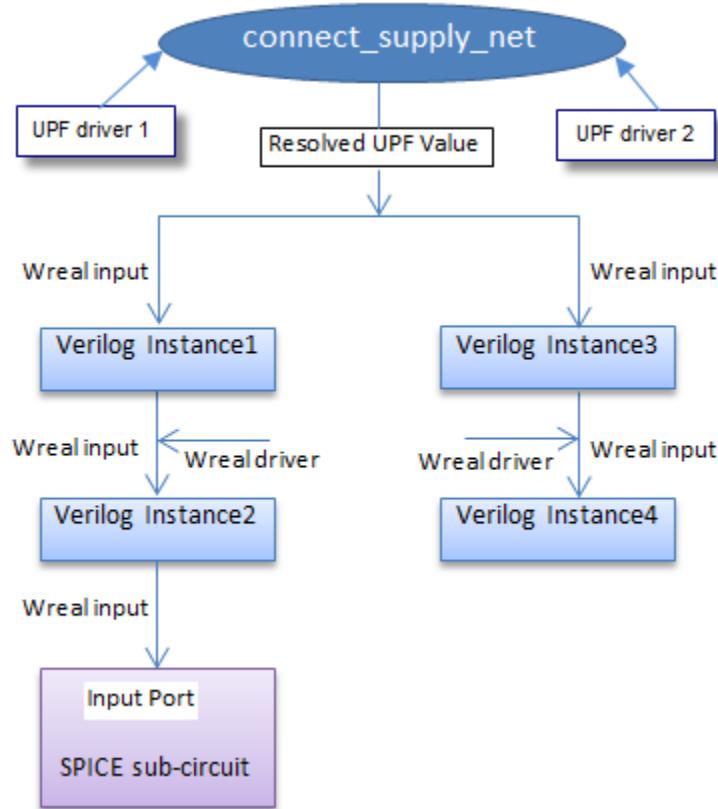


Figure 17-1 shows UPF to wreal supply connection in Verilog hierarchy.

VCS NLP also supports connecting UPF supply nets to the SPICE blocks through wreal network. [Figure 17-2](#) shows UPF to wreal to SPICE connection through the Verilog hierarchy.

Figure 17-2 UPF->Wreal->SPICE Connection Through Verilog Hierarchy



Example

Example 17-1 test_input.v

```

`timescale 10ps/10ps

module tb();
  import UPF::*;
  top top();
endmodule

initial begin

```

```

$display("start simulation");
    supply_on("vdd", 3.00);
    supply_on("vss", 0.00);
#1 supply_on("vdd", 3.1);
#1 supply_off("vdd");
#1 supply_on("vdd", 3.20);
#1 supply_off("vdd");
#1 supply_on("vdd", 3.3);
#1 $finish;
end
endmodule

module top();
wreal vdd1, vss1;

real rvdd, rvss;

assign vdd1 = 2.0;
assign vss1 = 2.0;

assign rvdd = vdd1;
assign rvss = vss1;
always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss =
%f", $time, rvdd, rvss);

bot BOT (vdd1, vss1);
endmodule

module bot(vdd2, vss2);
input wreal vdd2, vss2;
real rvdd, rvss;

assign vdd2 = 1.0;

assign rvdd = vdd2;
assign rvss = vss2;
always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss =
%f", $time, rvdd, rvss);

disp DISP (vdd2, vss2);
endmodule

```

```

module disp(vdd3, vss3);
    input wreal vdd3, vss3;
    real rvdd, rvss;

    assign vdd3 = 1.0;

    assign rvdd = vdd3;
    assign rvss = vss3;

    always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss
= %f", $time, rvdd, rvss);
endmodule

```

Example 17-2 test_input.upf

```

set_design_top tb/top

create_power_domain pd1 -include_scope

create_supply_net vdd -domain pd1 -resolve unresolved
create_supply_net vss -domain pd1 -resolve unresolved

connect_supply_net vdd -ports { BOT/vdd2 }

```

Compile `test_input.v` as shown below:

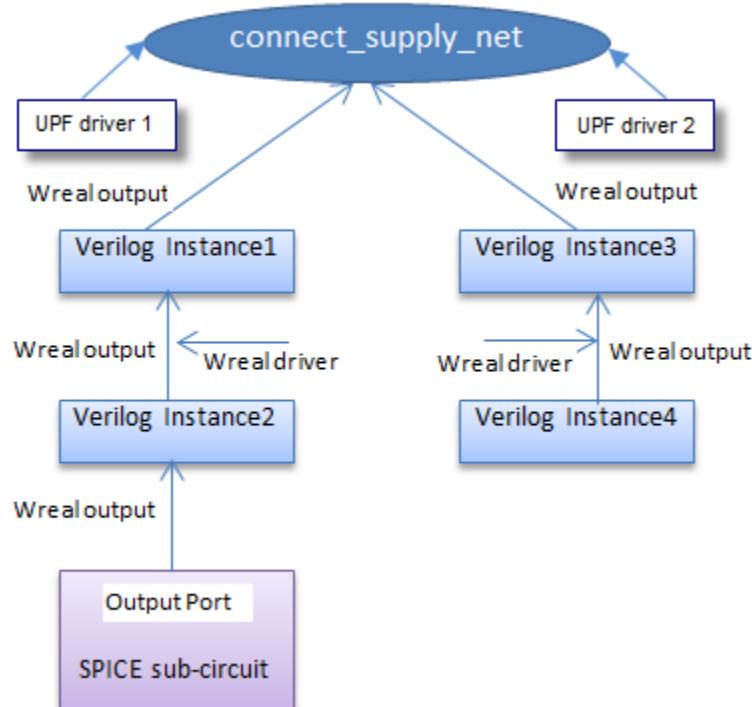
```
% vcs test_input.v -upf test_input.upf -wreal
res_sum -debug_access+all -sverilog
```

Connecting UPF Supply Nets to Wreal Output Ports

You can use the `connect_supply_net` command to connect the UPF supply net to wreal output ports. Resolved value of wreal ports are propagated to `connect_supply_net` after being converted through wreal-to-csn VCT. Propagated value is resolved with other UPF drivers (if any) according to the UPF resolution semantics.

[Figure 17-3](#) shows SPICE to Wreal to UPF connection through the Verilog hierarchy.

Figure 17-3 Connecting UPF supply Nets to Wreal Output Ports



Example

Example 17-3 test_output.v

```
`timescale 10ps/10ps

module tb();
    import UPF::*;
    top top();
endmodule

module top();
```

```

wreal vdd1, vss1;

real rvdd, rvss;

assign vdd1 = 2.0;
assign vss1 = 2.0;

assign rvdd = vdd1;
assign rvss = vss1;
always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss = %f",
    $time, rvdd, rvss);

bot BOT (vdd1, vss1);
endmodule

module bot(vdd2, vss2);
output wreal vdd2, vss2;
real rvdd, rvss;
real rvdddrv, rvssdrv;

assign vdd2 = rvdddrv;
assign vss2 = rvssdrv;

initial begin
$display("start simulation");
rvdddrv = 3.0;
rvssdrv = 0.0;
#1 rvdddrv = 3.1;
#1 rvdddrv = -3.0;
#1 rvdddrv = 3.20;
#1 rvdddrv = -3.0;;
#1 rvdddrv = 3.3;
#1 $finish;
end

assign rvdd = vdd2;
assign rvss = vss2;
always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss = %f",
    $time, rvdd, rvss);

disp DISP (vdd2, vss2);

```

```

endmodule

module disp(vdd3, vss3);
    input wreal vdd3, vss3;
    real rvdd, rvss;

    assign vdd3 = 1.0;

    assign rvdd = vdd3;
    assign rvss = vss3;

    always @(rvdd, rvss) $display ("%g: %m: vdd = %f | vss
= %f", $time, rvdd, rvss);
endmodule

```

Example 17-4 test_output.upf

```

set_design_top tb/top

create_power_domain pd1 -include_scope

create_supply_net vdd -domain pd1 -resolve unresolved
create_supply_net vss -domain pd1 -resolve unresolved

connect_supply_net vdd -ports { BOT/vdd2 }
connect_supply_net vss -ports { BOT/vss2 }

```

Compile test_output.v as shown below:

```
% vcs test_output.v -upf test_output.upf -wreal
res_sum -debug_access+all -sverilog
```

Connecting UPF Supply Network to SystemVerilog Nettype With Real Base Type

VCS NLP allows you to connect UPF supply network to the input and output ports of the real-valued SystemVerilog nettype using the `connect_supply_net` UPF command.

This feature allows you to connect real-valued SystemVerilog netlist to the UPF supplies in a mixed-signal design environment.

VCS NLP generates an error message if SystemVerilog nettype with non-real base type is used. Following is the sample error message:

```
Error-[UPF_NT_CSN] Unsupported Nettype Object with CSN
./Source/mvsim_nlp/nlp_pv/real_nettype_in_csn/
neg_non_real_sv_nettype/neg_non_real_sv_nettype.upf, 15
Element 'tb/top/DISP_UPF_IO/vdd5'
(./Source/mvsim_nlp/nlp_pv/real_nettype_in_csn/
neg_non_real_sv_nettype/neg_non_real_sv_nettype.v,102)
specified in the UPF file is of type 'SV nettype with non-
real base type'.
Reference to 'SV nettype with non-real base type' in UPF
'connect_supply_net' command is not supported.
Please modify the UPF file accordingly and recompile.
```

The following sections describe the use model:

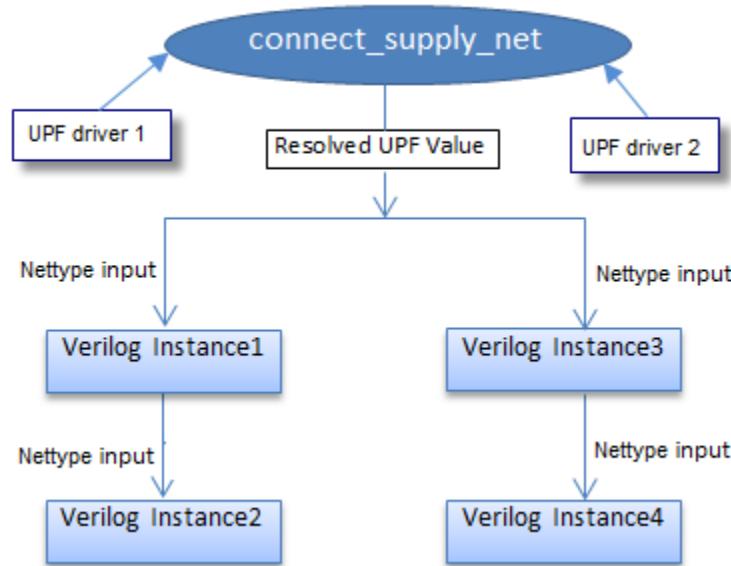
- [UPF Supply Nets Driving Real-Valued SystemVerilog Nettype Ports](#)
- [Real-Valued SystemVerilog Nettype Ports Driving UPF Supply Nets](#)

UPF Supply Nets Driving Real-Valued SystemVerilog Nettype Ports

You can use the `connect_supply_net` command to connect UPF supply nets to the input ports of the real-valued SystemVerilog nettype.

VCS NLP considers `connect_supply_net` to the real-valued nettype inout port connection as `connect_supply_net` to real-valued nettype input connection.

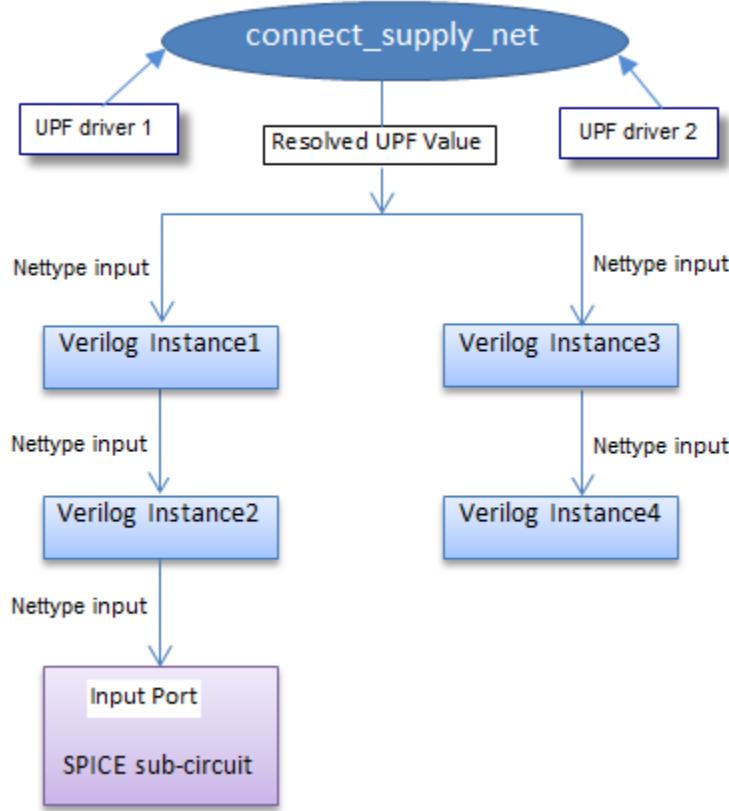
Figure 17-4 UPF Supply Nets Driving Real-Valued SV Nettype Ports



[Figure 17-1](#) shows UPF to real-valued nettype supply connection in Verilog hierarchy.

VCS NLP also supports connecting UPF supply nets to the SPICE blocks through the real-valued nettype hierarchy. [Figure 17-2](#) shows UPF to real-valued nettype supply connection and UPF to real-valued nettype to SPICE connection through Verilog hierarchy.

Figure 17-5 UPF->Nettype->SPICE Connection Through Verilog Hierarchy



Example

Example 17-5 test_input.v

```

`timescale 10ps/10ps

package analog_vl_pack;
    function automatic real analog_t_sum(input real
driver[]);
        analog_t_sum = 0.0;
        foreach (driver[i]) begin
            analog_t_sum += driver[i];
        end
    endfunction
    nettype real analog_nt with analog_t_sum;
endpackage : analog_vl_pack
  
```

```

import analog_vl_pack::*;

module tb();
    import UPF::*;
    top top();

    initial begin
        $display("start simulation");
        supply_on("vdd", 3.00);
        supply_on("vss", 0.00);
        #10 supply_on("vdd", 3.1);
        #10 supply_off("vdd");
        #10 supply_on("vdd", 3.20);
        #10 supply_off("vdd");
        #10 supply_on("vdd", 3.3);
        #10 $finish;
    end
endmodule

module top();
    analog_nt vdd1, vss1;

    real rvdd, rvss;

    assign vdd1 = 2.0;
    assign vss1 = 2.0;

    assign rvdd = vdd1;
    assign rvss = vss1;
    always @ (rvdd, rvss) begin
        #1;
        $display ("%g: %m: =====> vdd = %f | vss = %f", $time,
        rvdd, rvss);
    end

    bot BOT (vdd1, vss1);
endmodule

module bot(vdd2, vss2);
    input analog_nt vdd2, vss2;
    real rvdd, rvss;

```

```

assign vdd2 = 1.0;

assign rvdd = vdd2;
assign rvss = vss2;
always @ (rvdd, rvss) begin
#2;
$display ("%g: %m: =====> vdd = %f | vss = %f",
$time, rvdd, rvss);
end

disp DISP (vdd2, vss2);
endmodule

module disp(vdd3, vss3);
input analog_nt vdd3, vss3;
real rvdd, rvss;

assign vdd3 = 1.0;

assign rvdd = vdd3;
assign rvss = vss3;

always @ (rvdd, rvss) begin
#3;
$display ("%g: %m: =====> vdd = %f | vss = %f",
$time, rvdd, rvss);
end
endmodule

```

Example 17-6 test_input.upf

```

set_design_top tb/top

create_power_domain pd1 -include_scope

create_supply_net vdd -domain pd1 -resolve unresolved
create_supply_net vss -domain pd1 -resolve unresolved

connect_supply_net vdd -ports { BOT/vdd2 }

```

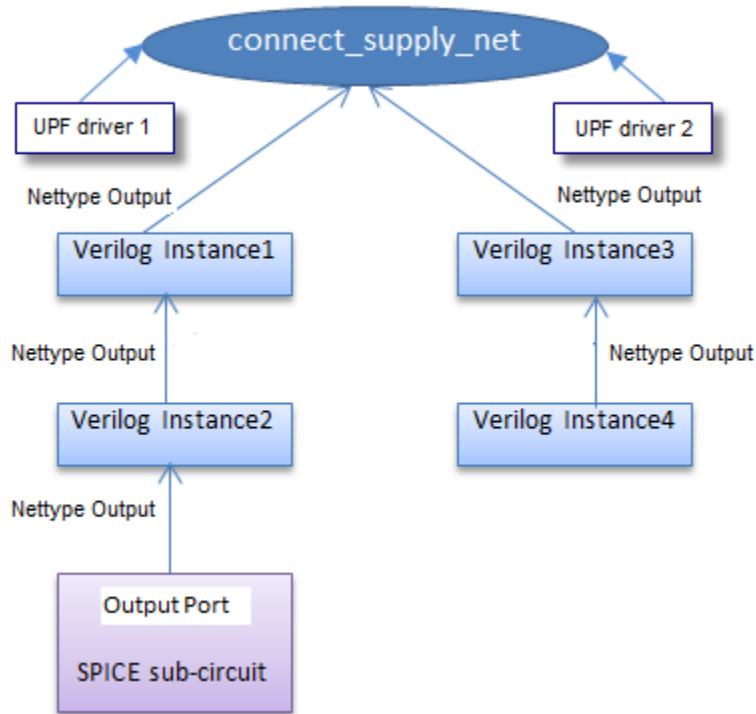
Real-Valued SystemVerilog Nettype Ports Driving UPF Supply Nets

You can use the `connect_supply_net` command to connect the UPF supply net to real-valued nettype output ports.

