

MVSIM Native Mode User Guide

G-2012.09
September 2012

Comments?
E-mail your comments about this manual to:
vcs_support@synopsys.com.

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2011 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSiM, HSPIICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia, Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSiMplus, HSPIICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

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

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

ARM and AMBA are registered trademarks of ARM Limited.

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

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

Contents

1. MVSIM Native Mode

Overview	2
Understanding Low Power Design Concepts	3
Low Power Concepts	3
Low Power Techniques	6
Specifying Power Intent of the Design Using UPF	9
IEEE Standard 1801 Commands Supported by MVSIM Native Mode	
9	
UPF Supply Package	16
Using Low Power in VCS	18
Power Techniques in VCS	19
HDL Constructs and Techniques in VCS	20
Corruption Semantics	21
Modeling the Delay Specification for a Power Switch	23
Random Corruption	23
UPF_dont_touch Support	26
Setting Supply Port Default Value	28

Initial Block Retriggering	29
Virtual Isolation Insertion Report	34
MVSIM Native Mode Limitations	35
UPF Commands	36
Using set_related_supply_net Command	36
Using set_port_attributes (SPA) Command	38
Using the query_cell_mapped Command	41
Support for -resolve_parallel Option.....	42
Wildcard Support in UPF Commands	42
Using upf_tool Variable	45
Unified Low Power Messaging	47
Assertions Related Clock/Reset Wiggle During Shutdown/Standby	
53	
Supply-Set-Based Power Intent Specification Support	55
Using set_isolation Command.....	59
Using add_power_state Command.....	66
Using associate_supply_set Command	70
Controlling supply_set States Using set_supply_set_state ...	71
CORRUPT_ON_ACTIVITY Simstate (Low-Vdd Standby)	74
Power Aware Verification Environment	76
Low Power Assertions in UPF Flow.....	76
Automated Assertions	76
Assertion Control Options	78
Custom Assertions.....	85
Debugging Low Power Designs Using DVE	93

Dumping Low Power Data to VPD.....	94
Low Power Debug Features	95
Coverage Support for Low Power Objects	127
Guidelines for Migrating to MVSIM Native Mode from MVSIM ...	139
Differences between MVSIM and MVSIM Native Mode	146
Gate-level Netlist Simulations (Simulating with Liberty (.db) Files)	147
Using Connect Supply Net and Library Mapping.....	147
Supported .db Attributes	149
Power Aware and Non-Power Aware Models	150
Simulation Behavior for Power Aware Models.....	151
Simulation Behavior for Non-Power Aware Models.....	151
Implicit Supply Net Connections	152
UPF2SV/SV2UPF Connections.....	153
Value Conversion Tables (VCTs)	154
Multi-rail Macro Simulation.....	158
Limitations of Gate-level Netlist Simulations	162
Synopsys Low Power Flow Consistency.....	163
-simulation_only Support.....	164
Synopsys Low Power Flow Consistency Check	165
Handling of Constants for Corruption and Isolation.....	166
Error Checking for Isolation Strategy with -location parent on Top-level Ports.....	168
Error for Conflicting Port State Names in the add_port_state Command.....	169
Appendix	171

Low Power Assertions	171
Using <code>bind_checker</code> Command	204

1

MVSIM Native Mode

This document describes basic elements of low power design, UPF support, and use model of MVSIM low power technology that enables power-aware verification.

This document contains the following sections:

- [“Overview” on page 2](#)
- [“Understanding Low Power Design Concepts” on page 3](#)
- [“Specifying Power Intent of the Design Using UPF” on page 9](#)
- [“Using Low Power in VCS” on page 18](#)
- [“UPF Commands” on page 36](#)
- [“Unified Low Power Messaging” on page 47](#)
- [“Supply-Set-Based Power Intent Specification Support” on page 55](#)

- [“CORRUPT_ON_ACTIVITY Simstate \(Low-Vdd Standby\)” on page 74](#)
- [“Power Aware Verification Environment” on page 76](#)
- [“Debugging Low Power Designs Using DVE” on page 93](#)
- [“Coverage Support for Low Power Objects” on page 127](#)
- [“Guidelines for Migrating to MVSIM Native Mode from MVSIM” on page 139](#)
- [“Gate-level Netlist Simulations \(Simulating with Liberty \(.db\) Files\)” on page 147](#)
- [“Synopsys Low Power Flow Consistency”](#)
- [“Appendix” on page 171](#)

Overview

Power management is a big challenge for SoC designs. As more transistors are integrated onto a single chip, the power density goes up drastically. Excessive power consumption can overheat and eventually burn out the entire chip.

Similarly, even though a higher voltage supply can make devices work faster, this approach is not followed as the power consumption also increases. These factors, in turn, reduce the already limited battery life drastically.

Therefore, balancing power and performance has become a critical design requirement. MVSIM native mode provides basic low power simulation capability in a UPF-based flow using power-gating and retention techniques.

Understanding Low Power Design Concepts

This section describes the various low power management concepts and techniques.

Low Power Concepts

Following are some commonly used terms for low power designs.

Rails

A net connected to a voltage source, possibly through some switching mechanism.

Functional Rails

Rails that drive logic values actively on signals, and are responsible for supplying or charging or discharging current to the cells output. These rails are the VDD/GND connections of a cell.

Non-functional Rails

Rails that control the behavior of the cell, but do not actively supply current for charge or discharge of the cell's output. Sleep, retention, and body-bias rails are examples of non-functional rails.

Voltage Rails

A single voltage provided by a power supply unit. This net does not have a logic value associated with it.

Power/Ground

The power and ground rails of a CMOS cell, similar to the one used in non-multi-voltage cells.

Sleep

A rail used to apply a positive zero, or negative gate-source bias voltage (V_{gs}) to the header/footer transistor, to cut off the (leakage) current.

Retention

An auxiliary rail used to power up a shadow sequential element, so that the positive supply voltage (V_{DD}) to the cell can be turned off.

Back-Bias

A rail applied to the bulk connection to control the threshold voltage (V_t). This rail can be either forward or reverse biased.

There are two types of back bias -- BBN (for NMOS transistors), and BBP (for PMOS transistors).

MV Partitions

Multi-voltage designs are partitioned into regions (such as power domains) depending on their voltage supply associations.

Power Domains

Domain is the drain of the driver. Domain indicates which rail is driving the signal.

MV States

The use of multiple supplies and different ways to control voltages results in the existence of different voltage states in a chip at any given point in time. For example, to reduce leakage power in certain modes of operation, parts of the chip may be in the shutdown state. Similarly, to reduce the dynamic power, another part can go into a low-VDD standby state. A retention state may also be deployed to save states during shutdown.

Shutdown

This is a state where the VDD to a domain is turned OFF.

Stand-by

Standby is a low power state, where a quick wakeup is expected. State retention in the memory elements is essential in this mode. Typically, clocks are gated, but Phased Locked Loops (PLLs) are not disabled. However, there could be multiple grades of Standby, turning off more circuitry as time passes. The overall Standby scheme is one that involves a process of gradually turning off/on clocks, PLLs, and voltages. Following are various techniques used for stand-by:

Clock-gated Stand-by

In this mode, there is no VDD control exercised. The clock network is gated. The PLL may be on/off depending on the resume time desired. While there is no VDD control, this mode serves as a preparation for it. Further, it is common for some domains to be shutdown while others are in this state. It is also common to apply a Back Bias to lower leakage in this situation.