VCS NLP also supports connecting UPF supply nets to the SPICE blocks through the real-valued nettype hierarchy.

[Figure 17-1](#) shows real-valued nettype to UPF supply connection and SPICE to real-valued nettype to UPF connection through Verilog hierarchy.

Figure 17-6 SPICE->Nettype->UPF Connection Through Verilog Hierarchy



Example

Example 17-7 test_input.v

```
`timescale `timescale 10ps/10ps

package analog_vl_pack;
    function automatic real analog_t_sum(input real
driver[]);
        analog_t_sum = 0.0;
        foreach (driver[i]) begin
            analog_t_sum += driver[i];
        end
    endfunction
    nettype real analog_nt with analog_t_sum;
endpackage : analog_vl_pack

import analog_vl_pack::*;

module tb();
    import UPF::*;
    top top();
endmodule

module top();
    analog_nt vdd1, vss1;

    real rvdd, rvss;

    assign vdd1 = 2.0;
    assign vss1 = 2.0;

    assign rvdd = vdd1;
    assign rvss = vss1;
    always @ (rvdd, rvss) begin
        #1;
        $display ("%g: %m: =====> vdd = %f | vss = %f", $time,
rvdd, rvss);
    end

    bot BOT (vdd1, vss1);
endmodule
```

```

module bot(vdd2, vss2);
    output analog_nt vdd2, vss2;
    real rvdd, rvss;
    real rvdddrv, rvssdrv;

    assign vdd2 = rvdddrv;
    assign vss2 = rvssdrv;

    initial begin
        $display("start simulation");
        rvdddrv = 3.0;
        rvssdrv = 0.0;
        #10 rvdddrv = 3.1;
        #10 rvdddrv = -3.0;
        #10 rvdddrv = 3.20;
        #10 rvdddrv = -3.0;;
        #10 rvdddrv = 3.3;
        #10 $finish;
    end

    assign rvdd = vdd2;
    assign rvss = vss2;
    always @(rvdd, rvss) begin
        #2;
        $display ("%g: %m: =====> vdd = %f | vss = %f",
        $time, rvdd, rvss);
    end

    disp DISP (vdd2, vss2);
endmodule

module disp(vdd3, vss3);
    input analog_nt vdd3, vss3;
    real rvdd, rvss;

    assign vdd3 = 1.0;

    assign rvdd = vdd3;
    assign rvss = vss3;

```

```

    always @ (rvdd, rvss) begin
        #3;
        $display ("%g: %m: =====> vdd = %f | vss = %f",
$time, rvdd, rvss);
    end
endmodule

```

Example 17-8 test_output.upf

```

set_design_top tb/top

create_power_domain pd1 -include_scope

create_supply_net vdd -domain pd1 -resolve unresolved
create_supply_net vss -domain pd1 -resolve unresolved

connect_supply_net vdd -ports { BOT/vdd2 }
connect_supply_net vss -ports { BOT/vss2 }

```

Driving UPF Supply Network Through SPICE

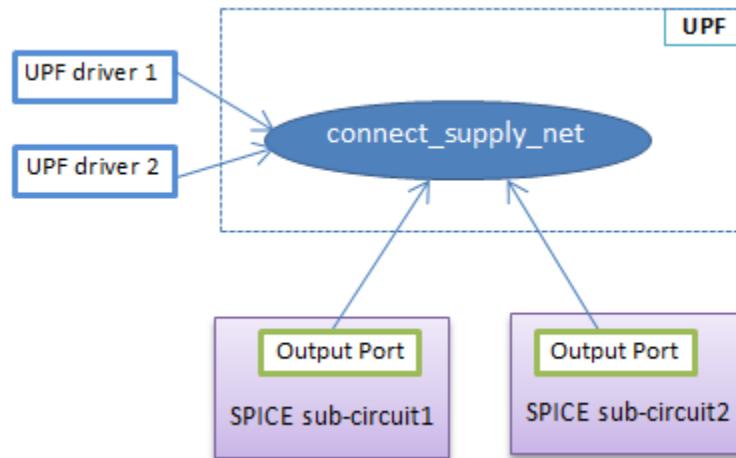
VCS NLP supports SPICE supply ports driving the UPF supply nets. You can connect the SPICE supply ports to the UPF supply nets. Both SPICE and UPF can drive the UPF supply network. These connections allow you to transfer the voltage state and value change information from CustomSim to VCS NLP.

If multiple UPF and SPICE drivers are specified on the UPF supply net, the drivers are resolved as per the UPF resolution mechanism, so that the UPF supply network is driven by the resolved power supply.

Use Model

Multiple SPICE Output Ports and UPF Drivers Driving the UPF Supply Net

Figure 17-7 SPICE Output Ports and UPF Drivers Driving UPF Supply Net

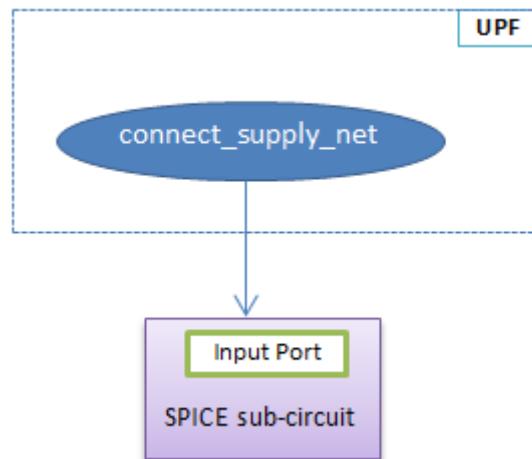


In Figure 17-7, the SPICE and UPF drivers are resolved as per the UPF resolution mechanism. The resolved driver drives the UPF supply net.

Key Points to Note

- If the SPICE port is an input port, as shown in Figure 17-8, UPF supply net drives the SPICE sub-circuit.

Figure 17-8 UPF Supply Net Driving SPICE Input Port



- If the SPICE port is an inout port, as shown in [Figure 17-9](#), then its direction is considered as input if:

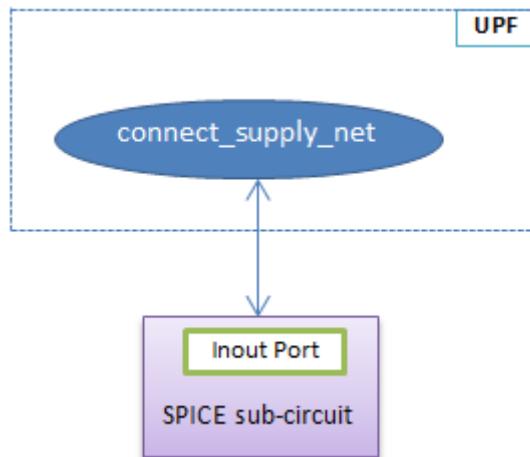
At least one UPF driver (non-SPICE) is present on the corresponding supply net.

Or,

The corresponding supply net has the resolve type UNRESOLVED.

Otherwise, the direction of the inout SPICE port is considered as output.

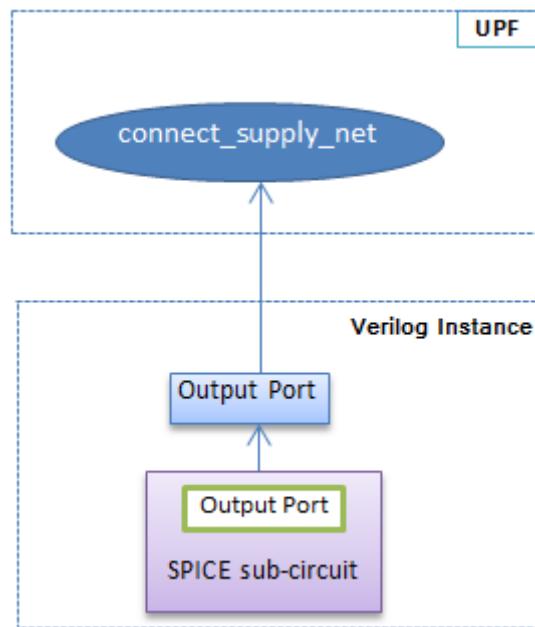
Figure 17-9 SPICE Inout Port Driving the UPF Supply Net



SPICE Ports Connected to the UPF Supply Net Through HDL

[Figure 17-10](#) shows the SPICE output port connected to the UPF supply net through HDL. The SPICE block is instantiated within a Verilog instance. The SPICE supply port is connected to the output supply port in the Verilog layer. VCS performs connectivity tracing and finds the SPICE power supply connection to the UPF supply net.

Figure 17-10 SPICE Ports Connected to the UPF Supply Net Through HDL



Support for Multiple View Modules for SPICE with UPF

Mixed-signal simulation with UPF allows designs containing SystemVerilog, VHDL, SPICE, and UPF to be simulated together.

In pure SPICE simulation, the direction and data-type of SPICE sub-circuit ports have no significance. However, in the context of mixed-signal simulation, the direction and type of a boundary SPICE port play an important role to communicate the signal values in the right direction. For example, value communication can happen from SPICE to Digital, Digital to SPICE, or both.

The type and direction of a boundary SPICE port can also be specified as `UPF::supply_net_type` using a multiple-view Verilog module. Using the multiple-view module, VCS

communicates the direction and the value-format of a boundary SPICE port to the SPICE kernel for proper value communication to happen at simulation time.

VCS allows a limited set of data-types, which can be specified for the multiple-view module ports. Some of these data types are as follows:

- Wire
- Real
- Real-nettype
- supply_net_type from UPF-package

The supply_net_type is a packed SystemVerilog structure that is defined in the built-in UPF-package as follows:

Example 17-9 Packed SystemVerilog Structure

```
typedef enum {
    OFF,
    UNDETERMINED,
    PARTIAL_ON,
    FULL_ON
} state;
typedef struct packed {
    state state;
    int voltage;
}
```

The UPF::supply_net_type is used as the type of a multiple-view-module port when the corresponding port of the SPICE sub-circuit is a power/ground port.

Use Model

The direction of the `UPF::supply_net_type` port can be either input or output. Following are the port parameters:

Port Width

The `UPF::supply_net_type` port of a SPICE multiple-view module can be scalar. The `UPF::supply_net_type` array-port is not supported.

High-conn of Port

The high-conn of the `UPF::supply_net_type` port of a SPICE sub-circuit can either be wire or `UPF::supply_net_type` object. If it is a wire, it must be used for analog-to-analog connections.

Scope of SPICE Instance

There is no restriction on the SPICE sub-circuit instantiations.

UPF commands

The `connect_supply_net` (CSN) commands are applicable to the `UPF::supply_net_type` SPICE ports. SRSN/SPA commands are not applicable.

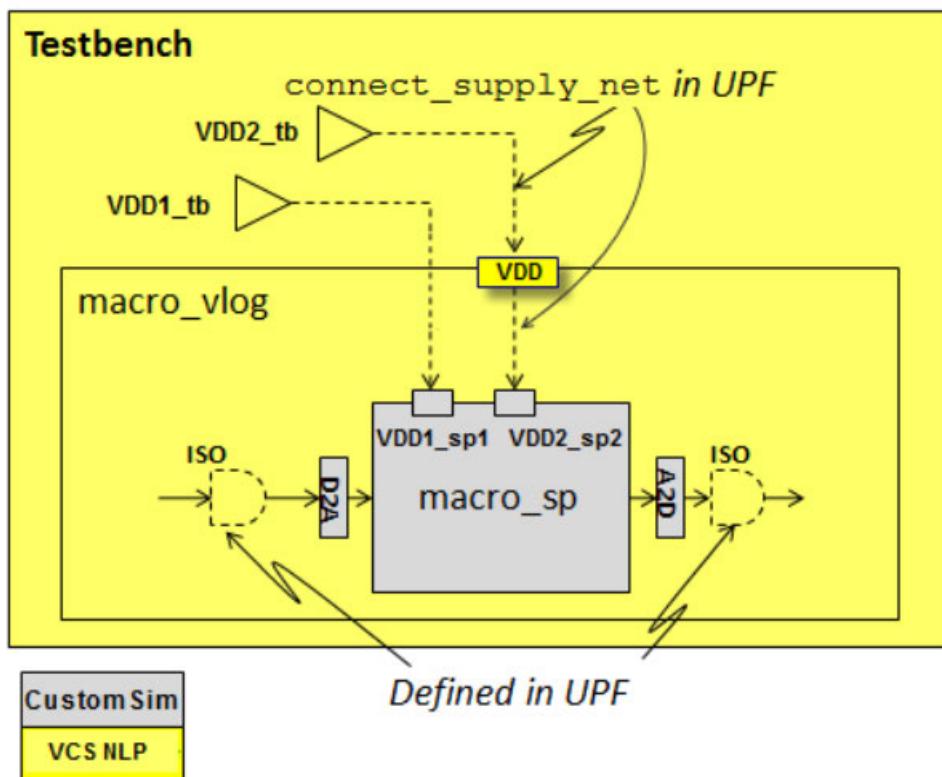
Supporting UPF in the VCS and CustomSim Cosimulation Flow

VCS NLP allows you to define supply network from the UPF for the entire design described in Verilog HDL, VHDL, and SPICE (Analog circuit simulation language supported by CustomSim).

You can use the UPF-based flow to simulate the entire supply network information and pass the supply-pin state information to the SPICE blocks during the VCS and CustomSim cosimulation, as shown in [Figure 17-11](#). This allows you to eliminate the supply nets from RTL and specify the supply nets for the entire design in the UPF to drive the supply pins in the SPICE blocks.

With the new UPF-based flow, the analog portion is treated as a black box, but the power supply for the SPICE blocks is directly controlled from the UPF.

Figure 17-11 Support for UPF in the VCS and CustomSim Cosimulation Flow



Use Model

To enable the UPF-based VCS and CustomSim cosimulation, you must specify the `-upf` option along with the `-ad` option in the `vcs` command-line.

Perform the following steps to setup the UPF-based VCS and CustomSim cosimulation environment:

1. Provide the following file to the current VCS and CustomSim cosimulation setup:
 - UPF for the complete hierarchy under the DUT being simulated

2. Since the supply nets are defined in the UPF, you must remove the supply nets defined in the RTL. These nets, if not removed from RTL, are simulated by VCS NLP as the regular signal nets, and get the UPF behavior (isolation, ON/OFF corruption) simulated on them.

Driving SPICE Supply Ports

This section describes the following topics:

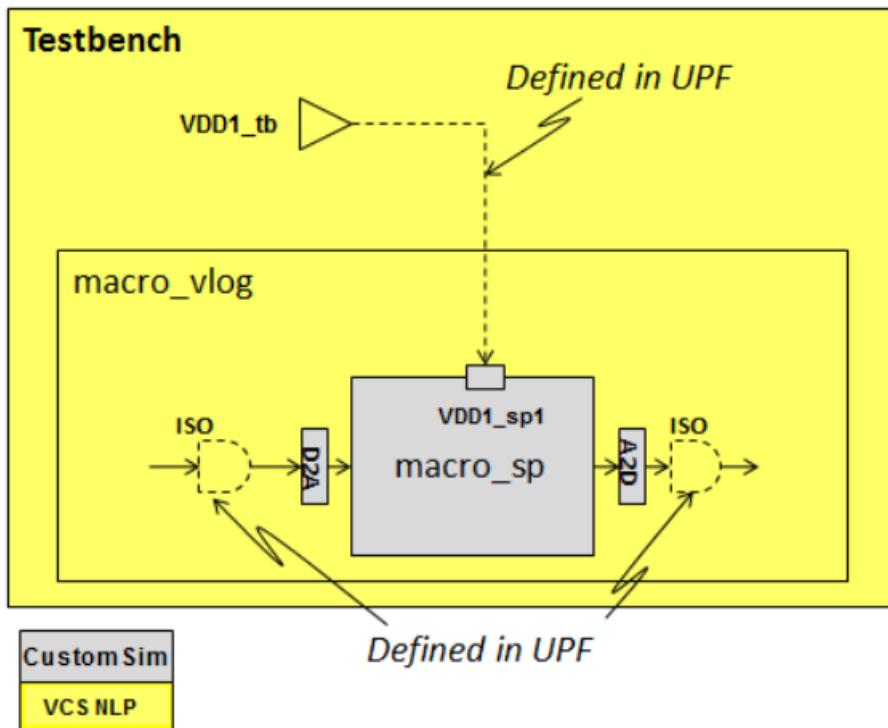
- [Using UPF to Specify Supply Pin Connections to the SPICE Blocks](#)
- [Driving SPICE Supply Ports Through HDL Supply](#)

Using UPF to Specify Supply Pin Connections to the SPICE Blocks

You can use the UPF to specify the supply pin connections to the SPICE blocks, as shown in [Figure 17-12](#). These connections allow you to transfer the voltage state and value information from VCS NLP to CustomSim. The CustomSim engine uses this information to simulate the corresponding supply pins and associated SPICE blocks.

By default, all supply pins of a SPICE instance are connected to the primary supply net of the UPF power domain to which the SPICE instance belongs. These default or implicit connections to the domain's primary supplies happens only if `.db` (liberty definition) is provided.

Figure 17-12 Specifying Supply Pin Connections to the SPICE Blocks



You can override this default behavior by specifying an explicit connection in the UPF using the `connect_supply_net` command. For example, as shown below:

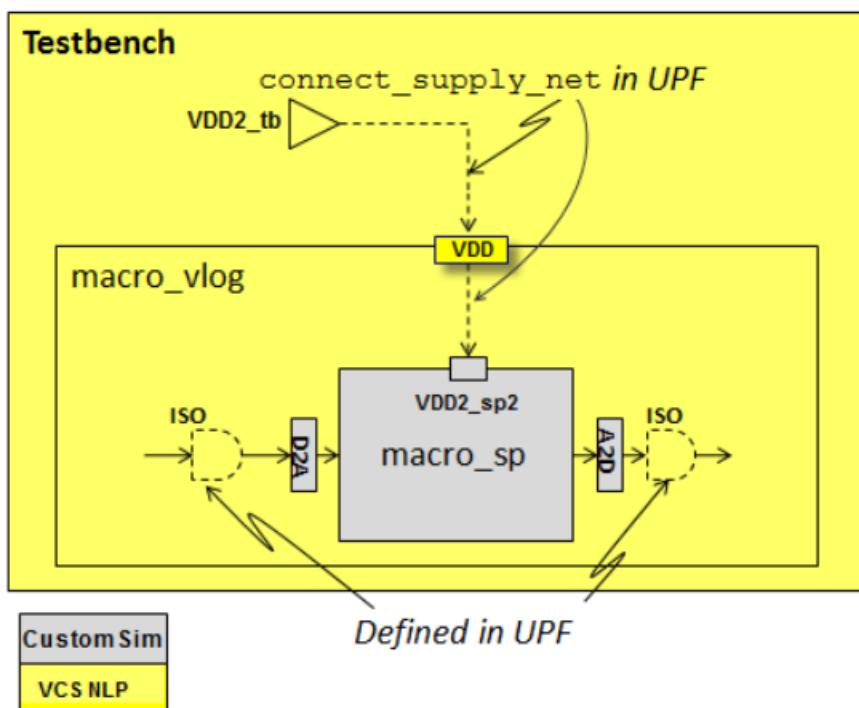
```
connect_supply_net VDD1_tb -ports inst1/inst2/
macroInst1/VDD1_sp1
```

Where, `inst1/inst2/macroInst1/VDD1_sp1` is the SPICE port.

Driving SPICE Supply Ports Through HDL Supply

SPICE supply ports are driven through the HDL nodes which in turn are driven by the UPF supply nets using the `connect_supply_net` command, as shown in [Figure 17-13](#). Any change in either voltage or on/off state is propagated to both the analog (SPICE) and the Verilog (digital) models.

Figure 17-13 Driving SPICE Supply Ports Through HDL Supply



Note:

The UPF supply net connection to the SPICE supply ports should not be done through buffers, jumpered ports, XMRs, aliases, inverters, or any logic in the RTL.

Example of jumpered port:

```
module jumpered_port(.p(a), .q(a));
```

```
    inout a;  
endmodule
```

Example of aliasing:

```
module sv_alias(a, b);  
    inout a, b;  
    alias a = b;  
endmodule
```

For example, consider the following examples:

Example 17-10 Verilog Testcase

```
module a_inv(o, i, vdd, vss);  
    output o;  
    input i;  
    input vdd, vss;  
    prt x_prt_i (o, i, vdd, vss); //SPICE instance  
endmodule
```

Example 17-11 UPF

```
create_supply_port vdd_pd -domain pd1  
create_supply_net vdd_pd -domain pd1  
connect_supply_net vdd_pd -ports vdd_pd  
  
connect_supply_net vdd_pd -ports a_inv/vdd  
connect_supply_net vss -ports a_inv/vss
```

Example 17-12 Testbench

```
supply_on ("vdd_pd", 3.0);  
#200  
supply_off ("vdd_pd", 0.0);
```

In the above examples, when the UPF supply net vdd_pd is switched off, the SPICE supply port is also turned off.

Isolating Digital/Analog Interface Signals

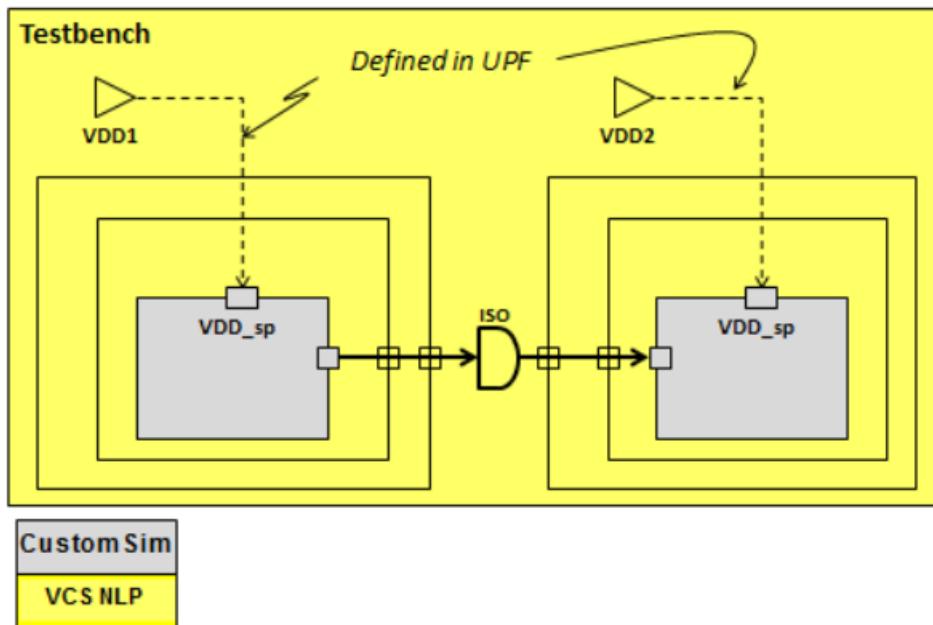
You can use the UPF to define isolation cell instances using the `set_isolation` command. VCS NLP simulates all isolation cells that are defined in the UPF and applicable to the digital part of the design.

If a SPICE instance contains an inbuilt isolation (that is, isolation that is already modeled in SPICE), you must make sure that there is no additional isolation specified for it in the UPF. If an isolation strategy in the UPF is applicable to a particular port of a SPICE instance, VCS NLP inserts that isolation irrespective of whether an internal isolation already exists.

Key Points to Note

- Any UPF command (except `connect_supply_net`) whose target is a SPICE instance, generates an error message.
- A2D and D2A conversions are done by the CustomSim engine, similar to the current use model in the VCS and CustomSim co-simulation flow. Operational voltage level for the logic input and output ports is defined in the mixed signal control file.
- VCS NLP does not do any low power instrumentation on the nested Verilog instance inside the SPICE instance.
- VCS NLP will not be aware of through nets between two SPICE instances. If this tunnel is represented in the RTL as a Verilog wire connection, VCS NLP treats it as any other wire connection and apply UPF semantics (isolation, corruption, and so on) to it (see [Figure 17-14](#)). If such wires exist, you must mark these nets as `dont_touch` using the `UPF_dont_touch` design attribute in the UPF file.

Figure 17-14 Tunneling Between SPICE Instances



Limitations

Following are the limitations:

- Merged VPD mode is not supported.
- VCS NLP does not support the following hierarchy for Mixed Signal Verification:

Verilog → SPICE → Verilog → SPICE

The SPICE sub-circuit cannot have VHDL as either parent or child.

- A nested Verilog instance inside SPICE can be the power top. However, UPF cannot be specified for another nested Verilog instance.

- UPF must not refer to any object in the nested Verilog instance under a SPICE instance.
- The `connect_supply_net` command is supported from or to the SPICE ports only if it is a digital-analog boundary and visible from the power-top.
- Supply net connection through XMRs, aliases, and jumper ports in the RTL is not supported.
- Connection to SPICE port of any direction is supported. However, SPICE ports are treated either as input or output when there is a connection between UPF and SPICE.

18

Automated Assertions

This chapter describes the automated assertions features in the following sections:

- Power-on Reset Assertions Support
- Checking for Assertion of Isolation Enable When Source Supply is OFF
- Identifying the Crossover Ports That do not Toggle
- Handling of Isolation Cells on Top-Level Ports with Pad Cells
- Reporting Write Attempt on OFF Domain Element

Power-on Reset Assertions Support

You can use the `-power=assert_reset_sequence` compile-time option to enable the power-on reset assertions in VCS NLP. To enable reset assertions, you must specify `Tmax` and `Twidth` in the `set_design_attributes` command as shown below:

```
set_design_attributes -attribute  
SNPS_active_reset_duration Twidth
```

```
set_design_attributes -attribute  
SNPS_inactive_reset_duration Tmax
```

Where,

`Twidth`

Minimum time duration for which the reset must be asserted after power-on.

`Tmax`

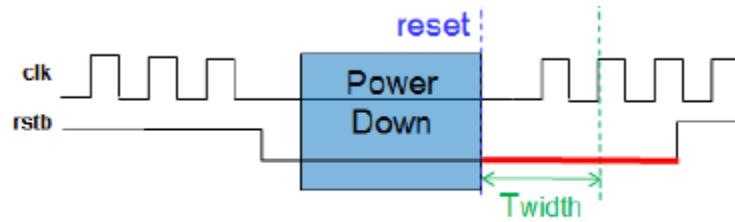
Maximum time limit after power-on within which reset must be asserted.

Example

Consider the following snippet of code:

```
always @(posedge clk or negedge rstb)  
  if (!rstb)  
    q <= 1'b0;  
  else  
    q <= dint;
```

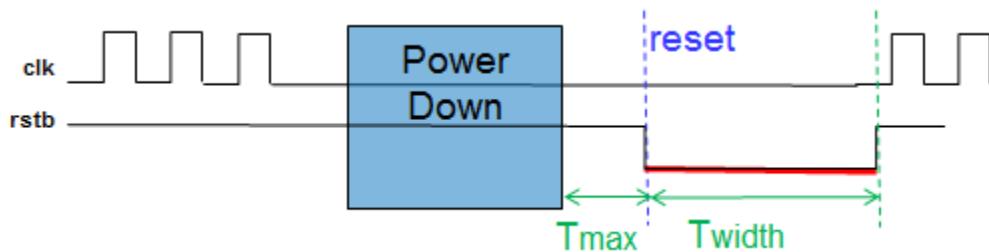
Reset Case-1



In the above figure, reset is asserted before Power Down. After power up, reset is asserted until T_{width} .

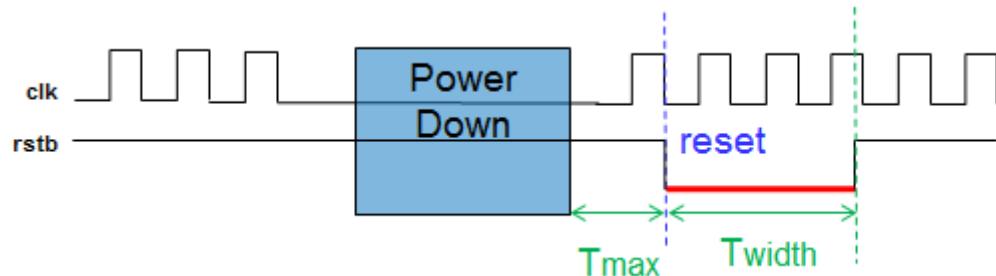
Since reset is already asserted during power up, T_{max} check is not required. However, VCS NLP checks that the reset remains asserted at least for T_{width} time.

Reset Case-2



In the above figure, reset is asserted after Power Down, and the clock is gated.

Reset Case-3



In the above figure, reset is asserted after Power Down, and the clock is not gated.

For **Reset Case-2** and **Reset Case-3**, VCS NLP checks if reset gets asserted within T_{max} time after power up or not. Once asserted, VCS NLP checks that the reset remains asserted for at least T_{width} time.

Use Model

Use the `-power=assert_reset_sequence` option at compile time, as shown below:

```
% vcs -upf <upf_file> -sverilog file_name.v -  
debug_access+all -power=assert_reset_sequence
```

Following is the sample content in the UPF file to set T_{max} and T_{width} :

```
//Attributes defined to set Tmax, Twidth  
  
set_design_attributes -attribute  
SNPS_inactive_reset_duration 5  
  
set_design_attributes -attribute  
SNPS_active_reset_duration 3
```

Based on the above sample UPF file:

- Reset must be asserted within 5 time units (T_{max}) after power-up
- Once reset is asserted, it must stay asserted for at least 3 time units (T_{width}) after power-up

VCS NLP generates the following assertion messages if T_{max} and T_{width} are not specified:

- LP_RESET_ASSERT_FAIL
- LP_RESET_DEASSERT_FAIL

Following are the sample warning messages:

```
[430000 ps] [WARNING] [LP_RESET_ASSERT_FAIL] Reset has not been asserted within '15' timeunit after power up. Instance: 'tb/u_top.FF_sub', Domain : 'tb/u_top/PD_FF', Clock : 'clk', Reset : 'rstb'.
```

```
[295000 ps] [WARNING] [LP_RESET_DEASSERT_FAIL] Reset is deasserted within '20' timeunit after power up. Instance: 'tb/u_top.FF_sub', Domain : 'tb/u_top/PD_FF', Clock : 'clk', Reset : 'rstb'
```

Limitations

- Only asynchronous resets are supported
- Multiple asynchronous loads are not supported:
always @ (posedge clk, posedge set, negedge reset)
- Tmax/Twidth is applicable for the entire design
- This feature does not support waiver mechanism for assertions on instance basis

Checking for Assertion of Isolation Enable When Source Supply is OFF

You can use the `-power=src_off_clamp_check` compile-time option to enable automated check at runtime for an assertion of isolation enable when the source supply is OFF.

The `-power=src_off_clamp_check` option generates the `LP_SRC_OFF_CLAMP` message at runtime, if isolation enable is not active when the source supply for the clamp cell is turned off.

Following is the sample error message:

```
[ERROR] [LP_SRC_OFF_CLAMP] Isolation enable is not asserted  
when input to isolation cell  
'testbench.top_wr_fflop_inst.result1' is corrupted due to  
source domain 'testbench/top_wr/PD_FF' being turned off.  
Isolation Strategy: 'testbench/top_wr/PD_FF/isolation/  
V1_ISO', Isolation Enable: 'testbench/top_wr/iso_ctrl',  
Reference Domain: 'testbench/top_wr/PD_FF', Port:  
'testbench.top_wr_fflop_inst.result1 [7]'.
```

Key Points to Note

- This feature is supported only for the virtual isolation cells
- This feature is supported only for the isolation strategies specified with the `-source/-sink/-diff_supply_only` options

Limitations

- Partition compile flow is not supported
- No assertion is reported when the source supply of an isolated port is a supply net specified with `set_related_supply_net`

Identifying the Crossover Ports That do not Toggle

The crossover (xover) in VCS Native Low Power (NLP) is a design path that crosses voltage and/or power domain boundaries. Each crossover path may include several design nodes in between that may be hierarchical ports, nets, pins, and low power cells such as isolation cells and level shifter cells.

VCS NLP allows you to check toggle on the crossover ports when the supply source changes its state from power-down to power-up (`FULL_ON`) until the end of the simulation. If the isolation cells are inserted on these crossovers, the outputs of the cells are checked. This feature is enabled using the

`-power=report_port_no_toggle` compile-time option. The crossover toggle information is displayed at the end of the simulation.

This feature allows you to identify the crossover ports that do not return to their original state when the supply source changes its state from shutdown to wake-up. This helps you to debug these crossover ports to find the root cause in early stages.

Note:

If there are multiple shutdown/power-up sequences, the toggle should be considered to be covered if it happens after any of those sequences. The toggle need not be checked for each sequence.

VCS NLP displays the crossover port toggle information at the end of the simulation. Following is the sample crossover port toggle information:

```
[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port  
'tb.top.instA1.in[1]' of power domain 'tb/top/A' did not  
toggle after the state of source supply 'tb/top/VDD' changed
```

to FULL_ON.

[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port 'tb.top.instA1.in[2]' of power domain 'tb/top/A' did not toggle after the state of source supply 'tb/top/VDD' changed to FULL_ON.

[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port 'tb.top.instA1.in[3]' of power domain 'tb/top/A' did not toggle after the state of source supply 'tb/top/VDD' changed to FULL_ON.