Low VDD Standby

This is a state where the VDD is lowered to just enough value, so that memory elements do not lose state. Inevitably, the clock has to be gated down for this situation.

Scan Based Standby

A state where the scan chain is used to stream out state and then take the power domain to shutdown. The scan chain is used to reload the state on wakeup.

Low Power Techniques

Following are the low power techniques for SoC designs:

- Power Gating
- Multiple Supply Voltages (Multi- V_{DD})
- Voltage Scaling
- Multi-Threshold CMOS (Multi- V_T)
- Adaptive Body-Biasing

Power Gating

Power gating involves temporarily shutting down blocks in a design when the blocks are not in use. This would reduce the leakage power of the chip. Power gating with state retention involves switching off an area of a design when its functionality is not required, then restoring power when it is required.

Shutdown

In this technique, the power supply of the entire design is cut off when the circuit is not in use. Such designs do not require data to be retained in the registers or latches used in the design. Functional verification of design is still required to make sure that the position of designs that are awake function properly and also to ensure that the system would work when power is restored in the sleeping part of the device.

Isolation

When the power is shut off, each power domain must be isolated from rest of the design, so that it does not corrupt the downstream logic. Power shutdown results in slow output from the power gated blocks. These output spends significant time at threshold voltage, causing large crowbar currents in the always on blocks.

Isolation cells are used to prevent these crowbar currents. The isolation cells are placed between the outputs of the power gated blocks and inputs of the always on blocks.

Retention

In this technique, retention memory elements are used to prevent data from being lost in a specific power domain during power off. flip-flops and latches are used to save state information before the power is switched off and restore it when the power is turned back on.

Multi- V_{DD}

In this technique, critical paths and blocks in the design are given access to maximum voltage for the process and specification, but voltage is reduced for those blocks that need less power.

Voltage Scaling

In this technique, the supply voltage and clock frequency is reduced based on workload that results in a quarter of the power consumption. Voltage scaling techniques can be of following types:

- Dynamic - Here, a larger number of voltage levels are dynamically switched to follow changing workloads.
- Adaptive - Here, a closed loop feedback system is used to adjust the voltage.

Reducing the supply voltage has a detrimental effect on performance.

Adaptive Body-Biasing (ABB)

This technique is used to control leakage current during stand-by and active mode. Reverse body bias for stand-by mode and Zero body bias for active mode. ABB method reduces the leakage current exponentially. When body-to-source junction is reversed biased, the voltage threshold increases, thus reducing the leakage.

Multi- V_T

In this technique, a low-threshold-voltage library is used for a first pass through synthesis to get maximum performance and meet timing goals. The critical paths in the design are then determined, that is the path or paths in the design that require the highest performance. Then areas that do not require low-threshold-voltage cells are located, and low-voltage cells are swapped for high-voltage cells to reduce overall power and leakage of the design.

Specifying Power Intent of the Design Using UPF

MVSIM native mode supports the industry-standard Unified Power Format (UPF) 1.0 and 2.0 subset that are used to capture low power design requirements. UPF is a standard set of Tcl-like commands used to specify the power intent of the design. Using UPF commands, you can specify the supply network, switches, isolation, retention, and other aspects pertinent to the power management of the design. This single set of commands can be used for verification, analysis, and implementation of the design.

The UPF version 1.0 and 2.0 standards are available as a free download from www.accellera.org.

IEEE Standard 1801 Commands Supported by MVSIM Native Mode

Following table lists the commands and switches of IEEE Std 1801 supported by MVSIM native mode. For more information about the syntax of these commands, see the United Power Format Standard available in www.accellera.org.

Support for IEEE Std 1801 Commands

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
add_port_state	-state	Adds state information to a supply port. Limitations: You cannot use <code>add_port_state</code> on an RTL port tied to a supply net in UPF. You can only use this command on the supply ports explicitly created in UPF using <code>create_supply_port</code> .
add_power_state	-state	Attributes an object with a power state definition.
	-logic_expr	
	-supply_expr	
connect_supply_net	-ports Vport -ports [tb/moda/insta/v1 tb/moda/insta/v2 V1 and V2 are explicit ports of the module.	Connects a supply net to the supply ports. The net is propagated through implicitly created ports and nets throughout the logic hierarchy. Example: <code>connect_supply_net Vnet -ports Vport</code>
create_supply_net	-domain	Creates a supply net.
	-reuse	
	-resolve <unresolved parallel>	
create_supply_set	-function	Creates a supply set.
	-reference_gnd supply_net_name	
	-update	
create_supply_port	-domain	Creates a supply port on a design element.
	-direction	

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
create_power_domain	-elements	Defines a set of design elements that share a common primary supply set.
	-include_scope	
	-scope	
	-supply	
create_power_switch	-domain	Defines a switch.
	-output_supply_port	
	-input_supply_port	
	-control_port	
	-ack_port	
	-ack_delay	
	-on_state	
	-off_state	
	-on_partial_state	
	-error_state	
create_logic_net		Creates a logic net in the active scope.
create_logic_port		Creates a logic port on the active scope.
connect_logic_net		Connects a logic net to the specified ports.
find_objects [*]	find_objects	
load_upf	-scope	Sets the scope to the specified instance and executes the specified UPF commands.
name_format [*]	-isolation_prefix	Allows user guidance in constructing port and signal names related to isolation or level shifter cells.

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
	-isolation_suffix	Note: As per UPF, the isolated net/port must be suffixed with <code>_UPF_ISO</code> by default. In case of back-to-back isolation on the same net/port, there will be two isolation cells inserted, and hence the suffixes are <code>_UPF_ISO_0</code> and <code>_UPF_ISO</code> .
	-level_shift_prefix	
	-level_shift_suffix	
query_cell_mapped		Identifies the cell that is used for the named instance <code>instance_name</code> . For more information, see “Using the query_cell_mapped Command” .
set_design_top [*]		Full hierarchical name of the instance from where UPF has to be applied. This also needs to be the first command in the UPF, if it is present.
set_domain_supply_net	-primary_power_net	Sets the default power and ground supply nets for a power domain.
	-primary_ground_net	
set_design_attributes	-models	Sets the specified attributes for models or design elements.
	-attribute	
set_isolation	-domain	Defines an isolation strategy.
	-isolation_power_net	
	-isolation_ground_net	
	-isolation_supply_set	
	-no_isolation	
	-clamp_value	
	-applies_to	

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
	-elements	
	-source	
	-sink	
set_isolation_control	-domain	<p>Specifies the control signals for a previously defined isolation strategy.</p> <ul style="list-style-type: none"> • Supports self, parent, fanout, sibling, and automatic in pure Verilog designs. •
	-isolation_signal	
	-isolation_sense	
	-location	
set_port_attributes	-driver_supply	<p>This command is used to specify the boundary conditions of the design under test. For more information, see “Using set_port_attributes (SPA) Command”.</p>
	-receiver_supply	
set_retention		<p>Specifies which registers in the domain need to be retention registers, and sets the save and restore signals for the retention functionality.</p> <p>This command supports all UPF LRM switches, except the switches related to supply set.</p>
	-domain domain_name	The domain for which this strategy is applied.
	-exclude_elements exclude_list	Defines a list of objects: design elements, named processes, or sequential reg or signal names that are not included in this strategy.