[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port 'tb.top.instA1.in[4]' of power domain 'tb/top/A' did not toggle after the state of source supply 'tb/top/VDD' changed to FULL_ON.

[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port 'tb.top.instA1.in[5]' of power domain 'tb/top/A' did not toggle after the state of source supply 'tb/top/VDD' changed to FULL_ON.

[110 ns] [ERROR] [LP_BOUNDARY_PORT_NO_TOGGLE] Boundary port 'tb.top.instA1.in[6]' of power domain 'tb/top/A' did not toggle after the state of source supply 'tb/top/VDD' changed to FULL_ON.

Limitations

- This feature is not supported with the X-bash flow, partition compile flow, and VHDL
- This feature is not supported with the following option:
-power=sinkbasedisoassert

Handling of Isolation Cells on Top-Level Ports with Pad Cells

When a pad cell pin is connected to a top-level port, VCS NLP prevents the insertion of isolation on the path from the pin to the top-level port. You can use the following methods to specify a cell as a pad cell:

- Using Liberty Attributes

A cell is considered as a pad cell if the `is_pad:true` attribute is specified for one or more of the cell pins along with the `pad_cell:true` attribute.

- Using the `set_port_attributes` command

If the liberty files are not provided, you can specify `is_pad` as an attribute in the `set_port_attributes` command as follows:

```
set_port_attributes -model <PAD_MODULE_NAME> -  
ports <PORT_NAME> -attribute {is_pad TRUE}
```

In the above command, `PAD_MODULE_NAME` is the name of the Verilog/VHDL module or entity. The port `PORT_NAME` is a pad port. The `-ports` option allows you to specify the list of pad ports.

Example:

```
set_port_attributes -model {rtl_1} -ports  
{iso_enable} -attribute is_pad TRUE
```

Reporting Write Attempt on OFF Domain Element

VCS NLP generates the runtime warning messages listed in [Table 18-1](#) when an attempt to write on a power domain element is detected, and the domain is in CORRUPT simstate. VCS NLP blocks the write on the power domain element. To enable this feature, use the `-power=check_power_down_write` compile-time option.

Table 18-1 Warning Messages

Action	Warning Message
XMR write to a Verilog signal from the Verilog scope	LP_HDL_XMR_WRITE_BLOCKED
Signal write through vpi/acc (PLI call)	LP_PLI_SIGNAL_WRITE_BLOCKED
Signal write from UCLI	LP_UCLI_SIGNAL_WRITE_BLOCKED

Supported Scenarios

This feature is supported for the following writes on elements of an OFF domain:

Scenario-1

`$deposit on a signal`

Example:

```
$deposit("chipTb.chipDut.domain.myreg"....)
```

Scenario-2

Writing to a reg through an XMR in the initial block

Example:

```
initial chipTb.chipDut.domain.myreg = 1'b0;
```

Scenario-3

force statement through an XMR

Example:

```
initial force chipTb.chipDut.domain.myout[9:0] = 10'b0;
```

Scenario-4

Changing the value of a signal using \$hdl_xmr

Example:

```
reg mysig;
initial $hdl_xmr("chipTb.chipDut.domain.myreg", "mysig");
```

Use Model

```
% vcs -sverilog -debug_access+all -upf file.upf
test.v -power=check_power_down_write -P pli.tab
pli.c
```

```
% simv -ucli -i ucli_file
```

Following is the sample output:

```
[WARNING] [LP_HDL_XMR_WRITE_BLOCKED] XMR write from scope 'test'
to signal/variable 'test.t.inv.mem' is blocked as the simstate
of the corresponding power domain 'test/t/A' is 'CORRUPT'. File
: test.v, Line : 21
```

```
[WARNING] [LP_PLI_SIGNAL_WRITE_BLOCKED] XMR write from scope
```

```
'test' to signal/variable 'test.t.inv.mem[1]' is blocked as the
simstate of the corresponding power domain 'test/t/A' is
'CORRUPT'. File : test.v, Line : 49

ucli% force {test.t.inv.mem[1]} 1
[WARNING] [LP_UCLI_SIGNAL_WRITE_BLOCKED] XMR write from scope
'PowerNetworkModel.\ps_test/t .A' to signal/variable
'test.t.inv.mem[1]' is blocked as the simstate of the
corresponding power domain 'test/t/A' is 'CORRUPT'. File :
UCLI_CMD, Line : 0.
```

Limitations

This feature does not support the following:

- Writing through the VHPI call
- XMR write through the continuous assign statement
- The following runtime force semantics:
- FORCE_PRIORITY, FORCE_AND_CORRUPT, and DEFER_FORCES

19

UPF Encryption Support

VCS NLP supports full encryption of the UPF IP code. You can use either one of the following two modes to achieve full encryption of the UPF IP:

- [UPF Encryption Using upf_protect1 Option](#)
- [UPF Encryption Using gen_ip Binary](#)

VCS NLP generates the encrypted file in the following format:

`<original_upf_file_name>.e`

For example, if the original UPF file name is `TOP_IP1.upf`, then the generated encrypted UPF file name will be `TOP_IP1.upf.e`, as shown in [Figure 19-1](#).

By default, VCS NLP generates the encrypted file in the current working directory. You can also specify the directory in which the generated encrypted file must be dumped. You cannot open or view the contents of an encrypted file.

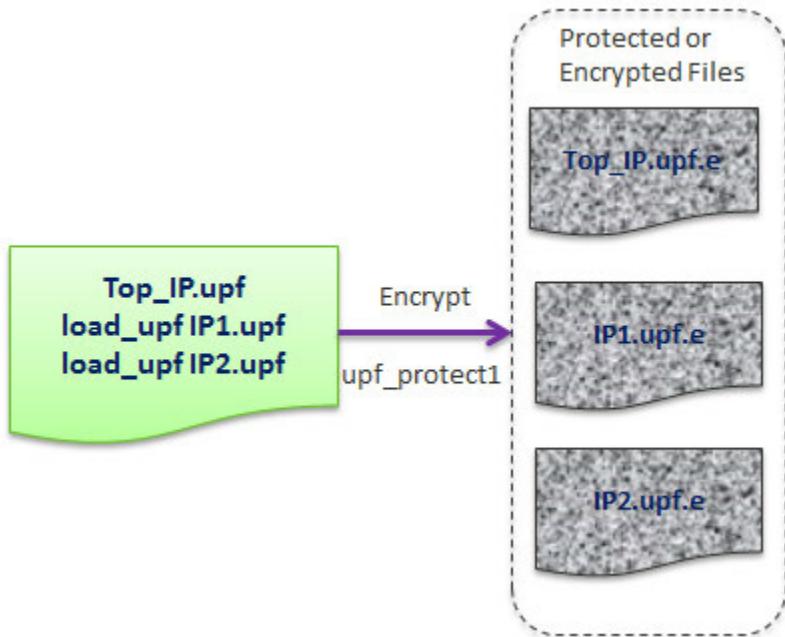
You can use the `+inccdir` compile time option to specify the encrypted UPF file location.

UPF Encryption Using `upf_protect1` Option

You can use the `-power=upf_protect1` compile-time option to automatically encrypt all the UPF files, including the UPF files specified with the `load_upf` command.

In this mode, VCS NLP performs complete syntax and semantic check on all the UPF files, including those specified with the `load_upf` command. VCS NLP generates individual protected or encrypted file corresponding to each original UPF file as shown in [Figure 19-1](#).

Figure 19-1 UPF Encryption Using upf_protect1 Option



Example

Consider the following command-line:

```
%vcs -upf <top.upf> tb.v top.v -power=upf_protect1
```

If `top.upf` loads `bot.upf`, then VCS NLP automatically generates both `top.upf.e` and `bot.upf.e` in the current working directory.

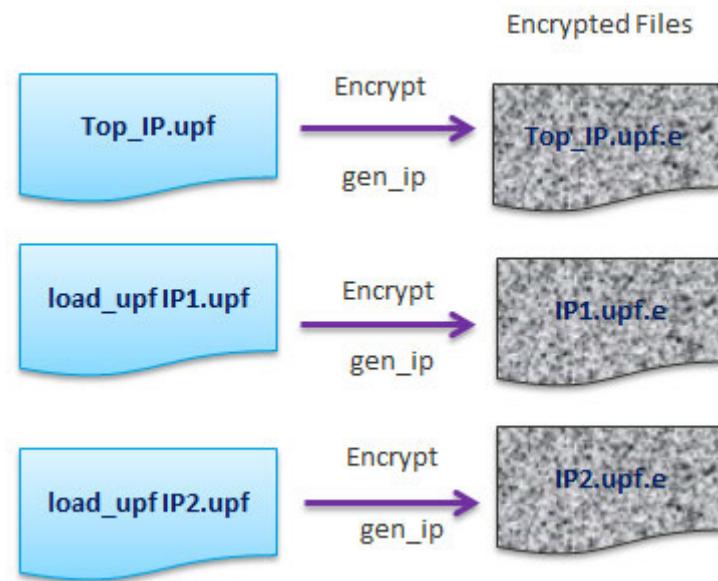
UPF Encryption Using gen_ip Binary

You can use the `gen_ip` standalone binary to encrypt only the specified UPF file. Lower level UPF files are not encrypted, that is, you must individually encrypt each UPF file specified with the `load_upf` command along with the top IP UPF file.

In this mode, no syntax and semantic check is performed on the UPF file. No RTL or design is required to be provided.

VCS NLP generates encrypted file for the specified UPF file as shown in [Figure 19-2](#).

Figure 19-2 UPF Encryption Using gen_ip Binary



Examples

Example-1:

Consider the following command-line:

```
$VCS_HOME/bin/gen_ip -f tcl top.upf
```

VCS NLP generates only `top.upf.e` in the current working directory.

Example-2:

Consider the following command-line:

```
$VCS_HOME/bin/gen_ip -work my_protect_dir -f tcl top.upf
```

VCS NLP generates only `top.upf.e` in the `my_prot_dir` directory.

Example-3:

Consider the following command-line:

```
$VCS_HOME/bin/gen_ip -work myprot -f tcl top.upf bot.upf
```

VCS NLP generates both `top.upf.e` and `bot.upf.e` in the `myprot` directory.

Encrypted UPF File Usage

If the current working directory or include path does not contain unencrypted UPF file, then VCS NLP automatically searches for the encrypted UPF file. For example, consider the following command line:

```
%vcs -upf top.upf -f tb.v top.v
```

If the current working directory contains `top.upf`, then VCS NLP reads `top.upf`. If the current working directory contains the encrypted UPF file `top.upf.e` instead of the unencrypted UPF file `top.upf`, then VCS NLP automatically picks up `top.upf.e`.

Similarly, if the UPF file contains the `load_upf` command as follows,

```
load_upf file.upf
```

and if the current working directory or include path contains encrypted UPF file `file.upf.e` instead of the unencrypted UPF file `file.upf`, then VCS NLP automatically picks up the encrypted UPF file `file.upf.e`. There is no need to modify the `load_upf` command to `load_upf file.upf.e`.

20

Generating Region Based SAIF Reports for Power Analysis

In low power circuits, the dynamic power (which includes switching power) dissipation depends on the logic transitions that occur within the design when operating. For power analysis, you need to accurately specify the switching activity information to the tool performing these tasks.

The Switching Activity Interchange Format (SAIF) file is used to store and pass the switching activity information of a design between power tools and simulators.

VCS now allows you to create regions for sub-blocks, monitor switching activity on the sub-blocks or the entire design unit, and report multiple SAIF files in a single simulation.

You can use the following PLI system tasks and UCLI commands to enable this feature:

\$saif_region_create

This system task allows associating a set of instances in the design with a tag name.

A design can typically contain multiple power domains; hence each domain must be given a unique name with the corresponding instances belonging to it. Similarly, a design can have one set of sub-blocks active only for certain part of the simulation, and another set of sub-blocks active for another part of the same simulation. You may be interested to monitor the switching activity for the group of instances only for a particular duration. So, you can create each such group of sub-blocks as a region.

Syntax

```
$saif_region_create("region_name", <depth>, <instance> {,  
<instance>})
```

Arguments

`region_name`

Unique name of the `$saif_region_create` call comprising of characters a to z, A to Z, 0 to 9, _, and starting with an alphabet. A region can be created at any time during the simulation.

`depth`

Corresponds to the instance hierarchy level, including the specified scopes to be considered for monitoring. A value of 0 means everything under the specified scope(s). Else, a valid positive integer value indicates the depth of the scope which will be considered in this region.

instances

The instance names can correspond to Verilog module instances in cross module reference form or in double quotes. VHDL target or top-level instance names must be enclosed in double quotes.

All instances for a given region must be under the same top level component, if there are multiple top-level units. The instance names can be relative path names or absolute hierarchical path names.

You can specify a hierarchical instance name under multiple regions. However, the switching activity is monitored separately for each region.

All objects (nets/registers) of synthesizable data type are monitored for switching activity in the given instance(s). Specifically, memories, MDAs, array-of-arrays, and VHDL variables are not considered for monitoring.

UCLI Command

```
saif -region <region_name> -depth <depth>  
<scopes>
```

\$saif_region_exclude

This system task allows you to exclude the specified scopes for the given region. These scopes must be in the child hierarchy of the regions specified for monitoring.

Once an instance is specified to be excluded for a given region, it is acceptable to be specified under a different region name for monitoring. For example, consider top.dut1 is registered in region1 and top.dut.uut1 is excluded from it. In such a case, top.dut1.uut1 can be registered in region2.

For a given region, you cannot call this task after the monitoring is started.

Syntax

```
$saif_region_exclude("region_name", <instance> {,  
<instance>})
```

Arguments

region_name

See [region_name](#).

instances

Specifies the list of instances to be excluded for a given region.

UCLI Command

```
saif -region <region_name> <scopes> -depth  
<depth> [-exclude <scopes>]
```

\$saif_region_monitor

This system task allows you to monitor the specified scopes for the given region.

Syntax

```
$saif_region_monitor("region_name", "start"|"stop"|"reset")
```

Arguments

`region_name`

See [region_name](#).

`start`

Sets the region monitor state internally as `start`. The current value of the object and the time at which this task was made is saved.

`stop`

Sets the region monitor state internally as `stop`. In this state, the switching activity is not monitored on the objects in the given region. The duration for which the object has been in T0/T1/TX state since the last change of the value is updated for the object.

`reset`

Sets the region monitor state internally as `reset`. The current value of the object and the time at which this task was made is saved. The toggle counts of the object, T0/T1/TX values, SD/PD counters are reset to zero.

UCLI Command

```
saif [-start|-stop|-reset] <region_name>
```

\$saif_region_report

This system task reports the switching activity for a given region name after issuing stop monitor command for that region. Reports can be generated for a specific region independent of the status of other regions.

Syntax

```
$saif_region_report("region_name", "report.saif",  
<timescale>)
```

Arguments

region_name

See [region_name](#).

report.saif

SAIF file which contains the switching activity information.

timescale

Allows you to specify timescale information. The timescale can be specified in 1e-9 format to indicate 1ns, and so on.

UCLI Command

```
saif -report <file_name> -tres <resolution>  
-region <region_name>
```

\$saif_read_lib

You must call this system task before calling any other \$saif system task. The only exception is \$saif_region_diag system task, which can be called before this system task.

Syntax

```
$saif_read_lib("forward_lib_saif.txt" [,  
"exclude_cells_list_file.txt"])
```

Arguments

`forward_lib_saif.txt`

Text file which corresponds to the library forward SAIF file, typically generated by the `lib2saif` command (Library Compiler tool from Synopsys). You can specify empty string as the first argument, if there is no forward library SAIF file to be considered.

Note:

SAIF does not capture toggle information inside cells, whether they are marked explicitly with ``celldefine` in Verilog source, or implicitly treated as cells when they are pulled in to the design due to the `-v/-y` options on the `vcs` command-line.

`exclude_cells_list_file.txt`

Text file which specifies the list of module names that are not supposed to be considered as cells, instead of considering all the cells as regular modules. This file can have only one module name per line.

Note:

You can use the `+vcs+saif_libcell` option to consider all modules corresponding to `-v/-y` as regular modules.

UCLI Command

```
saif -lib_saif <forward_lib_file.txt> [-  
exclude_cells <file.txt>]
```

\$saif_region_diag

Reports the diagnostic information to the specified text file. The information could include: list of instances considered when creating a region, monitor state change, and report notifications. Each time this task is called, a text file with specific name is opened in write mode. File corresponding to the previous call, if any, is closed.

Syntax

```
$saif_region_diag("diagnostics_file.txt")
```

Arguments

`diagnostics_file.txt`

Text file which contains the diagnostic information.

UCLI Command

```
saif -diag "diagnostics_file.txt"
```

If the above system task calls are present in the Verilog source, then the \$set_toggle_region, \$toggle_start, \$toggle_stop, \$toggle_reset and \$toggle_report, \$read_lib_saif, \$set_gate_level_monitoring tasks are not supported. Similarly, power command is not supported at the ucli prompt.

Example

In [Example 20-1](#), the top module is instantiating add, which is instantiating add1.

SAIF region creation calls are performed twice during the simulation, and two SAIF report files named report.saif and report1.saif are generated in the current working directory. The SAIF report files report.saif and report1.saif contain details of all nets/objects.

Example 20-1 test.v

```
`timescale 1ps/1ps
module top;
logic [3:0] o;
logic [3:0] i,r;

initial begin
    i = 5;
    r = 5;
    #5 i = 10;
    r = 5;
    i = 4;
    r = 3;
    #5 i = 1;
    r = 8;
    i = 5;
    r = 5;
    #5 i = 10;
    r = 5;
```

```

    i = 4;
    r = 3;
    #5 i = 1;
    r = 8;
end

add u1(i,r,o);

initial
begin
    $saif_region_diag("diag.txt");
    $saif_region_create("r", 0, "top.u1");
    $saif_region_exclude("r", "top.u1.u2");
    $saif_region_monitor("r","start");
    #100;
    $saif_region_monitor("r","stop");
    $saif_region_report("r","report.saif",1.0e-12);
    #40;
    $saif_region_create("r1", 0, "top.u1");
    $saif_region_monitor("r1","start");
    #100;
    $saif_region_monitor("r1","stop");
    $saif_region_report("r1","report1.saif",1.0e-12);
    $finish;
end

endmodule

module add(input logic[3:0] c,input logic[3:0] d,output
logic [3:0] o);
logic [3:0] l2;
assign o = l2;
add1 u2(c,d,l2);
endmodule

module add1(input logic[3:0] c,input logic[3:0] d,output
logic [3:0] l2);
always_comb
begin
    l2 = c + d;
end
endmodule

```

Perform the following commands to compile and run [Example 20-1](#).

```
% vcs -sverilog test.v  
% simv
```

The SAIF report files `report.saif` and `report1.saif` are generated in the current working directory.

Alternatively, you can use the following UCLI commands, instead of the PLI calls, if there are no `$saif_region` calls in the testbench:

```
% vcs -sverilog test.v -debug_access+all  
% cat ucli.in  
    saif -region r -depth 0 top.u1 -exclude top.u1.u2  
    saif -start r  
    run 100  
    saif -stop r  
    saif -region r -report report.saif -tres 1e-12  
    run 100  
    saif -region r1 -depth 0 top.u1  
    saif -start r1  
    run 100  
    saif -stop r1  
    saif -region r1 -report report1.saif -tres 1e-12  
    run 100  
    quit  
% simv -ucli -i ucli.in
```


21

Appendix

This chapter contains the following sections:

- Low Power Assertions
- Custom Assertion Checking for Clock/Reset During Shutdown
- Warning Messages for Unconnected Supply Ports
- Root Supply Driver
- TFIPC-L Lint Check in Presence of UPF Connections

Low Power Assertions

This section describes the predefined data types and UPF query commands that are used to write low power assertions. This information is organized into the following sections:

- [List of Automated Low Power Messages](#)
- [Enabling Runtime Isolation Messages](#)
- [Supported UPF Query Commands](#)

List of Automated Low Power Messages

[Table 21-1](#) lists all the automated low power messages.

Table 21-1 Automated Low Power Messages (LP_MSG)

Context	ID	Severity	Sample Message
HDL2UPF Connections	LP_SNET_HDL_SNET_UPDATE	INFO	HDL Supply Net 'tb/u_dut/vddcx_1' updated to value '{OFF, 0}'
	LP_SNET_HDL_SNET_CONN_UPDATE	INFO	HDL Supply Port 'testbench/top/in_logic' driving UPF Supply Net 'testbench/top/Vdd1' updated to value '{OFF, 0}'.
Clock/Reset Wiggle	LP_CLOCK_WIGGLE, LP_RESET_WIGGLE	WARNING	[LP_CLOCK_WIGGLE] Clock 'clk' wiggling in instance 'top.U_foo' during shutdown for the power domain 'top/PD_FOO'. [LP_RESET_WIGGLE] Reset 'reset' wiggling in instance 'testbench.top.fflop_inst' during shutdown for the power domain 'testbench/top/PD_FF'.
	LP_COA_CLOCK_WIGGLE, LP_COA_RESET_WIGGLE	WARNING	[LP_COA_CLOCK_WIGGLE] Clock 'clk' wiggling in instance 'topunit.TOP_U.ul' during standby for the power domain 'topunit/TOP_U/PDB'. [LP_COA_RESET_WIGGLE] Reset 'reset' wiggling in instance 'testbench.top.fflop_inst' during standby for the power domain 'testbench/top/PD_FF'.
Isolation	LP_ISO_EN	INFO	Isolation enabled for isolation strategy 'isolate_u_inst_2_outputs' of power domain 'tb/u_TOP/PD_2'.

Context	ID	Severity	Sample Message
Isolation	LP_ISO_EN_PWRDN	INFO	Isolation enabled for isolation strategy 'ISO_PD2_from_PD1' when Power Domain 'tb/TOP/PD2' is OFF.
	LP_ISO_DIS	INFO	Isolation disabled for isolation strategy 'isolate_1_or_2' of power domain 'tb/u_TOP/PD_2'.
	LP_ISO_DIS_PWRDN	INFO	Isolation disabled for isolation strategy 'ISO_MOD1' when Power Domain 'tb/i1/TOP' is OFF.
	LP_ISOEN_INIT_VALUE	INFO	Isolation enable 'tb/TOP/iso_ctrl' of isolation strategy 'ISO_2_1' of power domain 'tb/TOP/PD2' started with value 'St0'. Isolation sense specified is 'high'.
	LP_ISO_INIT_OFF	WARNING	Isolation strategy 'mem_iso_and' of power domain 'testbench/top/VMEM_domain' started in CORRUPT state.
	LP_ISOEN_INIT_INVALID	WARNING	Isolation enable 'tb/i1/iso_sig' of isolation strategy 'ISO_MOD1' of power domain 'tb/i1/TOP' started with an invalid value 'StX'.
	LP_ISOEN_CHANGE	INFO	Isolation enable 'testbench/DUT/iso_en' of isolation strategy 'iso_pd3' of power domain 'testbench/DUT/PD2' changed from 'St1' to 'St0'. Isolation sense specified is 'high'.

Context	ID	Severity	Sample Message
Isolation	LP_ISOEN_INVALID	ERROR	Isolation enable 'testbench/top/iso_sig2' of isolation strategy 'MY_ISO' of power domain 'testbench/top/Island_V2' changed to an invalid value 'StX'.
	LP_ISOPN_INIT	INFO	Isolation power net 'vddcx_1' for isolation strategy 'iso_low' of power domain 'top_tb/top_inst/SUBA_INT' started in state FULL_ON.
	LP_ISOPN_STATE	INFO	State of isolation power net 'VDD' for isolation strategy 'dma_iso_and' of power domain 'testbench/top/VDMA_domain' changed from OFF to FULL_ON.
	LP_ISOGN_INIT	INFO	Isolation ground net 'ss_PDD.ground' for isolation strategy 'ISO_PDF42_PDT42_1' of power domain 'tbl/t1/PDT42' started in state FULL_ON.
	LP_ISOGN_STATE	INFO	State of isolation ground net 'VSS' for isolation strategy 'isolate_inst_unit1_outputs' of power domain 'topunit/TOP_U/u2/Island_u2' changed from OFF to FULL_ON.
	LP_ISOPN_OFF	WARNING	State of isolation power net 'vdd_soc_n' for isolation strategy 'iso_soc' of power domain 'test/pd_soc' changed to OFF.

Context	ID	Severity	Sample Message
Isolation	LP_ISOGN_OFF	WARNING	State of isolation ground net 'TOP_VSS_NET' for isolation strategy 'isolate_top_outputs' of power domain 'tb/u_TOP/TOP_PD' changed to OFF.
	LP_ISO_SEQFAIL1	WARNING	Isolation enable 'tb/i_dut/i_mrm/pd' for isolation strategy 'iso_CVCC_SW' is not active when the source power domain 'tb/i_dut/i_mrm/CVCC_SW' changed to CORRUPT state.
Low-Vdd Standby/ CORRUPT_ON_ACTIVITY	LP_COA_CORRUPT_ALW	WARNING	LHS in always block corrupted due to activity on sensitivity list elements in simstate 'CORRUPT_ON_ACTIVITY'. Instance: tb.dut_top_inst.u_wrap_6.BUFFER[1].in_buf, File: ./Source/ buffer1.v, Line: 7.
	LP_COA_CORRUPT_CA	WARNING	LHS of cont-assign corrupted due to activity on RHS in simstate 'CORRUPT_ON_ACTIVITY'. Instance: tb.dut_top_inst, File: ./Source/top.v, Line: 64.
	LP_COA_CORRUPT_GATE	WARNING	Output of primitive gate corrupted due to activity on inputs in simstate 'CORRUPT_ON_ACTIVITY'. Instance : 'tb.dut_top_inst.gates_i', File: './Source/gates.v', Line : '22

Context	ID	Severity	Sample Message
Low-Vdd Standby/ <u>CORRUPT_</u> <u>ON_ACTIVITY</u>	LP_COA_CORRUPT_UDP	WARNING	Output of UDP corrupted due to activity on inputs in simstate 'CORRUPT_ON_ACTIVITY'. Instance: tb.inst_top.inst_mem.dff1, File: ./Source/nm.v, Line: 4390.
	LP_COA_CORRUPT_SIG	WARNING	Signal :\\testbench/DUT/COMB\\:SIG_COMB corrupted due to activity.
	LP_COA_CORRUPT_VAR	WARNING	Variable :\\testbench/top\\:INST_VH_2:APF_COMPAT:VAR_FLOP_SIG corrupted due to activity.
Message Control	LP_CFG_FILE_ERR	ERROR	Cannot open specified config file: ./Source/lpconfig_01xz.tcl, please check for the file existence and file permissions and rerun the simulation.
	LP_MSG_SUMMARY	INFO	Message 'LP_MSG_SUMMARY' is now 'DISABLED'.
	LP_MSG_ONOFF	INFO	Message 'LP_MSG_ONOFF' is now 'ENABLED'.
	LP_MSG_NF	WARNING	'tb.v', '91': Message ':' referenced in function 'set_lp_msg_onoff' not found.
	LP_MSG_INVALID_ARG	WARNING	'tb.v', '15': Argument 'Retention power net %s or ground net %s are in off state' in function 'lp_msg_register' is invalid. Valid values are 'ON, OFF'.

Context	ID	Severity	Sample Message
Message Control	LP_MSG_SEV	INFO	Severity of message 'LP_PSW_ISP_STATE' is now 'ERROR'.
	LP_MSG_REG	INFO	Message 'LP_RET_POW_ON' is registered successfully.
	LP_MSG_INT	WARNING	'./Source/tb.v', '93': 'LP_PD_INIT_STATE' is an internal message. 'lp_msg_print' can be used only with messages registered through 'lp_msg_register'.
	LP_MSG_AE	WARNING	'testbench.v', '23': Message 'LP_POWER_DOMAIN' in function 'lp_msg_register' already Exists. Already registered message cannot be registered again.
Power Domain	LP_PD_INIT_STATE	INFO	Power domain 'testbench/top/Island_V1' started in 'NORMAL' state.
	LP_PD_STATE_CHANGE	INFO	Power domain 'tbench/duv/pd_ecc' state changed from 'CORRUPT' to 'NORMAL'.
	LP_PPN_INIT_STATE	INFO	Primary power net 'VDD' of power domain 'tst/itop/PD_VDD_VSS' started in OFF state.
	LP_PPN_INIT_VALUE	INFO	Primary power net 'PWR_SW_OUTw' of power domain 'TOP/F1/PDF1' started with voltage 0 V.

Context	ID	Severity	Sample Message
Power Domain	LP_PGN_INIT_STATE	INFO	Primary ground net 'vss' of power domain 'top/TOP' started in FULL_ON state.
	LP_PGN_INIT_VALUE	INFO	Primary ground net 'VSS' of power domain 'testbench/top/AO' started with voltage 0 V.
	LP_PPN_STATE_CHANGE	INFO	State of the primary power net 'VDD2' of power domain 'tb/testbench/top_inst/PD2' changed from OFF to FULL_ON.
	LP_PGN_STATE_CHANGE	INFO	State of the primary ground net 'ss_PDD.ground' of power domain 'tbl/t1/PDF21' changed from OFF to FULL_ON.
	LP_PPN_VALUE_CHANGE	INFO	Voltage of the primary power net 'V1SW' of power domain 'testbench/top/Island_V1' changed from 1.8 V to 0 V.
	LP_PGN_VALUE_CHANGE	INFO	Voltage of the primary ground net 'SEQ_VSS_NET' of power domain 'tb/u_TOP/SEQ_PD' changed from 0 V to 1 V.
Power Switch	LP_PSW_CTRL_INIT_INVALID	ERROR	Signal 'tb/top/b_on1' connected to control port 'ctrl1' of power switch 'tb/top/PSW_B' started with an invalid value 'StX'. Note: This message is not reported when the input supply of the power switch is in OFF state.

Context	ID	Severity	Sample Message
Power Switch	LP_PSW_CTRL_INIT_VALUE	INFO	Signal 'top/pwr[1]' connected to control port 'ctrl_lr' of power switch 'top/PS' started with a value 'St1'
	LP_PSW_ISP_INIT_STATE	INFO	Supply net 'tb/UUT/Vtop' tied to input supply port 'Vin' of power switch 'tb/UUT/SW_CORE' started with state FULL_ON.
	LP_PSW_ISP_INIT_VALUE	INFO	Supply net 'testbench/VCC1_net' tied to input supply port 'VCC1_port' of power switch 'testbench/top' started with voltage 1 V.
	LP_PSW_OSP_INIT_STATE	INFO	Supply net 'TOP/ab1/PDAB_VDD_OUT_N' tied to output supply port 'OUT' of power switch 'TOP/ab1/PDAB_PSW' started with state UNDETERMINED.
	LP_PSW_OSP_INIT_VALUE	INFO	Supply net 'top/blk1/VDD' tied to output supply port 'VDD' of power switch 'top/blk1/blk_sw2' started with voltage 0 V.
	LP_PSW_INIT_STATE	INFO	Power switch 'tb/i1/SOURCE_switch' started in ON_1 (ON) state.
	LP_PSW_STATE_CHANGE	INFO	State of power switch 'TB_TOP/IO2/vdd_switch' changed from logic_off (OFF) to logic_on (ON).

Context	ID	Severity	Sample Message
Power Switch	LP_PSW_CTRL_CHANGE	INFO	Signal 'tb/top/c_on' connected to control port 'ctrl3' of power switch 'tb/top/PSW_C' changed from 'St0' to 'St1'.
	LP_PSW_CTRL_INVALID	ERROR	<p>Signal 'testbench/top/psw_ctrl' connected to control port 'ctrl' of power switch 'testbench/top/V1_header_switch' changed from 'St0' to an invalid value 'StX'.</p> <p>Note: This message is not issued when the input supply of the power switch is in OFF state.</p> <p>This message is issued on the change in the power switch control signal when the input supply of the power switch is in ON state.</p> <p>The message is issued even if the input supply of the power switch switches to OFF state in the same simulation cycle after the power switch control signal is sampled as X or Z.</p>
	LP_PSW_OSP_STATE	INFO	State of supply net 'tb/DUT/Vswp' tied to output supply port 'Vout' of power switch 'tb/DUT/sw_periph' changed from FULL_ON to OFF.
	LP_PSW_OSP_VALUE	INFO	Voltage of supply net 'tb/dut/VDDMS' tied to output supply port 'out' of power switch 'tb/dut/mult_sw' changed from 0.8 V to 0.6 V.

Context	ID	Severity	Sample Message
Power Switch	LP_PSW_ISP_STATE	INFO	State of supply net 'tb/u_TOP/DVDD_DVFS' tied to input supply port 'RVDD' of power switch 'tb/u_TOP/SW_PD_CPU_0' changed from FULL_ON to OFF.
	LP_PSW_ISP_VALUE	INFO	Voltage of supply net 'tb/dut/VDDM' tied to input supply port 'in' of power switch 'tb/dut/mult_sw' changed from 0.8 V to 1 V.
	LP_PSW_SUPPLY_CHANGE	INFO	Output Supply Port of Power Switch 'testbench/V1_header_switch' changed to state = 'OFF' and voltage = 0 volt(s) with enable expression '0'.
	LP_PSW_INIT	INFO	Output Supply Port of Power Switch 'tb/header_switch' initialized to state = 'FULL_ON' and enable expression '1'.
PST	LP_PST_INIT_STATE	INFO	Design started with 'R1, R2' state in pst 'tb/TOP/my_pst'.
	LP_PST_INIT_ILLEGAL	WARNING	Design started with 'ILLEGAL' state (illegal) in pst 'tb/top/LS'.
	LP_PST_STATE_CHANGE	INFO	Design state changed from 'ILLEGAL' to 's0' in pst 'tb/dut/chiptop_pst'.
	LP_PST_STATE_ILLEGAL	WARNING	Design state changed from 's1' to 'ILLEGAL' (illegal) in pst 'tb/top/LS'.