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
	-retention_power_net net_name	Defines the supply net used as the power for the retention logic inferred by this strategy.
	-retention_ground_net net_name	Defines the supply net used as the power for the retention logic inferred by this strategy.
	-retention_supply_set	
	-elements element_list	Defines a list of objects: design elements, named processes, or sequential reg or signal names to which this strategy is applied.
	-no_retention	When this option is used, the storage elements specified by this strategy shall not have retention capability added.
	-save_signal {{logic_net <high low posedge negedge>}}	The -save_signal and -restore_signal options define a name of a logic net or port and its active level or edge. The default sensitivity is high (for both -save_signal and -restore_signal).
	-restore_signal {{logic_net <high low posedge negedge>}}	
	-save_condition {{boolean_function}}	Defines a Boolean expression. Default is True.
	-restore_condition {{boolean_function}}	Defines a Boolean expression. Default is True.
set_retention_control	-domain domain_name	Specifies the control signals and assertions for a previously defined retention strategy.
	-save_signal	The signal that causes the register values to be saved into the shadow registers.
	-restore_signal restore_net	The signal that causes the register values to be restored from the shadow registers.

Table 1-1 IEEE Std 1801 Commands Supported by MVSIM native mode

UPF Commands	Options Supported	Description
	-assert_r_mutex {{net_name <high low posedge negedge>}}	
	-assert_rs_mutex {{net_name <high low posedge negedge>}}	Even if you do not pass any argument to this option, you still have to use {}.
	-assert_s_mutex {{net_name <high low posedge negedge>}}	
set_related_supply_net		Associates an external supply net to design ports. For more information, see “Using set_related_supply_net Command” .
	-object_list	List of hierarchical references to the design or PG ports relative to the current scope.
	-ground	The ground net which drives the ground value for the design or PG ports declared in the –object list.
	-power	The power net which drives the power for the design or PG ports declared in the –object list.
set_retention_element		Defines a list of objects that can be used in set_retention and map_retention_cell commands. See UPF LRM for more information on list of options supported by this command.
set_scope		Specifies the active UPF scope.

* Non-Eclipse options or commands.

UPF Supply Package

UPF recommends a supply package to be used for driving values on the supply pads in the design. The supply package is read in from the release automatically in the UPF flow as the first file. In the UPF flow, a module or a package cannot have the name UPF as there is a built-in UPF package.

Supply pads are UPF Supply ports created in the UPF design top. The UPF design top is picked up from the UPF `set_design_top` command. The supply functions `supply_on` and `supply_off` are supported. The `PAD_NAME` to be passed to these functions must be a string constant and a valid UPF supply port created in the UPF top module.

In order to use these functions in the testbench, the testbench must import the UPF package as follows:

```
module testbench;
    import UPF::*;
    ----
    ----
endmodule
```

It is a good practice to embed the UPF "import" statement and the supply functions call in a include guard as follows (also helps compile the testbench in a non-UPF flow):

```
module testbench;
`ifdef UPF_1_0
import UPF::*;
`endif
----
----
    initial begin
`ifdef UPF_1_0
```

```

        supply_on("VDD", 1.0);
        supply_on("VSS", 0.0);
        supply_on("Vmem", 0.8);
    `endif
endmodule

```

The `synopsys_sim.setup` file should exist in the following area with these settings:

```

WORK > DEFAULT
DEFAULT : work
upf : $VCS_HOME/$platform/packages/upf

```

When the same `synopsys_sim.setup` has to be reused for different platforms, use the `$ARCH` variable, as shown below:

```

upf : $VCS_HOME/$ARCH/packages/upf

```

`ARCH` is set automatically by VCS based on the platform being used.

Note the following guidelines for using the supply functions:

- In Verilog and mixed designs, only `supply_pads` can be driven using supply calls. Supply nets cannot be driven.
- Real variable cannot be used as supply pads. Supply calls are required to be used.
- Supply pads declared as outputs cannot be driven using supply calls.

Using Low Power in VCS

You can specify only a single top-level UPF file for compilation in VCS.

To compile the UPF file, use the following command:

```
vcs -upf <filename> -power=<power_options> <vcs_options>  
<source_files>
```

`-upf`

Used to pass the UPF file to VCS.

`<filename>`

Identifies the UPF file.

`<power_options>`

Used to specify the following option:

`ret_level_as_edge` — Enables VCS to treat a level-sensitive retention policy defined in UPF as edge sensitive.

`<vcs_options>`

Identifies the VCS switches. You can use any HDL compilation switches in VCS with the `-upf` option. (For example, `-v`, `-y`, `+incdir`, `+define`, `-f` etc.)

`<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 UPF.

Note:

- UPF commands can be called from TCL procedures and can be interleaved with various TCL coding styles. VCS will execute the TCL commands that are part of the UPF file.
- Pure TCL commands with outputs ('puts') will be executed during UPF compile time and results displayed onto the screen/ log file as indicated by switches passed to VCS.

Power Techniques in VCS

VCS supports the following power techniques:

- Power-gating
- Isolation - Automatically inserting isolation cells as per the defined isolation policy in UPF.

- Retention - Save and Restore events as described in the UPF policy.

-

HDL Constructs and Techniques in VCS

VCS supports the following HDL construct in syntax and in LP semantics:

1. Verilog 95
 2. Verilog 2K
 3. SystemVerilog Design constructs
- Not corrupted:
 - Time
 - sv-string and character
 - enum
 - event
 - Dynamic data types: class objects (dynamic arrays, smart queues etc.)
 - Corrupted with: SIZE'X:
 - real
 - integer
 - logic
 - Corrupted with: SIZE'0:

- bit
- byte
- Int
- shortint
- int
- short int
- long int

Support in syntax but NOT in semantics

The SVTB HDL syntax is supported in VCS. VCS gives a parse warning if this construct is encountered. However, LP semantics are not applied on this construct during simulation.

Corruption Semantics

Corruption semantics are value of data types or events during shutdown. The following are the corruption semantics per UPF standard:

Power Gating

Following are the power gating semantics:

- Supply-package calls (`supply_on` and `supply_off`) determines the voltage value being applied to a supply network.
- All elements in the power-domain will continue with regular logic simulation in ON state.

- For a domain in PARTIAL-ON and OFF state, all design elements in that domain will be corrupted with their default initial value.

Retention

Following are the retention state semantics:

- By default, the SAVED data in VCS memory will be corrupted.
- On SAVE event, if back-up supply-net is in ON state, VCS saves in memory, the current values for all registers identified by the retention policy.
- SAVE event is ignored with appropriate ERROR if back-up supply-net is in OFF or partial-ON state.
- If back-up supply-net for retention policy goes into PARTIAL-ON or OFF state after a successful SAVE operation, SAVED data in VCS memory will be corrupted.
- On RESTORE event during ON state, VCS puts the saved values back into the respective registers.
- On RESTORE event during OFF or PARTIAL-ON state, the RESTORE operation fails and VCS gives appropriate error.