Context	ID	Severity	Sample Message
PST (enabled using the <code>-power=pst</code> <code>_state_ms</code> <code>g</code> compile- time option)	LP_PST_STATE_INIT	INFO	Design started with 'ALL_ON' state in pst 'tb/t1/sub1/SUB1_PST' with Supplies 'VDD_SUB1=ON,VDD_MUX=ON, VDD_BUF=ON,VSS=ON1:ON'.
	LP_PST_STATE_ILLEGAL_INIT	WARNING	Design started with 'ILLEGAL' state in pst 'testbench/inst/ top_pst' with Supplies 'VDD=illegal,V1=illegal, V2=ON,V3=illegal,VSS= illegal'.
	LP_PST_STATE_ILLEGAL_COMBINATION_INIT	ERROR	Design started with 'ILLEGAL' state in pst 'test/topblk_pst' with Supplies 'SS_VDD1.power=V_OFF, SS_VDD2.power=V_OFF,SS_ VDD1.ground=GND'.
	LP_PST_STATE_TRANSITION	INFO	Design state changed from 'PD3_OFF' to 'ALL_ON' in pst 'tb/ top_inst/pst_1' with Supplies 'VDD = ON, VDD1 = ON, VDD2 = ON, VDD3 = ON, VSS = VSS_ON' and Supply 'VDD3' changed.
	LP_PST_STATE_ILLEGAL_TRANSITION	WARNING	Design state changed from 'ILLEGAL' to 'ILLEGAL' in pst 'test/ topblk_pst' with Supplies 'SS_VDD1.power=illegal ,SS_VDD2.power=V_NOM, SS_VDD1.ground=GND' and Supply 'SS_VDD1.power' changed.

Context	ID	Severity	Sample Message
PST	LP_PST_STATE_ILLEGAL_COMBINATION	ERROR	Design state changed from 'state0' to 'ILLEGAL' in pst 'test/top blk_pst' with Supplies 'SS_VDD1.power=V_MIN, SS_VDD2.power=V_NOM, SS_VDD1.ground=GND' and Supply 'SS_VDD2.power' changed.
Random Corruption (enabled using the -pa_random_corrupt compile-time option)	LP_DESIGN RAND_SEED	INFO	Random corruption seed specified for design is: '71'.
	LP_AUTO_RAND_SEED	INFO	Random corruption seed is not specified for design. Using automatic generated random corruption seed '3'.
Retention	LP_RET_PN_INIT	INFO	Retention power net 'VDDI' for retention strategy 'inst_ret' of power domain 'tb/dut/INST' started with FULL_ON state.
	LP_RET_GN_INIT	INFO	Retention ground net 'VSSw' for retention strategy 'RET' of power domain 'TOP/F1/PDF1' started with FULL_ON state.
	LP_RET_PN_STATE	INFO	State of retention power net 'VRet' for retention strategy 'RET_PD4' of power domain 'testbench/core/PD4' changed from OFF to FULL_ON.

Context	ID	Severity	Sample Message
Retention	LP_RETGN_STATE	INFO	State of retention ground net 'VSSCX' for retention strategy 'retention_ff2' of power domain 'TB/top/macro' changed from OFF to FULL_ON.
	LP_RET_PWR_OFF	WARNING	Retention power for retention strategy 'retain_dflop_1' of power domain 'top/dut_top/DFF_1' turned OFF. Shadow registers are corrupted.
	LP_SAVE_ON	INFO	Save asserted for retention strategy 'RET' of power domain 'top/TOP'.
	LP_SAVE_OFF	INFO	Save deasserted for retention strategy 'RET_MOD1' of power domain 'tb/TOP/MOD1/PD_MOD1'.
	LP_SAVE_RETPWRDN	ERROR	Save event triggered for retention strategy 'my_retention_level_1' of power domain 'testbench/top/Island_V3' when retention power is OFF.
	LP_RESTORE_ON	INFO	Restore asserted for retention strategy 'V1_ret_policy' of power domain 'testbench/PD1'.
	LP_RESTORE_OFF	INFO	Restore deasserted for retention strategy 'flop_ret' of power domain 'tb_top/core/PD4'.

Context	ID	Severity	Sample Message
Retention	LP_RESTORE_RET_PWRDN	ERROR	Restore event triggered for retention strategy 'FUSEPG_RETENTION' of power domain 'top/PD_SPVDD' when retention power is OFF.
	LP_SAVE_INIT	INFO	Save signal 'testbench/core/save' for retention strategy 'RET_PD4' of power domain 'testbench/core/PD4' started with a value 'St0'. Save sense specified is 'high'.
	LP_SAVE_INIT_INVALID	WARNING	Save signal 'ChipTop/inst_save' for retention strategy 'inst_ret' of power domain 'ChipTop/INST' started with an invalid value 'HiZ'. Note: This message is not reported when the retention supplies are in OFF state.
	LP_SAVE_CHANGE	INFO	Save signal 'tb/dut/gprs_save' for retention strategy 'gprs_ret' of power domain 'tb/dut/GPRS' changed from 'St0' to 'St1'. Save sense specified is 'high'.

Context	ID	Severity	Sample Message
Retention	LP_SAVE_INVALID	WARNING	<p>Save signal 'tb/top/u_wrap_B/save_i' for retention strategy 'RET' of power domain 'tb/top/B' changed from 'St0' to 'StX'. Note: This message is not issued when the retention supplies are in OFF state. This message is issued on the change in the save control signal when the retention supply is in ON state.</p> <p>The message is issued even if the retention supply switches to OFF state in the same simulation cycle after the save signal is sampled as X or Z.</p>
	LP_SAVE_PRIMPWR_OFF	WARNING	Save operation enabled for retention strategy 'LEVEL1_ret1' of power domain 'tb/dut_top/LEVEL1_inst1' while the primary power of domain is OFF. Save signal 'tb/dut_top/ret_ctrl' is at an active level 'St1'.
	LP_RESTORE_INIT	INFO	Restore signal 'tb/u_TOP/restore_pd' for retention strategy 'retain_SEQ_2_PD' of power domain 'tb/u_TOP/SEQ_PD' started with a value 'St0'. Restore sense specified is 'high'.

Context	ID	Severity	Sample Message
Retention	LP_RESTORE_INIT_INVALID	WARNING	<p>Restore signal 'tb/top/restore' for retention strategy 'RET_B' of power domain 'tb/top/B' started with an invalid value 'StX'.</p> <p>Note: This message is not reported when the retention supplies are in OFF state.</p>
	LP_RESTORE_CHANGE	INFO	<p>Restore signal 'tb/u_top/restore_top' for retention strategy 'RETAIN_SEQ_4' of power domain 'tb/u_top/SEQ_PD' changed from 'St1' to 'St0'. Restore sense specified is 'high'.</p>
	LP_RESTORE_INVALID	WARNING	<p>Restore signal 'testbench/top/f_restore_1' for retention strategy 'my_retention_level_1' of power domain 'testbench/top/Island_V3' changed from 'St0' to 'StX'.</p> <p>Note: This message is not issued when the retention supplies are in OFF state. This message is issued on the change in the restore control signal when the retention supply is in ON state.</p> <p>The message is issued even if the retention supply switches to OFF state in the same simulation cycle after the restore signal is sampled as X or Z.</p>

Context	ID	Severity	Sample Message
Retention	LP_RESTORE_PRIMPWR_OFF	WARNING	Restore operation disabled. Primary power of power domain 'RET_Ivonoff' turned OFF while restore signal 'fifo_tb/Ivonoff' of retention strategy 'fifo_tb/save_restore' is at an active level 'St0'.
	LP_RESTORE_NOSAVE	ERROR	Restoring X on the registers of power domain 'testbench/top/Island_V3'. Restore operation is enabled without a save event, after retention power is turned ON. Retention Strategy : 'my_retention_level_2', Restore Signal : 'testbench/top/f_restore_3', Save Signal : 'testbench/top/f_save_3'.
	LP_ASSERT_R_MUTEX	ERROR	'assert_r_mutex' defined for retention strategy 'my_retention_edge' of power domain 'testbench/top/Island_V3' failed. Restore signal 'testbench/top/f_restore_2' ('posedge') and signal 'testbench/top/f_restore_3' ('low') are not mutually exclusive.

Context	ID	Severity	Sample Message
Retention assert_mutex	LP_ASSERT_S_MUTEX	ERROR	'assert_s_mutex' defined for retention strategy 'V2_RETENTION' of power domain 'testbench/TOP_U/Island_Vret' failed. Signal 'save' ('high') is not mutually exclusive with the save signal 'testbench/TOP_U/pmuInst/save' ('high').
	LP_ASSERT_RS_MUTEX	ERROR	'assert_rs_mutex' defined for retention strategy 'V2_RETENTION' of power domain 'testbench/TOP_U/Island_Vadd_Vadder_sleep_Vret' failed. Signal 'restore' ('low') and save signal 'testbench/TOP_U/pmuInst/save' ('high') are not mutually exclusive.
Supply Sets [*] [*] Issued only for the supply sets that have the power states defined in the UPF.	LP_SS_INIT	INFO	Supply Set 'testbench/core/PD2.primary' initialized to state 'HV' with simstate 'NORMAL'.
	LP_SS_ILLEGAL	WARNING	Supply Set 'tb/dut_top_inst/TOP_prim' initialized to illegal state 'TOP_HV' with simstate 'NORMAL'.
	LP_SS_STATE_CHANGE	INFO	Supply Set 'topunit/TOP_U/Island_V1.primary' transitioned to state 'DEFAULT_NORMAL' with simstate 'NORMAL'.

Context	ID	Severity	Sample Message
Supply Sets	LP_SS_STATE_CHANGE_ILLEGAL	WARNING	Supply Set 'tb/dut_top_inst/PD1_prim' transitioned to illegal state 'PD1_OFF' with simstate 'NORMAL'.
	LP_SS_EXPR_MISMATCH	ERROR	Supply expression's value '0' does not match with Logic expression value '1' for state 'OFF' of Supply Set 'tb/top_inst/SS_TOP'.
Supply Source	LP_PORT_STATE_CHANGE	INFO	Supply Source 'tb/top/PSW_C/vout' changed to 'HV' state with root net 'tb/top/PSW_C/vout' at state 'FULL_ON' and voltage 1.08 V.
UPF::supply_on/ supply_off	LP_SUPPLY_PAD_NOT_FOUND	WARNING	Supply Pad 'UUT/VDD_18' not found in 'UPF::supply_on' function. File: ./Source/Regr_Auto/SourceSink_mx/9000620215/.tb.sv, Line: 37.
	LP_SUPPLY_PAD_CALL	INFO	Supply Pad function 'UPF::supply_on' called for Supply Pad 'Vdd'. Supply Pad value changed to `'{FULL_ON, 1000000}`. File: ./Source/top.v, Line: 22.
	LP_SUPPLY_PAD_VOLTAGE	WARNING	Supply Pad function 'UPF::supply_on' called for Power Supply Pad 'V1' with '0' voltage. Supply Pad value changed to `'{FULL_ON, 0}`. File: ./Source/testbench.v, Line: 193.

Enabling Runtime Isolation Messages

By default, the following runtime isolation messages do not consider the state of the sink supply:

- LP_ISO_SEQFAIL1
- LP_ISOEN_INVALID
- LP_ISOPN_OFF
- LP_ISOGN_OFF

You can use the `-power=sinkbasedisoassert` compile-time option to avoid reporting of the above messages when all sink supplies/domains are OFF. If this option is used, then the above messages are issued only when at least one of the sink supply/domain is ON.

Verbose Illegal PST State Change Messages

During simulation, PST state changes are captured by LP automated messages. In case of a PST state transition to or from `ILLEGAL` state, it is helpful to know the contributing supplies that resulted in an `ILLEGAL` state.

You can use the `-power=pst_state_msg` compile-time option to enable verbose messaging for the following:

- Illegal state transitions (PST state changes to or from `ILLEGAL` state)
- Invalid combinations (Any valid combination of PST state supplies that is not defined in the PST table)

For example, consider following UPF code and PST:

```
# Power State Information:  
add_power_state SS_VDD1 \  
-state V_MIN {-supply_expr {power == `{{FULL_ON},0.7}}}  
add_power_state SS_VDD1 \  
-state V_NOM {-supply_expr {power == `{{FULL_ON},0.8}}}  
add_power_state SS_VDD1 \  
-state V_MAX {-supply_expr {power == `{{FULL_ON},0.9}}}  
add_power_state SS_VDD1 \  
-state V_OFF {-supply_expr {power == `{{OFF}}}}  
add_power_state SS_VDD1 \  
-state GND {-supply_expr {ground == `{{FULL_ON},0.00}}}  
add_power_state SS_VDD2 \  
-state V_MIN {-supply_expr {power == `{{FULL_ON},0.7}}};  
add_power_state SS_VDD2 \  
-state V_NOM {-supply_expr {power == `{{FULL_ON},0.8}}};  
add_power_state SS_VDD2 \  
-state V_MAX {-supply_expr {power == `{{FULL_ON},0.9}}};  
add_power_state SS_VDD2 \  
-state V_OFF {-supply_expr {power == `{{OFF}}}};  
  
#PST Details:  
create_pst blk_pst -supplies {SS_VDD1.power SS_VDD2.power  
SS_VDD1.ground};  
add_pst_state state0 -pst blk_pst -state {V_MIN V_MIN GND}  
add_pst_state state1 -pst blk_pst -state {V_MIN V_OFF GND}  
add_pst_state state2 -pst blk_pst -state {V_MAX V_OFF GND}  
add_pst_state state3 -pst blk_pst -state {V_OFF V_OFF GND}
```

Various possible cases and example LPA messages generated using the above PST are described below:

Case-1: For a given PST, if any of the supplies go into an ILLEGAL/UNDEFINED state, then that transition causes the PST to enter into ILLEGAL state.

In this case, VCS NLP flags LP_PST_STATE_ILLEGAL_INIT at time 0 and LP_PST_STATE_ILLEGAL_TRANSITION for later part of the simulation (non-time 0).

Example message at time 0:

```
[0] [WARNING] [LP_PST_STATE_ILLEGAL_INIT] Design started  
with 'ILLEGAL' state in pst 'test/top/blk_pst' with Supplies  
'SS_VDD1.power=off, SS_VDD2.power=illegal,  
SS_VDD1.ground=GND'.
```

Example message at non-time 0:

```
[55] [WARNING] [LP_PST_STATE_ILLEGAL_TRANSITION] Design  
state changed from 'ILLEGAL' to 'ILLEGAL' in pst 'test/top/  
blk_pst' with Supplies  
'SS_VDD1.power=illegal, SS_VDD2.power=V_NOM, SS_VDD1.ground=  
GND' and Supply 'SS_VDD1.power' changed.
```

Case-2: For a given PST, if all the supplies have legal values, but the combination of the supply state does not match any PST entry in UPF.

In this case, VCS NLP flags

LP_PST_STATE_ILLEGAL_COMBINATION_INIT at time 0 and
LP_PST_STATE_ILLEGAL_COMBINATION during later part of the simulation (non-time 0).

Example message at time 0:

```
[0] [ERROR] [LP_PST_STATE_ILLEGAL_COMBINATION_INIT] Design  
started with 'ILLEGAL' state in pst 'test/top/blk_pst' with Supplies  
'SS_VDD1.power=off, SS_VDD2.power=off,  
SS_VDD1.ground=GND'.
```

Example message at non-time 0:

```
[35] [ERROR] [LP_PST_STATE_ILLEGAL_COMBINATION] Design  
state changed from 'state0' to 'ILLEGAL' in pst 'test/top/  
blk_pst' with Supplies  
'SS_VDD1.power=V_MIN, SS_VDD2.power=V_NOM, SS_VDD1.ground=GN  
D' and Supply 'SS_VDD2.power' changed.
```

Case-3: For a given PST, all the supplies have legal values, and the combination of supply state also matches an entry in the PST.

In this case, VCS NLP flags LP_PST_STATE_INIT at time 0 and LP_PST_STATE_TRANSITION during later part of the simulation (non-time 0).

Example message at time 0:

```
[0] [INFO] [LP_PST_STATE_INIT] Design started with 'state3'  
state in pst 'test/top/bk_pst' with Supplies  
'SS_VDD1.power=off, SS_VDD2.power=off,  
SS_VDD1.ground=GND'.
```

Example message at non-time 0:

```
[15] [INFO] [LP_PST_STATE_TRANSITION] Design state changed  
from 'state2' to 'state1' in pst 'test/top/bk_pst' with  
Supplies  
'SS_VDD1.power=V_MIN,SS_VDD2.power=off,SS_VDD1.ground=GND'  
and Supply 'SS_VDD1.power' changed.
```

Note:

The severity of the LPA messages

LP_PST_STATE_ILLEGAL_COMBINATION_INIT and
LP_PST_STATE_ILLEGAL_COMBINATION is ERROR. These two
LPA messages imply that the simulation environment generated
a valid combination of supplies and this combination is not defined
in the PST. You can downgrade the severity to WARNING or INFO.
For more details on how to control severity of LPA messages,
refer to "[Override the Default Severity of Messages](#)".