Isolation

Isolation is simulated as per the location identified in the `-location` switch in `set_isolation_control` command in UPF.

Modeling the Delay Specification for a Power Switch

To model the realistic behavior of a power switch, MVSIM native mode has a facility to specify a delay value to be used before the power switch turns ON after the control signal and the input power are valid. Use the `SNPS_power_switch_on_delay` attribute of the `set_design_attributes` command to specify the delay value, as shown below:

```
set_design_attributes -elements \  
{<Hierarchical name of the power switch policy>} \  
-attribute SNPS_power_switch_on_delay <value>
```

Add this command to the config file you specify at runtime using `-power <config file>` with the simulation executable. For example:

```
% simv -power lpconfig.tcl <other runtime options>
```

The delay units are based on the simulator timescale. For example, if you specify `value` as `100`, then for a timescale of `1ns/1ns` the power switch powers up after 100ns, after control is Active and input supply is ON.

Random Corruption

By default, MVSIM native mode corrupts four-state variables to `x` and two-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 MVSIM native mode:

- `-pa_random_corrupt` compile-time option — Enables random corruption for cont assign
- `-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>} \  
-attributes 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

Values not in the above list are illegal.

Note:

- If X or Z is chosen randomly as the corruption value, two-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 -elements {<power domain>} \
-attributes SNPS_random_seed <value>
```

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 a power domain

```
set_design_attributes -elements \  
{<PD_NAMES (separated by space)>} \  
-attribute SNPS_random_seed 25
```

4. Applying a random seed on the entire design

```
set_design_attributes -attribute SNPS_random_seed 37
```

Note:

- For a vector, all bits are corrupted with the same value. For example, `reg[2:0] a` gets corrupted to `3'b000`, `3'b111`, `3'bzzz`, `3'bxxx` depending on the type of corruption.
- Enum variables when corrupted with zero take the value of 0, and when corrupted to 1 take the value of all ones (`32'hfff_fff`).

UPF_dont_touch Support

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. No power activity such as corruption, isolation insertion and retention is seen on these modules or module signals.

- The recommended usage of this attribute is on leaf level modules.
- The module is treated as a black box when marked as `UPF_dont_touch`.
- All the instances of the don't touch model does not get corrupted during shut-off.
- When a model is marked as don't touch, then the model and its child instances, if any, 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.
- Instance of don't touch model will not have `upf_simstate` and `mvsim_flag` variables populated.
- Any specific pin of don't touch model cannot be marked as `UPF_dont_touch FALSE`.
- The isolation policy is not applied on any instance of don't touch model including the child instances, if any.
- Initial block retriggering does not replay the initial blocks in the don't touch model.

Usage

`UPF_dont_touch` on a design element can be invoked using `set_design_attributes` command. The options for the command are as follows:

```
set_design_attributes
  -models <model_list>
  -elements <element_list>
  -attribute UPF_dont_touch <TRUE/FALSE>
```

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

- `-elements <element_list>` without `-models` is not supported.
 - Instance level and hierarchical signal level `UPF_dont_touch` attribution is not supported.
- Specifying bit/part selects with `-elements` is not supported.

Setting Supply Port Default Value

This feature allows you to set default state on supply port using `set_design_attribute`. Once the default state is applied on a supply port which is a supply pad, it will start from that state.

Use Model

Set the default state for the given port using `set_design_attribute`, 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_attribute -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_attribute`.

```

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_attribute -elements { V3 } -attribute
SNPS_default_supply_net_state ON_D

```

Note:

If `set_design_attribute` is used to define default state of 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.

Initial Block Retriggering

MVSIM native mode supports initial block retriggering (`reinit`). You can use initial blocks in a design to assign power-up values to storage elements, write behavioral models of analog macros for digital simulations, handle memory reads through tasks, and so on.

However, with multiple power domains in the design, when you remove voltage from the power domain containing these initial blocks, MVSIM native mode simulations corrupt 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 MVSIM Native Mode

You can use the `set_design_attributes` UPF 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` — A list of design elements or named initial blocks.
- `-models model_list` — A 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 default), the command applies to the descendants of the elements.

Initial Block Retriggering Supported Attributes

Initial block retriggering supports the attributes shown in [Table 1-2](#).

Table 1-2 Supported Attributes for Initial Block Retriggering

Attribute Name	Description
SNPS_reinit	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.
SNPS_dont_reinit	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 below three properties to 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 applies the `reinit` or `don't reinit` property in descending order of priority, as listed below:

5. Named initial blocks in a module
6. Design instance
7. Design module
8. Power domain
9. Entire design

Transitive Reinit

You can apply the `reinit` property on a design instance or module transitively on all power-managed descendents of design module instances. VCS only applies the `transitive reinit` property on initial blocks which have the `uninitialized` property.

Examples for Different Reinit Scenarios

- Module-level support. That is, 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 a instance. That is, initial blocks of descendents are also retriggered on wakeup.

```
set_design_attributes -elements U_Inst \  
-attribute SNPS_reinit TRUE \  
-transitive TRUE
```

- Transitive support for a module. That is, initial blocks of descendents 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. That is, 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`.

Virtual Isolation Insertion Report

MVSIM native mode generates an isolation insertion report at compile which provides the details such as signal being isolated and its corresponding isolation control, isolation sense and clamp value.

Usage

Compile time option `'-power=write_mvinfo'`

Reporting Format

```
{isolation_control  isolation_sense  clamp_value  
isolated_signal}
```

1. Report will be dumped in `mvsim_native_reports/isolation_insertion_info.txt` 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, "low" if 0 in 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 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\[3\]}  
}
```

MVSIM Native Mode Limitations

- Timing simulation with SDF is not supported.

UPF Commands

This section describes the following topics:

- [“Using set_related_supply_net Command” on page 36](#)
- [“Using set_port_attributes \(SPA\) Command” on page 38](#)
- [“Using the query_cell_mapped Command” on page 41](#)
- [“Support for -resolve_parallel Option” on page 42](#)
- [“Wildcard Support in UPF Commands” on page 42](#)
- [“Using upf_tool Variable” on page 45](#)

Using set_related_supply_net Command

The `set_related_supply_net` 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 for hierarchical UPF flow and 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 in the `-power_top` compile-time option using the `set_design_top` command.

Use Model

The 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 one of the `-power` or `-ground` options in the above command. For unspecified option, this command picks the power or ground net of the power domain in the current scope.

If you do not specify `-object_list`, then all the ports of Design top specified by `set_design_top` or `-power_top` compile-time option are taken as the default object list. `objects` is the hierarchical reference to design or PG ports relative to the current scope.

Example

The following is the example which describes 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, the other ports remain ON as instA1's power domain A is ON.

Using `set_port_attributes` (SPA) Command

MVSIM native mode has added support for `set_port_attributes` command in line with the all the other tools in the Synopsys Eclipse Flow.

`set_port_attributes` is used to specify the boundary conditions of the design under test. During block level verification, this command is used to specify power information about the logic driving the primary input ports and logic driven by the primary output ports of the design.

Syntax

```
set_port_attributes
[-ports {port_list}]
[{-elements {element_list} [-applies_to <inputs | outputs |
both>]}]
[-receiver_supply supply_set_ref]
[-driver_supply supply_set_ref] [-attribute name value]
```

Supported attributes with `-attribute` are `iso_source`, `iso_sink`, `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 policy with the driver supply as `-source` will apply on the port as long as all other filtering criteria on policy are met. In other words, MVSIM native mode will use the `driver_supply` for policy association for this 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 policy with the receiver supply as `-sink` will apply on the port as long as all other filtering criteria on policy are met. In other words, MVSIM native mode will use the `receiver_supply` for policy association for this 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 MVSIM native mode for policy association. It is not used for determining the corruption on the port. If a port has both `iso_source` and `-driver_supply`, the `iso_source` will be used for policy association and `-driver_supply` will be used for corruption. Similarly, for `iso_sink` and `-receiver_supply`, the `iso_sink` will be used for policy association and `-receiver_supply` will be 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_attribute` or `set_related_supply_net` command on the port.

Comparison with `set_related_supply_net` (SRSN)

The `set_related_supply_net` command continues to have the same corruption and policy association semantics as before. However, if there is a `set_port_attributes` 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
```

- Policy association

```
iso_source/iso_sink > driver_supply/receiver_supply >
set_related_supply_net > related_supply_default_primary
```

Also, note that `set_port_attributes` can be applied on all ports of an instance by using the `-elements` option in the command. If a particular port figures both in `-elements` list and `-ports` list, the attribute with `-ports` list is applied.

`set_port_attributes` on Non-Boundary Ports

MVSIM native mode will apply the corruption semantics, but ignore the `set_port_attributes` for policy association. Instead, it will use 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 for the `set_port_attributes` command.

Using the `query_cell_mapped` Command

This command identifies the cell that is used for the named instance `instance_name`. 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.

Usage Example

The following is the usage example for the `query_cell_mapped` command:

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

Limitations

The following are the limitations of the `query_cell_mapped` command:

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

Support for `-resolve parallel` Option

MVSIM native mode supports the `-resolve parallel` option with the `create_supply_net` UPF command. The following limitation apply:

- If a supply net is in `UNDETERMINED` state, then the logic in the domain driven by this supply net is not corrupted.
- If a `create_power_switch` command in the UPF and power switch instantiated in the design drive the same supply net, then the UPF `create_supply_net` command for that supply net needs to have `-resolve parallel`.
- `-resolve parallel` is not supported with implicit supply sets. The supply set function must be explicitly connected to a supply net that is created using `-resolve parallel`.
- MVSIM native mode doesn't check that the root supply for parallel switches are the same.

Wildcard Support in UPF Commands

MVSIM native mode supports wildcard characters to refer design-objects in UPF. The support for wildcard characters is only applicable for design objects (HDL) and not for UPF objects. 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 wildcards in a string or different hierarchies. A wildcard character does not match:

- The hierarchy separator or patterns involving the hierarchy separator.
- The bus delimiters. However, a wildcard character matches the indexes of array instances.

For example:

`inst*` — Matches instance `[0]`

`array*[0]` — Does not match `array[3][0]`

`array[*][1]` — Does not match `array[0][1]`, `array[1][1]`

Table 1-3 UPF Commands Supporting Wildcard Characters

Commands	Options
<code>create_power_domain</code>	<code>-elements</code>
<code>set_isolation</code>	<code>-elements</code>
<code>set_retention</code>	<code>-elements</code>
<code>set_design_attributes</code>	<code>-models</code>
<code>set_related_supply_net</code>	<code>-object_list</code>

Examples

Wildcard characters:

? – For single character

* – For zero or more characters

Following is the basic example with `create_power_domain`:

Suppose there are two modules: Muxa, Muxb

To add both the modules in PD1

```
create_power_domain PD1 -elements {Mux?}
```

To add only Muxa using wildcard

```
create_power_domain PD1 -elements {*xa}
```

Wildcard Usage with Isolation

This example illustrates 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 shown below:

```
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 UPF_dont_touch

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 -model sta*  
set_design_attributes -attribute UPF_dont_touch TRUE -model MUXCY_?
```

Wildcard Usage with set_related_supply_net

Below example gives idea about usage of wildcard with set_related_supply_net.

```
set_related_supply_net -object_list {u_wrap_A/  
ou?_*[1:0]} -power V2 -ground VSS
```

Using upf_tool Variable

MVSIM native mode supports the usage of tcl variable `upf_tool` in the UPF file. For MVSIM native mode, 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 MVSIM native mode 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 MVSIM native mode or MVSIM.

```
if {$upf_tool == "MVSIM_NATIVE"} {  
    set_design_attribute ....  
}  
  
if {$upf_tool ne "MVSIM_NATIVE"} {  
    set_related_supply_net ...  
}
```

Unified Low Power Messaging

All the low power runtime messages capturing the power events during the simulation will be in the following format:

```
[time] [severity] [msg_id] <Message>
```

Example 1-1 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'.
```

The `vcs_lpmsg.log` file will be generated at runtime. This file captures all the Automated Low Power Assertions by default.

See [“List of Low Power Assertions \(LP_MSG\)”](#) section for the list of all low power runtime messages.

Message Control

The following two modes of message control are supported:

- Runtime Config File
- Function calls from Testbench

The options specified through the runtime config file are used for initialization. The function calls from the testbench will override 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 message/s
- Filter messages based on severity

Runtime Config File

Usage

```
-power <config_file>
```

The following options are supported in the config file.

Logging Mode

```
set_lp_msg_log_mode <DEFAULT/CUSTOM/  
DEFAULT_CUSTOM> -file { filename }
```

- DEFAULT - all the messages will be logged into the simulation log file specified with the `-l` runtime option
- CUSTOM - all the messages will be logged into the log file specified with `-file` with `set_lp_msg_log_mode`
- DEFAULT_CUSTOM - all the messages will be logged into the simulation log file specified with the `-l` runtime option and `-file` with `set_lp_msg_log_mode`

Example:

```
set_lp_msg_log_mode CUSTOM -file {test.log}
```

All LP messages will be dumped in 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 will result in the ON/OFF value being applied to all the 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_ISOEN_INIT }
```

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_ISOEN_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 will be filtered.

- INFO - Default severity level. No messages will be filtered.
- WARNING - messages with severity INFO will be filtered.
- ERROR - messages with severity INFO and WARNING will be filtered.

- FATAL - messages with severity FATAL will be ON and the rest will be 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 Any Message/s

```
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_ISOGEN_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 below control options, for listing 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");
```

Config:

```
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.

Below 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 below APIs, the Low Power Message Summary includes these custom messages.