Custom Assertion Checking for Clock/Reset During Shutdown

VCS NLP supports custom assertion checking for clock/reset inputs to power domain that do not toggle when the domain is OFF. You must use the following options at compile time to enable this feature:

`-power=assert_clk_during_shutdown`

This option allows custom assertion checking for clock inputs to power domain that do not toggle when the domain is OFF. When this feature is enabled, VCS NLP reports the first toggle of the clock or first time clock is toggled when the domain is OFF.

Following is the sample warning message:

```
[WARNING] [LP_CLOCK_WIGGLE] Clock 'CLK' wiggling in  
instance '/topunit/TOP_WRAPPER_U/TOP_U/U2  
(unit2.vhd:49)' during shutdown for the power domain  
'topunit/TOP_WRAPPER_U/U2_INST_UNIT1_PD'.
```

`-power=assert_async_during_shutdown`

This option allows custom assertion checking for reset inputs to power domain that do not toggle when the domain is OFF. When this feature is enabled, VCS NLP reports the first toggle of the reset or first time reset is toggled when the domain is OFF.

Following is the sample warning message:

```
[WARNING] [LP_RESET_WIGGLE] Reset 'reset' wiggling in  
instance 'testbench.top.fflop_inst' during shutdown for  
the power domain 'testbench/top/PD_FF'.
```

Supported UPF Query Commands

You can use the following UPF query commands to write custom low power assertions.

Note:

- The query commands are supported only in the Tcl file passed with `-lpa_bind`. VCS NLP generates an error message, if you use query commands in the UPF file.
- The names of the UPF/design objects passed to query commands must be specified using absolute paths.

query_power_domain

The following commands describe the usage of this command:

- `query_power_domain *`

Return Type: Tcl List

Return Data: Returns a list of all the power domains defined (full hierarchical path).

- `query_power_domain`
`<full_hierarchical_path_of_domain> -detailed`

Return Type: Tcl array, list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
domain_name	Full hierarchical name of the power domain.	SV string
elements	NA	NA
primary_ground_net	Full hierarchical path of the primary ground net.	UPF::supply_net_type/ SV string

Key	Value	Bind Type
primary_power_net	Full hierarchical path of the primary power net.	UPF::supply_net_type/ SV string
supply	NA	NA
sim_state	sim_state of the power domain.	UPF::power_state_simstate type/ SV string

NA - Unsupported Keys

query_supply_net

The following commands describe the usage of this command:

- `query_supply_net*`

Return Type: Tcl List

Return Data: Returns a list of all the supply nets defined in the UPF (full hierarchical path).

- `query_supply_net <net_name> -detailed`

Return Type: Tcl array, list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
domain	Full hierarchical name of the power domain.	SV string
is_supply	Returns 1 if the specified net_name is a supply port, else returns 0.	SV bit
net_name	Name of the supply net.	SV string
resolve	NA	NA

query_pst

The following commands describe the usage of this command:

- `query_pst *`

Return Type: Tcl List

Return Data: Returns a list of all the power state tables defined (full hierarchical path).

- `query_pst <pst_name> -detailed`

Return Type: Tcl array, list of {key value} pairs

Following is the output of the above command:

Key	Value	Bind Type
supplies	List of supply nets or ports of specified pst table.	UPF::supply_net_type/ SV string
table_name	Full hierarchical pst name.	SV string
design_state	Index of current state of the design. Integer index to the list of states returned by the <code>query_pst_state * -pst <pst_name></code> command. Illegal/undefined state is represented by index 0, so while populating the list of pst states in Tcl, the first state has to be manually added as “illegal” or any other user-defined name for illegal/ undefined state.	SV int

query_pst_state

The following commands describe the usage of this command:

- `query_pst_state * -pst <pst_name>`

Return Type: Tcl List

Return Data: Returns a list of all states defined for the specified power state table.

- `query_pst_state <state> -pst <pst_name> -detailed`

Return Type: Tcl list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
pst	Full hierarchical name of the pst	SV string
state	List of power states of the supplies for the specified pst_state	SV string
state_name	Name of the pst_state	SV string

query_power_switch

The following commands describe the usage of this command:

- `query_power_switch *`

Return Type: Tcl List

Return Data: Returns a list of all the power switches defined (full hierarchical path).

- `query_power_switch <switch_name> -detailed`

Return Type: Tcl list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
ack_delay	NA	NA
ack_port	NA	NA
control_port	{port_name net_name} net_name is returned in the name alias format. Use the 'query_original_name' tcl proc to get its simple name	{‘SV string’ ‘SV wire/SV string’}
domain	NA	NA
error_state	List of all error states	SV string
input_supply_port	{port_name supply_net_name}	{SV string UPF::supply_net_type}
off_state	List of all off states	SV string
on_partial_state	NA	NA
on_state	List of all on states	SV string

Key	Value	Bind Type
output_supply_port	{port_name supply_net_name}	{SV string UPF::supply_net_type}
states	{state_name state_type*} {state_name state_type*} {..}	{SV string SV string}
supply_set	NA	NA
switch_name	Full hierarchical name of the power switch	SV string
switch_state	Index that reflects the current state of the power switch. array_of_all_state_names[index]. Index 0 represents undetermined state. Array of all state_names must be generated from 'state_name' value of 'states' key.	SV int

* state_type is UNDETERMINED, ON, PARTIAL_ON, ERROR, OFF (in the same order) based on which of these are defined.

query_isolation

The following commands describe the usage of this command:

- query_isolation * -domain <ref_domain_name>

Return Type: Tcl List

Return Data: Returns a list of all the isolation strategies defined for the specified domain.

- `query_isolation <isolation strategy> -domain <full_hierarchical_path_of_ref_domain> -detailed`

Return Type: Tcl list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
clamp_value	Specified clamp value (0/1/Z/any/latch)	SV string
domain	Full hierarchical name of the power domain	
elements	NA	NA
force_isolation	NA	NA
isolation_ground_net	Name alias of isolation ground net. Use 'query_original_name' tcl proc to get the simple name of the isolation ground net.	UPF::supply_net_type/SV string
isolation_name	Name of the isolation strategy	SV string
isolation_power_net	Name alias of isolation power net. Use 'query_original_name' tcl proc to get the simple name of the isolation power net.	UPF::supply_net_type/SV string
isolation_signal	Name alias of isolation signal specified for the isolation strategy. Use 'query_original_name' tcl proc to get the simple name of the isolation signal.	SV wire/SV string
isolation_supply_set	NA	NA

Key	Value	Bind Type
location	Location specified for the isolation strategy (automatic self fanout parent)	SV string
no_isolation	Returns 1 if -no_isolation specified, else returns 0.	SV bit
sink_off_clamp	NA	NA
source_off_clamp	NA	NA
isolation_sense	Number representing the isolation sense specified in isolation strategy (0 - low, 1 - high)	SV bit
primary_power_net	Name alias of primary power net of the power domain. Use ‘query_original_name’ tcl proc to get the simple name of the primary power net	UPF::supply_net_type/SV string
primary_ground_net	Name alias of primary ground net of the power domain. Use ‘query_original_name’ tcl proc to get the simple name of the primary ground net.	UPF::supply_net_type/SV string

query_retention

The following commands describe the usage of this command:

- query_retention * -domain
<full_hierarchical_path_of_domain>

Return Type: Tcl List

Return Data: Returns a list of all the retention strategies defined for the specified power domain.

- `query_retention <retention strategy> -domain <full_hierarchical_path_of_domain> -detailed`

Return Type: Tcl list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
domain	Full hierarchical name of the power domain.	SV string
elements	NA	NA
exclude_elements	NA	NA
no_retention	NA	NA
output_related_supply_set	NA	NA
parameters	NA	NA
primary_power_net	Name alias of primary power net of the power domain. Use ‘query_original_name’ tcl proc to get the simple name of the primary power net.	UPF::supply_net_type/SV string
primary_ground_net	Name alias of primary ground net of the power domain. Use ‘query_original_name’ tcl proc to get the simple name of the primary ground net.	UPF::supply_net_type/SV string
restore_signal	{restore_signal sense} Restore_signal is returned in the name alias format. Use ‘query_original_name’ tcl proc to get its simple name. Sense is either high, low, posedge, or negedge.	{SV wire/string SV string}

Key	Value	Bind Type
restore_condition	Name alias of specified restore_condition. Returns 1 if restore_condition evaluates to true, else returns 0.	SV logic
retention_condition	NA	NA
retention_power_net	Name alias of retention power net. Use 'query_original_name' tcl proc to get the simple name of the retention power net.	UPF::supply_net_type/SV string
retention_ground_net	Name alias of retention ground net. Use 'query_original_name' tcl proc to get the simple name of the retention ground net.	UPF::supply_net_type/SV string
retention_name	Name of the retention strategy.	SV string
retention_supply_set	NA	NA
save_signal	{save_signal sense} save_signal is returned in the name alias format. Use 'query_original_name' tcl proc to get its simple name. Sense can be high, low, posedge, or negedge.	{SV wire/string SV string}
save_condition	Name alias of save_condition specified. Returns 1 if restore_condition evaluates to true, else returns 0.	SV logic
use_retention_as_primary	NA	NA

query_retention_control

The following commands describe the usage of this command:

- `query_retention_control * -domain <full_hierarchical_path_of_domain>`

Return Type: Tcl List

Return Data: Returns a list of all the retention strategies defined for the specified power domain.

- `query_retention_control <retention_strategy> -domain <domain> -detailed`

Return Type: Tcl list of {key value} pairs

Following is the output of the above command.

Key	Value	Bind Type
retention_name	Name of the retention strategy.	SV string
assert_r_mutex	{net_name sense} Sense is either high, low, posedge, or negedge.	{SV wire/SV string SV string}

Key	Value	Bind Type
assert_rs_mutex For more information, see “ assert_rs_mutex ” section.	1. Key not available if not defined. 2. If rs_mutex is default, then key is available with no value. 3. If logic net is specified, then {net_name sense} Sense is either high, low, posedge, or negedge.	1. NA 2. “” 3. {SV wire/SV string SV string}
assert_s_mutex	{net_name sense} Sense is either high, low, posedge, or negedge.	{SV wire/SV string SV string}
domain	Full hierarchical name of the power domain.	SV string
restore_signal	{restore_signal sense} restore_signal is returned in the name alias format, use ‘query_original_name’ tcl proc to get its simple name. Sense is either high, low, posedge, or negedge.	{SV wire/string SV string}
save_signal	{save_signal sense} save_signal is returned in the name alias format, use ‘query_original_name’ tcl proc to get its simple name. Sense is either high, low, posedge, or negedge.	{SV wire/string SV string}

Warning Messages for Unconnected Supply Ports

VCS NLP generates the following five different warning message tags for unconnected supply ports based on the direction and the connectivity of the port.

Case-1: Supply port is neither an input port nor an output port

```
Warning-[UPF_USP] Unconnected supply port  
save_restore_high.upf, 6  
Supply Port 'TOP/VDD' does not have a supply net connected  
either to its lowconn or highconn.
```

Case-2: Supply port is an input port

- **No Lowconn connection**

```
Warning-[UPF_ISPNL] Input Supply Port with no load  
save_restore_high.upf, 6  
Input Supply Port 'TOP/VDD' does not have a supply net  
load.  
Add a load by using 'connect_supply_net' to a net at the  
current/lower scope.
```

- **No Highconn connection**

```
Warning-[UPF_ISPND] Input Supply Port with no driver  
save_restore_high.upf, 6  
Input Supply Port 'TOP/VDD' does not have a supply net  
driver.  
Add a driver by using 'connect_supply_net' to a net at a  
higher scope.
```

Case-3: Supply port is an output port

- **No Lowconn connection**

```
Warning-[UPF_OSPND] Output Supply Port with no driver  
save_restore_high.upf, 6  
Output Supply Port 'TOP/VDD' does not have a supply net  
driver.  
Add a driver by using 'connect_supply_net' to a net at  
the current/lower scope.
```

- **No Highconn connection**

```
Warning-[UPF_OSPNL] Output Supply Port with no load
```

```
save_restore_high.upf, 6
Output Supply Port 'TOP/VDD' does not have a supply net
load.
Add a load by using 'connect_supply_net' to a net at a
higher scope.
```

Root Supply Driver

A root supply driver is of the following type:

- Undriven supply net
- Undriven supply port
- Output supply port of the power switch
- HDL/Spice driver to the UPF using `connect_supply_net`
- HDL Net - A net that is brought into the UPF using `create_supply_net -reuse`, the first time it is created in the UPF

Root supply drivers are those supply ports/nets which can inject supply state from either HDL/Spice drivers, power switch, or `supply_on/supply_off` calls.

A supply source can have multiple root supply drivers if at least one of the connected nets is resolved `parallel` or resolved `one_hot`. A supply source cannot have both resolve `one_hot` and resolve `parallel` nets.

TFIPC-L Lint Check in Presence of UPF Connections

To detect unconnected ports in the design, you can use the `+lint=TFIPC-L` compile-time option. This option flags the following lint message for the unconnected ports:

Lint-[TFIPC-L] Too few instance port connections

The above message is issued only for the ports that are unconnected in either design or UPF.

22

VCS NLP Quick Reference

This chapter provides information on all the low power command line options and UPF design attributes supported by VCS NLP. This chapter includes the following sections:

- [Command Line Options](#)
- [Design Attributes to be Specified in UPF](#)
- [HDL Attributes](#)

Command Line Options

- [Compile-Time Options](#)
- [Simulation Option](#)

Compile-Time Options

This section describes all the compile-time command line options supported by VCS NLP.

Table 22-1 Compile-time Options

Compile-Time Option	Description
+lint=MV_DB_CELL_MISMATCH	Generates a warning message when there is a mismatch between the Verilog definition and liberty definition of a cell. For more information, see “Lint Check for Mismatch Between Verilog and Liberty Definitions of a Cell” section .
+lint=TFIPC-L	Allows you to detect unconnected ports in the design. For more information, see “TFIPC-L Lint Check in Presence of UPF Connections” section .
-lpa_bind <filename>	Allows you to specify the custom bind Tcl file. For more information, see “Custom Assertions” section .

Compile-Time Option	Description
-lpa_bind_filelist<file_list>	Allows you to specify a file list listing multiple custom bind checker Tcl files. For more information, see “ Custom Assertions ” section.
-mvrpt	Allows you to redirect or rename the compile static reports directory. For more information, see “ Redirecting/Renaming mvsim_native_reports Directory ” section.
-pa_random_corrupt	Enables the random corruption feature. For more information, see “ Random Corruption ” section.
-power_config <config file>	Allows you to specify the compile-time configuration file that specifies the search_path and link_library for the liberty (.db) files. For more information, see “ Library Mapping ” section.
-power_top <module name>	In VCS NLP, you can specify power top either in UPF using the set_design_top command with the full path, or on VCS command-line using the -power_top option.

List of -power Compile-Time Options

This topic describes all the -power compile-time command-line options supported by VCS NLP. Following is the syntax of -power compile-time option:

```
%vcs -power=<option1> -power=<option2>... [other_options]
```

Or,

```
%vcs -power=<option1>+<option2>... [other_options]
```

[Table 22-2](#) describes the supported options.

Table 22-2 List of -power Compilation Options

Option	Description
accurate	Skips the insertion of the virtual isolation cells. This option is useful when the isolation cells are pre-inserted in the design, and isolation strategies are defined in the UPF. For more information, see “ Association of Pre-Inserted Isolation Cells ” section.
attributes_on	Enables the following attributes which are used to control the corruption and wake-up semantics of the initial and always blocks: vcs_reinit, vcs_dont_reinit, vcs_always_on, vcs_dont_trigger_on_wakeup For more information, see “ Controlling the Corruption and Wake-up Semantics of the Initial and Always Blocks ” section.
apfcompat	Skips the corruption of reals, constants, and the signals assigned inside the initial blocks. For more information, see “ Skipping Corruption ” section.
auto_complete	Enables the auto-completion for the UPF commands and command options. For more information, see “ Support for Auto-Completion of UPF Commands ” section.
ackport_driver_check	Allows you to check for the drivers of the ack_port signal in the entire design hierarchy and reports the UPF_MDOAPS warning message (with driver information), if there is a driver that can contribute to a value on the ack_port signal present in the design. For more information, see “ Using create_supply_net Command ” section.

Option	Description
assert_clk_during_shutdown	Allows custom assertion checking for clock inputs to power domain that do not toggle when the domain is OFF. For more information, see “Custom Assertion Checking for Clock/Reset During Shutdown” section .
assert_async_during_shutdown	Allows custom assertion checking for reset inputs to power domain that do not toggle when the domain is OFF. For more information, see “Custom Assertion Checking for Clock/Reset During Shutdown” section .
assert_reset_sequence	Enables power-on reset assertions. For more information, see “Power-on Reset Assertions Support” section .
check_power_down_write	Generates the runtime warning messages when an attempt to write on a power domain element is detected, and the domain is in CORRUPT simstate. For more information, see “Reporting Write Attempt on OFF Domain Element” section .
clamp_toggle_check_on_iso_enable	Adds an assertion that checks for a stable value on output of an isolation cell only when isolation is enabled. For more information, see “Implementing Assertion Checks for Isolation Cells” section .
clamp_toggle_check_on_iso_disable	Adds an assertion that checks for a stable value on output of an isolation cell only when isolation is disabled. For more information, see “Implementing Assertion Checks for Isolation Cells” section .
clamp_toggle_check	Assertion check happens when isolation is enabled/disabled. For more information, see “Implementing Assertion Checks for Isolation Cells” section .