The format of the message will be 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");
```

Assertions Related Clock/Reset Wiggle During Shutdown/Standby

You can use the `assert_clkrsttoggle` option, as shown below, to enable assertions related clock/reset wiggle during shutdown or standby.

Use Model

```
% vcs <vcs_options> -upf <upf_file> -  
power=assert_clkrsttoggle
```

The assertion will be thrown for the below conditions when `-power=assert_clkrsttoggle` is specified to VCS step:

- Clock is wiggling during CORRUPT/CORRUPT_ON_ACTIVITY simstate (LP_CLOCK_WIGGLE, LP_COA_CLOCK_WIGGLE).

- Reset is wiggling during CORRUPT/CORRUPT_ON_ACTIVITY simstate (LP_RESET_WIGGLE, LP_COA_RESET_WIGGLE).

Note:

- Enabling these assertions might have simulation performance impact.
- Reset wiggle assertions are for asynchronous resets only.

Supply-Set-Based Power Intent Specification Support

MVSIM native mode now supports supply-set-based specification of power intent. A supply set relates multiple supply nets as a complete power source for one or more design elements. Each supply net in a supply set provides a function.

MVSIM native mode currently supports the following predefined supply net functions:

- Power
- Ground
- 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.

MVSIM native mode supports the following supply-set-based UPF features:

- “Explicit Supply Sets with Explicit Supply Nets”
- “Explicit Supply Sets with Implicit Supply Nets”
- “Implicit Supply Sets”
- “Supply Sets in Policies”

Explicit Supply Sets with Explicit Supply Nets

This feature allows you to convert an existing supply-net-based UPF to a supply set. To enable this feature, you must create supply sets and connect the supply set handles to the existing supply nets using the `create_supply_set` command. Supply pads must be driven using `supply_on` and `supply_off` calls. Consider the code shown in [Example 1-2](#).

Example 1-2 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 choose to replace the `create_power_domain` and `set_domain_supply_net` commands with a single `create_power_domain -supply` command, as shown in [Example 1-3](#).

Example 1-3 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}
```

Using the explicitly created supply nets and ports, you can drive the different voltage values on to `supply_set` functions with `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 1-4](#).

Example 1-4 Explicit Supply Sets

```
create_supply_set SS_TOP
create_power_domain PD_TOP -supply {primary SS_TOP}
```

The supply set and consequently the power domain is driven to different states based on logic expressions defined in the `add_power_state` command (see [Example 1-5](#)).

Example 1-5 Add Power State Command

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

In [Example 1-5](#), note that the `-simstate CORRUPT` is mandatory to take the domain into shutdown; otherwise the domain continues to be in `NORMAL` state, and no corruption happens. Since there are no explicit nets and ports created, you cannot use `supply_on` and

`supply_off` calls to make state transitions. All power domain state transitions happen based on the value of the specified logic expression.

Implicit Supply Sets

This feature allows you to use default power domain handles in the UPF (see [Example 1-6](#)), thereby avoiding the need to create supply sets.

Example 1-6 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 1-6](#), 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.

Because there are no explicit nets and ports created, you cannot use `supply_on` and `supply_off` calls to make state transitions. All power domain state transitions happen based on the value of the specified logic expression.

Supply Sets in Policies

Isolation and retention policies now supports using supply-set functions instead of power and ground nets.

Limitations

Following are the limitations of MVSIM native mode support of Supply-Set-Based Power Intent Specification:

- Usage of the `supply_set` function in the `create_pst` command is not supported.
- Only `NORMAL`, `CORRUPT` and `CORRUPT_ON_ACTIVITY` simstates are supported.
- Implicit supply nets (`domain.primary.power/ground`) cannot be driven using `supply_on` or `supply_off` calls.

Using `set_isolation` Command

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

For any port in the `prefilter_element_list`, the following filtering functions are applied. If an element in `effective_element_list` is not on the interface of `ref_domain_name`, then it shall not be isolated.

- `-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 shall be 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 the domain for which this strategy is defined that have the specified mode.
- When one or more of the above filters are used together, then `effective_element_list` contains only those elements which satisfy all the filters.

The `-source` and `-sink` options require supply sets. For `-diff_supply_only`, it will be possible to use this option without supply sets, as the difference will be evaluated on a supply net basis.

The default value of `-applies_to` is `outputs` in the `set_isolation` command. But with `-source/-sink/-diff_supply_only` usage, the default value of `-applies_to` is changed to `both`.

Usage

```
set_isolation isolation_name
-domain ref_domain_name
[-elements element_list]
[-source source_supply_ref | -sink sink_supply_ref
| -applies_to <inputs | outputs | both>]
```

```

[-isolation_power_net net_name] [-isolation_ground_net
net_name]
[-no_isolation]
[-isolation_signal signal_list [-isolation_sense {<high |
low>*}]]
[-clamp_value {< 0 | 1 | any | Z | latch | value>*}]
[-location <automatic | self | parent]
[-diff_supply_only <TRUE | FALSE>]

```

Examples

Basic Source – Sink Policy

The basic usage of `-source/-sink` is illustrated here. `-source` and `-sink` uses 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` switch, are isolated whose source/sink criteria are satisfied.

```

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 with only `-source <source_supply_ref>` defined. In this case, all ports of reference domain, which are driven by the supply set specified in the `-source` switch, 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

With `-source` and `-elements` options used together in the `set_isolation` command, policy would only apply on those ports which 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 with only `-sink <sink_supply_ref>` defined. In this case, all ports of reference domain, for which 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

With `-sink` and `-elements` options used together in the `set_isolation` command, policy would only apply on those ports which 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 will be applied on the elements mentioned in the `-elements` switch, 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` will not be 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/FALSE

It determines the isolation behavior between driver and receiver supply sets. It applies the 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/FALSE>
```

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

Source – Sink with diff_supply_only

IEEE 1801 standard says that: It shall be an error if a

-diff_supply_only, -source, or -sink argument is used together. But MVSIM native mode supports it. In this case, policy should be applied to 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 will be applied on the elements mentioned in the

-elements switch, 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 will not be isolated. Other elements fulfilling the -diff_supply_only TRUE conditions 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 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 more 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 && -nonsocial`
- 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:

- When multiple isolation policies are referring to a single port, then a warning message is issued to indicate that multiple isolation policies have been applied and it also indicates that out of multiple policies, which policy has been applied to that port.
- Top domain boundary ports can not be considered as a source or sink. If you want to use top domain boundary ports as source or sink, then you must use `-set_port_attribute`. Refer to the IEEE 1801 standard for more details on the `-set_port_attribute` command.
- By default, Verilog Continuous Assignment statements are considered as feed-through in MVSIM native mode. Hence `cont_assigns` or buffers (Verilog Buf) in the design are not treated as a driver/load.
- When isolation policy is referring to any specific member of SystemVerilog union, then policy is applied on all the members of union. Whereas a specific member of SystemVerilog Structure is isolated properly.
- IEEE 1801 standard says that: It shall be an error if a `-diff_supply_only`, `-source`, or `-sink` argument is used together. But MVSIM native mode supports it.

Using `add_power_state` Command

MVSIM native mode now supports the `add_power_state` command. This section describes the simulation semantics of the this command. Following is the syntax for the `add_power_state` command:

```
add_power_state object_name
```



```
{-state state_name {[-supply_expr {boolean_function}] [-
logic_expr {boolean_function}]}
[-simstate simstate] [-legal | -illegal] [-update]}}
[-simstate simstate] [-legal | -illegal]
[-update]
```

The `object_name` must be a `supply_set`. You can use the `add_power_state` command to specify:

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

Supply Expressions and Logic Expressions

Supply expressions contain the `supply_set` function (`power`, `ground`). The expression should contain the state and voltage value, or a range of voltage values of the supply function. The expression can be a combination of multiple functions of the supply set (see [Example 1-7](#)).