Option	Description
coverage	Creates covergroups for low power objects based on the power intent (UPF). For more information, see “Creating Covergroups for the Low Power Objects” section .
cov_pd_simstate	Enables the power domain simstate coverage. For more information, see “Enabling Coverage on Low Power Objects” section .
cov_port_state	Enables port state coverage. For more information, see “Enabling Coverage on Low Power Objects” section .
cov_implicit_supply_set	Enables coverage on the implicit supply sets only (implicit supply set simstate and power state). For more information, see “Enabling Coverage on Low Power Objects” section .
cov_explicit_supply_set	Enables coverage on the explicit supply sets only (explicit supply set simstate and power state). For more information, see “Enabling Coverage on Low Power Objects” section .
cov_supply_set	Enables all types of supply set coverage. For more information, see “Enabling Coverage on Low Power Objects” section .
cov_psw	<p>Enables power switch coverage for the following low power objects:</p> <ul style="list-style-type: none"> • Power switch state (on/off/ partial_on) • ACK port • Control port <p>For more information, see “Enabling Coverage on Low Power Objects” section.</p>

Option	Description
cov_pst_state	Enables only the state coverage of PST. For more information, see “ Enabling Coverage on Low Power Objects ” section.
cov_pst_transition	Enables only transition coverage of PST. For more information, see “ Enabling Coverage on Low Power Objects ” section.
cov_pst	Enables Power State Table (PST) coverage. This option enables both the state and transition coverage for PST. For more information, see “ Enabling Coverage on Low Power Objects ” section.
cov_ret	Enables coverage on the retention save and restore signals. For more information, see “ Enabling Coverage on Low Power Objects ” section.
cov_iso	Enables coverage on the isolation enable signal. For more information, see “ Enabling Coverage on Low Power Objects ” section.
dont_touch_mem	Prevents the corruption of memories that are initialized using \$readmem* and have no other structural drivers. For more information, see “ Preventing Corruption of Memories During Power Shutdown ” section.
dcccompat_gen_inst_name_match	Treats the generate instance name (any depth) as a single token for matching wildcard character “*” specified in the UPF. For more information, see “ Treating Generate Instance as a Single Token for Wildcard Matching ” section.

Option	Description
dont_isolate_floating_nets	Skips the isolation on the undriven/floating boundary ports. For more information, see “Skipping Isolation on the Undriven/Floating Ports” section .
dump_hvp	Allows you to dump the coverage database to be used with the -lpcov URG option to generate low power coverage reports. For more information, see “Low Power Coverage Report View in URG” section .
dump_pd_tchk_rpt	Allows you to capture the information related to the timing checks on elements of a power domain in the pd_tcheck_mapping.rpt file. For more information, see “Support for Timing Simulation With SDF in Presence of UPF” section .
force_priority	Overrides corruption with the HDL force. For more information, see “Overriding Corruption with HDL Forces” section .
ignore_latch_retention	Ignores latches from retention. For more information, see “Ignoring Latches From Retention” section .
ignore_redundant_constraint	Ignores SRSN/SPA buffers specified on all intermediate boundary ports. For more information, see “Ignoring SRSN/SPA Buffers Specified on Intermediate Boundary Ports” section .
pass_thru_const	Skips the corruption on constants. For more information, see “Handling of Constants for Corruption and Isolation” section .

Option	Description
portconnect_exprs	Enables isolation/corruption on the logical expressions in the port map. By default, isolation/corruption is not enabled on the logical expressions in the port map. For more information, see “Isolation/Corruption on Ports with Logical Expressions in Port Map” section.
pst_state_msg	<p>Use this option to enable verbose messaging for the following:</p> <ul style="list-style-type: none"> • Illegal state transitions (PST state changes to or from <code>ILLEGAL</code> state) • Invalid combinations (Any valid combination of PST state supplies that is not defined in the PST table) <p>For more information, see “Verbose Illegal PST State Change Messages” section.</p>
report_port_no_toggle	Allows you to check toggle on the crossover ports when the supply source changes its state from power-down to power-up (<code>FULL_ON</code>) until the end of the simulation. For more information, see “Identifying the Crossover Ports That do not Toggle” section.
rtlpg	Enables VCS NLP to automatically match the supply nets or ports created in UPF with the nets or ports in RTL within the same scope. For more information, see “Design Namespace Conflict Resolution for the Creation of Supply Ports/Nets” section.

Option	Description
src_off_clamp_check	Enables automated check at runtime for an assertion of isolation enable when the source supply is OFF. For more information, see “ Checking for Assertion of Isolation Enable When Source Supply is OFF ” section.
UPF_dont_touch_BC	Enables the backwards compatibility when the UPF_dont_touch attribute is used in UPF. For more information, see “ Using UPF_dont_touch Attribute ” section.
unconnected_bias_pins_on	Keeps the unconnected bias PG pins of cells ON. For more information, see “ Implicit Supply Net Connections ” section.
upf_lint	Generates a warning message for every power domain without explicit specification of power or ground nets in the UPF. For more information, see “ Lint Check for Explicit Primary Supplies for a Domain ” section.
upf_protect1	Automatically encrypt all the UPF files, including the UPF files specified with the load_upf command. For more information, see “ UPF Encryption Using upf_protect1 Option ” section.
undriven_supplies_off	Allows you to initialize all the supply ports (except supply pads) and supply set functions with no explicit nets to OFF state. For more information, see “ Supply Network Initialization ” section.

Option	Description
verbose1	Allows the tool to display the boundary_port_partitioning.rpt file. This file shows the repeaters implemented and its associated supplies. For more information, see “ Using the –repeater_supply Option ” section.
voltage_regulator_cell	Allows you to specify default ON voltage for internal PG pins of a macro. For more information, see “ Specifying Default ON Voltage for Internal PG Pins of a Macro ” section.
write_mvinfo	Allows you to generate the virtual isolation insertion report (isolation_insertion_info.txt) / virtual isolation inference report (new_isolation_association.rpt). For more information, see “ Generating Virtual Isolation Reports ” section.

Simulation Option

[Table 22-3](#) describes the simulation option supported by VCS NLP.

Table 22-3 Simulation Option Quick Reference

Runtime Option	Description
<code>-power <config_file_name></code>	<p>Allows you to specify the runtime configuration file. You can specify the runtime design attributes and Unified Message Control options in the runtime configuration file.</p> <p>Example: % simv -power power_config.tcl</p> <p>Example of power_config.tcl file:</p> <pre>set_design_attributes -elements \ {power_pa_tb/noRandomCorruption} \ -attribute SNPS_random_corruption 01X</pre>

Table 22-4 lists the runtime design attributes supported by VCS NLP. You must specify these attributes in the configuration file you specify with the `-power <config_file_name>` runtime option.

Table 22-4 Runtime Design Attributes Quick Reference

Attribute	Value	Description
SNPS_apf_compat	TRUE	Skips the corruption of VHDL variables, real and integer types. For more information, see “ Skipping Corruption ” section.
SNPS_disable_cdc_corruption		For a given power domain, you can use this attribute in the compile time configuration file to specify the list of isolation strategies on which you want to disable corruption. For more information, see “ Corrupting Isolation Cell Output on Assertion/De-assertion of Isolation Enable ” section.
SNPS_keep_unassociated_iso_on	TRUE	Treats the unassociated ISO/ELS cells as always-on. For more information, see “ Association of Pre-Inserted Isolation Cells ” section.
SNPS_random_seed	value	Allows you to specify a seed using the <code>set_design_attributes</code> command. A seed is a number that initializes a pseudo-random generator used for corruption during shutdown. For more information, see “ Random Corruption ” section.
SNPS_random_corruption	value	Runtime attribute. Allows you to set the corruption value to any of 0/1/X/Z during shutdown. For more information, see “ Random Corruption ” section.
SNPS_treat_undriven_as_off		Initializes all the supply ports (except supply pads) and supply set functions with no explicit nets to OFF state. For more information, see “ Supply Network Initialization ” section.

Attribute	Value	Description
SNPS_treat_undriven_as_undetermined		Initializes all the undriven supply ports/nets and supply set functions with no explicit nets to UNDETERMINED state. For more information, see “ Supply Network Initialization ” section.
SNPS_treat_undriven_supply_nets_as_undetermined		Initializes all the undriven supply net to UNDETERMINED state. For more information, see “ Supply Network Initialization ” section.
SNPS_voltage_based_supply_on	TRUE	Turn OFF the supply when <code>supply_on</code> is called with 0 volts. For example: <code>supply_on ("VDD", 0.0)</code> . For more information, see “ Voltage-Based supply_on ” section.

Design Attributes to be Specified in UPF

[Table 22-5](#) lists the compile-time design attributes supported by VCS NLP. You must specify these compile-time design attributes in the UPF.

Table 22-5 Compile-Time Design Attributes Quick Reference

Attribute	Value	Description
conservative_diff_supply_only_isolation	TRUE	Allows you to insert an isolation cell for the -diff_supply_only isolation strategies in the cases where one or more of the loads/sinks might have the same supply as the driver/source, but at least one of them uses a different supply. Specify the following in the UPF. For more information, see “Support for Conservative diff_supply_only Isolation” section.
deselect_coverage_object	value	Allows you to specify the low power objects that you want to exclude from the coverage collection. For more information, see “Excluding UPF Objects from Low Power Coverage” section.
enable_bias	TRUE / FALSE	When set to TRUE, enables the bias feature. Transitive effect is applied on child blocks under top bias block. For more information, see “Enabling Bias Feature” section.
hetero_fanout_isolation	true	Allows you to insert isolation cell in the extent of the domain of the self or parent hierarchy/instance. For more information, see “Path-Based Isolation Support for Heterogeneous Loads” section.

Attribute	Value	Description
is_soi	TRUE	Enables FDSOI. For FDSOI, both pwell and nwell are tied to the ground supply. To enable FDSOI, set the <code>is_soi</code> design attribute to TRUE in UPF, as shown below: <pre>set_design_attributes -models <model_name> -attribute is_soi TRUE</pre> For more information, see “ Implicit Supply Net Connections ” section.
lower_domain_boundary	TRUE/ FALSE	When set to TRUE, extends the definition of a power domain boundary to include the lower domain boundary ports/pins for isolation. For more information, see “ Supporting Lower Domain Boundary for Isolation ” section.
repeater_ground_net	supply_net_ground	Allows you to specify ground net. For more information, see “ Using the -repeater_supply Option ” section.
repeater_power_net	supply_net_power	Allows you to specify power net. For more information, see “ Using the -repeater_supply Option ” section.
SNPS_dont_reinit	TRUE/ FALSE	When set to TRUE, enables <code>dont_reinit</code> property on the initial blocks targeted by the command. For more information, see “ Initial Block Retriggering ” section.
SNPS_default_power_upf2sv_vct	vct_name	Allows you to change the default VCT for power port connection.
SNPS_default_power_upf2vh_vct	vct_name	
SNPS_default_power_sv2upf_vct	vct_name	
SNPS_default_ground_upf2sv_vct	vct_name	Allows you to change the default VCT for ground port connection.

Attribute	Value	Description
SNPS_default_ground_upf2vh_vct	vct_name	
SNPS_default_ground_sv2upf_vct	vct_name	
SNPS_default_supply_net_state	STATE_NAME	Allows you to set default state on the supply port. For more information, see “ Setting Supply Port Default Value ” section.
SNPS_force_semantics	CONFIG NAME	Allows you to configure semantics for runtime forces during power shutdown and wake-up. For more information, see “ Configuring Semantics for Runtime Forces During Power Shutdown and Wake-up ” section.
SNPS_logic_expr_drives_supply_expr	TRUE/ FALSE	Drives the supply set function which is either undriven or has a single undriven root supply driver based on the logic expression. For more information, see “ Driving Supply Expression Using Logic Expression ” section.
SNPS_macro_port_based_corruption	ENABLE/ DISABLE	Enable/disable the port-based corruption on a particular macro or a full design. For more information, see “ Configuring Port-Based Corruption on Macros ” section.
SNPS_reinit	TRUE/ FALSE	When set to TRUE, enables re-triggering of the initial blocks on wake-up. For more information, see “ Initial Block Retriggering ” section.

Attribute	Value	Description
SNPS_real_number_corruption	real_value	Allows you to set a unique value for real data types under corruption. For more information, see “ Setting User-Defined Values for the Corrupted Real Data Type Signals ” section.
terminal_boundary		Use the terminal_boundary design attribute to mark hierarchical boundaries as terminal points for isolation strategies. For more information, see “ Support for Isolation Terminal Points at the Hierarchical Boundaries ” section.
UPF_dont_touch	TRUE	Excludes certain portions of the design from being power managed. For more information, see “ Using UPF_dont_touch Attribute ” section.

HDL Attributes

Table 22-6 lists the HDL attributes used to control the corruption and wake-up semantics of the initial and always blocks in Verilog. For more information, see “[Controlling the Corruption and Wake-up Semantics of the Initial and Always Blocks](#)” section. You must specify these attributes in the design code.

Table 22-6 HDL Attributes to Control Corruption and Wake-up Semantics

Attribute	Description
(* vcs_reinit *)	Allows you to retrigger the selected initial blocks on power-up without using the UPF commands.
(* vcs_dont_reinit *)	Disables the retriggering of the selected initial block after power-up.
(* vcs_always_on *)	Marks the selected always block as always-on.
(* vcs_dont_trigger_on_wakeup *)	Disables the automatic retriggering of the selected always block on power-up.

Index

A

add_port_state 3-21, 3-148, 3-149
add_power_state 3-21, 3-59
associate_supply_set 3-5
-attribute name value 6-15

B

bind_checker 3-38

C

connect_logic_net 3-24
connect_supply_net 3-8, 16-16, 17-3
-covg_disable_cg=ALL 14-6
-covg_disable_cg=LP 14-6
-covg_disable_cg=USER 14-6
create_hdl2upf_vct 16-16
create_logic_net 3-23
create_logic_port 3-23
create_power_domain 3-6
create_power_state_group 3-22
create_power_switch 3-6
create_pst 3-20
create_supply_net 3-8, 3-84
create_supply_port 3-7

create_upf2hdl_vct 16-16
CSN 16-9

D

describe_state_transition 3-40
-diff_supply_only 9-20
DISABLE_INSTANCE_WITH_ELEM 9-34
-domain 3-59
don't reinit 6-16
-dump full_exclusions group 14-32
dump_pd_tchk_rpt 16-33

E

-elements 6-14
ELS 16-10
enable_state_propagation_in_add_power_state 5-10
-exclude_elements 9-29

I

ISO 16-10
isolation_ground_net 16-10
isolation_power_net 16-10

L

load_simstate_behavior 3-41
load_upf 3-33
load_upf_protected 3-41
-ipa_bind_filelist 12-11
-lpcov 14-20
lphvp.LP.html 14-21

M

map_isolation_cell 3-15, 9-60
map_level_shifter_cell 3-12
map_power_switch 3-9
map_retention_cell 3-19
merge_power_domains 3-36
-models model_list 6-14
mv_upf_strategy_in_inst_name 9-10

N

name_format 3-12
NLP 8-4
-no_isolation 3-149

P

-pa_random_corrupt 8-4
package_name
 resolution_name 3-84
Power Domain Detail page 14-21
Power Domain detail page 14-23
Power Scope Detail page 14-21, 14-22
-power=write_mvinfo 9-34, 9-36

Q

query_cell_instances 3-26
query_cell_mapped 3-26, 3-27
query_design_attributes 3-29
query_hdl2upf_vct 3-30

query_net_ports 3-27
query_port_net 3-27
query_power_domain 3-30
query_power_switch 3-28
query_pst 3-28
query_pst_state 3-28
query_retention 3-31
query_retention_control 3-31
query_supply_net 3-31
query_upf 3-29

R

reate_composite_domain 3-38
reinit 6-14
retention_element_list 11-36
retention_list_name 11-33

S

set_design_attributes 3-34
set_design_top 3-32, 3-42
set_domain_supply_net 3-10
set_equivalent 3-42
set_isolation 3-14
set_isolation_control 3-15
set_level_shifter 3-11
set_partial_on_translation 3-32
set_port_attributes 3-34, 3-132
set_power_switch 3-36
set_retention 3-17, 11-33
set_retention_control 3-18
set_retention_elements 3-19, 11-32
set_scope 3-33
set_simstate_behavior 3-43
-sink 9-20
SNPS_dont_reinit 6-15
SNPS_random_seed 8-5
SNPS_reinit 6-15
-source 9-20
-supply_expr 3-61

`supply_net_type.state` [3-39](#), [16-17](#)

`UPF` [16-9](#), [17-23](#)
`upf_version` [3-44](#), [3-146](#)

T

-transitive [6-15](#)

U

-unconnected [3-132](#)

-update [9-29](#)

V

-vct [16-20](#)

W

`wreal netlist` [17-3](#)