Example 1-7 Supply Expressions

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

Logic expressions are regular SystemVerilog boolean expressions. You must use logic signals from the design to create logic expressions (see [Example 1-8](#)).

Example 1-8 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.

Supply Set State with Logic Expression

In case of explicit supply sets and implicit supply nets, the supply set reaches the state when the logic expression is `TRUE`. Consequently, the power domain driven by this supply set will go to the specified simstate.

Supply Set State with Supply Expression

If the functions are explicitly connected to supply nets in the UPF, then the supply expression is said to be fully specified. If so, the supply set reaches the state when the supply expression is `TRUE`.

If even one function in the supply expression is not connected to a supply net in the UPF, then the supply expression is not fully specified. In this case, the supply expression is ignored. This state can never be reached.

Supply Set State with Supply Expression and Logic Expression

If the supply expression is fully specified, then the supply expression is used to reach the state. The logic expression is used only for assertion.

If the supply expression is not fully specified, the logic expression is used to reach the state. The supply expression is ignored.

Simstate

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, then the simstate of a domain will be based on precedence order, as specified in the LRM. The order is `DEFAULT_CORRUPT`, `CORRUPT`, `NORMAL`, and `DEFAULT_NORMAL`.

The default simstate for power state is taken as `CORRUPT`, if supply expression uses `OFF` supply state for supply function power or ground.

Limitations

The following limitations of MVSIM Native Mode `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/-update` 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

MVSIM native mode now supports `associate_supply_set` UPF command.

Syntax

```
associate_supply_set supply_set_ref -handle  
supply_set_handle
```

Where,

- `supply_set_ref` is the rooted name of the supply set or a `supply_set_handle` to associate.
- `supply_set_handle` is the `supply_set` handle.

When an implicit supply set (supply handle) is associated with another supply set, they are resolved to same real supply net Power/Ground pair and hence 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 the 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 shall be an error to make circular associations.
- `associate_supply_set` command cannot be used with explicit supply sets in `-handle` option.

- Implicit supply set (supply handle) specified with `-handle` option must be at or below the scope of supply set with which it is being 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.

Controlling supply_set States Using `set_supply_set_state`

MVSIM Native Mode allows you to control `supply_set` states through testbench using the `set_supply_set_state` function. This means `supply_sets` which do not have any `-logic_expr` in the `add_power_state` command (that is, external supplies that are always-on, or shutdown externally) can be directly controlled through testbench.

Use Model

```
set_supply_set_state (supply_set_name, 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`.

Usage Example

Consider the following `add_power_state` command for `supply_set SS_TOP`.

```
add_power_state SS_TOP -state v1p2 {-supply_expr {power ==  
  `{FULL_ON, 1.2}}}  
add_power_state SS_TOP -state zero {-supply_expr {ground ==  
  `{FULL_ON, 0.0}}}
```

The `set_supply_set_state` function will be called from testbench, as shown below, to set the `supply_set` to a given `power_state`.

```
initial begin
    set_supply_set_state(SS_TOP , v1p2);
    set_supply_set_state(SS_TOP , zero);
end
```

Simulation Behavior

[Table 1-4](#) describes describes the simulation behavior of `set_supply_set_state`.

Table 1-4 set_supply_set_state behavior

supply_set	add_power_state	set_supply_set_state behavior
Dummy Nets		
	logic_expr	<ol style="list-style-type: none"> 1. Changes to the specified power state only when <code>logic_expr</code> matches. 2. Warns if the <code>logic_expr</code> is not true when <code>set_supply_set_state</code> is used.
	supply_expr	<ol style="list-style-type: none"> 1. Changes to the specified power state when <code>set_supply_set</code> state used. 2. Drives the supplies to voltage specified in <code>supply_expr</code>.

supply_set	add_power_state	set_supply_set_state behavior
	Both	<ol style="list-style-type: none"> 1. Changes to the specified power state only when <code>logic_expr</code> matches. 2. Warns if the <code>logic_expr</code> is not true when <code>set_supply_set_state</code> used. 3. Drives supplies to voltage specified in <code>supply_expr</code> if <code>logic_expr</code> is true.
Defined Nets		
	<code>logic_expr</code>	<ol style="list-style-type: none"> 1. Changes to the specified power state only when <code>logic_expr</code> matches. 2. Warns if the <code>logic_expr</code> is not true when <code>set_supply_set_state</code> is used.
	<code>supply_expr</code>	<ol style="list-style-type: none"> 1. Changes to the specified power state when <code>supply_expr</code> matches. 2. Warns if the <code>supply_expr</code> is not true when <code>set_supply_set_state</code> is used.
	Both	<ol style="list-style-type: none"> 1. Changes to the specified power state when <code>supply_expr</code> matches. 2. Warns if the <code>supply_expr</code> is not true when <code>set_supply_set_state</code> is used. 3. Warns if <code>supply_expr</code> and <code>logic_expr</code> mismatch.

CORRUPT_ON_ACTIVITY Simstate (Low-Vdd Standby)

In simstate, CORRUPT_ON_ACTIVITY, the active state of nets and state elements driven by the element will remain unchanged at the transition. The processes modeling the behavior of the element will remain enabled for activation (evaluation). Any net or state element that is actively driven after transitioning to this state will be corrupted.

-simstate of add_power_state command now 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

The following table describes the simulation behavior of CORRUPT_ON_ACTIVITY.

Table 1-5 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
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

- 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.

Power Aware Verification Environment

Power aware verification allows you to functionally verify the operation of a design under active power management.

Power Aware Verification Environment (PAVE) is an infrastructure that enables accessing the UPF objects, monitor low power events, and write power-aware assertions. It 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.

Low Power Assertions in UPF Flow

Low power assertions are classified into two categories:

- “Automated Assertions”
- “Custom Assertions”

Note:

You must use the `-sverilog` compile-time option to enable Low Power Assertions.

Automated Assertions

Automated assertions are inbuilt assertions that report all the low power events and also check for any shutdown protocol violations. Automated assertions are exhaustive set of messages related to power domain, supply network, power switch, isolation, retention, and PST.

Use Model

Use the `-sverilog` compile-time option to enable automated assertions. All low power messages will be logged into the `lpa_msg.log` file. These messages will have the below format:

```
[Time] [Severity] [Tag] <Message Text>
```

Example:

```
[2665] [INFO] [LP_PD_STATE_CHANGE] Power domain  
'testbench/top/Island_V2' state changed from  
CORRUPT to NORMAL.
```

Note:

For the list of all the automated assertions, refer to [“List of Automated Assertions”](#).

Assertion Control Options

MVSIM NATIVE provides powerful set of assertion control options like:

- Ignoring assertions based on Tag, Severity, and Time
- Overriding the severity
- Multiple log modes

MVSIM NATIVE supports the following three methods to provide assertion controllability:

- [PAVE Control Configuration File](#)
- [Runtime Control Options](#)
- [PAVE APIs for Dynamic Control](#)

PAVE Control Configuration File

This configuration file is read by MVSIM Native at the start of simulation for initialization.

MVSIM NATIVE reads the PAVE control configuration file from the locations in descending order, as shown below:

1. File defined by the `SNPS_LPA_CONFIG` environment variable
2. File passed with the `+lpa_config=<filename>` runtime option
3. The `lpa_config.ini` file in the current working directory

If all the above mentioned steps fail, then MVSIM NATIVE reads the default configuration file from the below mentioned path:

```
$VCS_HOME/packages/lpa/lpa_config.ini
```

The following section describes the PAVE configuration file (`lpa_config.ini`) options.

PAVE Configuration File Options

The following table lists the PAVE configuration file options. You can use these options for time zero initialization.

Option	Description
<code>lpa_AssertOff = <TRUE/FALSE></code>	When set to TRUE, this option will turn OFF all the low power assertions. Default value is FALSE.
<code>lpa_severity = <INFO/ WARNING/ERROR/FATAL></code>	Must be one of INFO, WARNING, ERROR, FATAL. Assertions with specified severity level and above will be printed. Default severity is INFO.
<code>lpa_ignore_assertions = <ID1 : ID2 : .. : IDn></code>	Specify the colon separated LP assertion IDs that need to be ignored.
<code>lpa_log_mode = <DEFAULT/ CUSTOM/ DEFAULT_CUSTOM></code>	<p>Specifies the logging mode for all low power assertions, where:</p> <ul style="list-style-type: none">• DEFAULT: Log in the simulation log file specified by the <code>-l</code> VCS option and STDOUT.• CUSTOM: Log only in the log file specified by the <code>lpa_logfile</code> ini variable.• DEFAULT_CUSTOM: Log in both the simulation log and the file specified by the <code>lpa_logfile</code> config setting including STDOUT. <p>The default mode is DEFAULT.</p>
<code>lpa_logfile = <filename></code>	<p>Specifies the file name where all the low power assertions should be logged when <code>lpa_log_mode</code> is CUSTOM or DEFAULT_CUSTOM.</p> <p>If unspecified, the default custom log file is <code>lpa_msg.log</code> in the current working directory.</p>

Note:

The ID or IDs that are ignored using `lpa_ignore_assertions` will remain ignored unless explicit `lpa_enable_assertions` API is used to enable them.

Runtime Control Options

Runtime command-line options have higher precedence over the configuration file settings. The following table describes the runtime control options which are used to provide assertion controllability.

Option	Description
+lpa_config=<filename>	Specifies the path to PAVE control config file name.
+lpa_AssertOff	Turns OFF all the low power assertions. The low power assertions for that simulation are turned OFF completely. This option ignores all the other control options including the PAVE APIs.
+lpa_log_mode=<log mode>	Specifies the logging mode for all low power assertions. Mode can be DEFAULT, CUSTOM or DEFAULT_CUSTOM. This option overrides lpa_log_mode setting in the <code>lpa_config.ini</code> file.
+lpa_logfile=<filename>	Specifies the file name to which all the low power assertions should be logged. By default, all the assertions will be logged into the simulation log file specified by the -l VCS option. This option overrides the lpa_logfile setting in the <code>lpa_config.ini</code> file.
+exit_on_lpa_error	Use this option to end the simulation on any low power assertion with severity ERROR.

PAVE APIs

All the PAVE config file (`lpa_config.ini`) settings can be overridden using the PAVE APIs. The PAVE APIs are used to provide dynamic assertion control.

Use Model

To use the PAVE APIs, you must import the `SNPS_LPA_PKG` package:

```
import SNPS_LPA_PKG ::*;
```

The following table lists the PAVE APIs:

Function	Description
<code>`lpa_severity(lpa_severity_T severity);</code>	Assertions with specified severity level and above will be printed. Argument 'severity' can be INFO, WARNING, ERROR, or FATAL. This option overrides the <code>lpa_severity</code> setting in the <code>lpa_config.ini</code> file.
<code>`lpa_override_severity(string ID, lpa_severity_T severity);</code>	Specifies the appropriate ID and the desired severity. This option overrides the default severity level of a specific low power assertion.
<code>`lpa_ignore_assertions(ID1 : ID2: .. : IDn)</code>	Specifies the colon separated low power assertion IDs that need to be ignored. It appends the list of IDs to the list specified with the <code>lpa_ignore_assertions</code> setting in the <code>lpa_config.ini</code> file. Pattern matching is supported, and the expression must be a Perl regular expression.
<code>`lpa_enable_assertions(ID1 : ID2: .. : IDn)</code>	Specifies the colon separated low power assertion IDs that need to be enabled. Pattern matching is supported and the expression must be a regular Perl regular expression.

Function	Description
<code>`lpa_AssertOff</code>	When set to TRUE, this option turns OFF all the low power assertions. This option is equivalent to <code>lpa_AssertOff</code> setting in <code>lpa_config.ini</code> file.
<code>`lpa_AssertOn</code>	Returns to the last known ON state. Setting it to TRUE will turn ON all the low power assertions excluding the ignored assertions and the assertions turned OFF based on severity. This option enables the assertions even if the <code>lpa_AssertOff</code> setting in the <code>lpa_config.ini</code> file is set to TRUE.
<code>`lpa_register_assertions</code> (<code>lpa_msg_table_T</code> <code>lpa_custom_table</code>);	Specifies the list of custom assertions that must be registered. Argument <code>lpa_custom_table</code> should be of predefined type <code>lpa_msg_table_T</code> , a dynamic array of structure. Once registered, all the controllability options are also applicable to the custom assertions as well.
<code>`lpa_log(string ID, msg);</code>	For overall consistency, you can choose the MVSIM NATIVE messaging format for custom assertions. All logging options apply to the custom assertions, if you follow the below syntax for the <code>msg</code> argument: <pre>`lpa_log("ID", \$psprintf(`lpa_msg_format("ID"), ARG1, ARG2, ..., ARGn));</pre> This option is the suggested alternative to system tasks like <code>\$display</code> , <code>\$write</code> .
<code>`lpa_id_enabled(string ID);</code>	Returns 1 if ID is enabled, else this option returns 0. Logic in the custom checker module can be under this check for each custom ID.

PAVE API Usage Examples

You must use the PAVE APIs only in the procedural blocks (initial/always). The following table contains the usage examples of PAVE APIs.

Usage	Example
To disable low power assertions with severity INFO.	<code>`lpa_severity(WARNING);</code>
To change the severity of low power assertion ID LP_RESTORE_X to FATAL.	<code>`lpa_override_severity("LP_RESTORE_X", FATAL);</code>
To ignore assertions, pass the colon separated list of IDs.	<code>`lpa_ignore_assertions("LP_PGN_STATE_CHANGE : LP_PPN_INIT_VALUE :");</code>
To enable the ignored list of assertions.	<code>`lpa_enable_assertions("LP_PGN_STATE_CHANGE : LP_PPN_INIT_VALUE :");</code>
To turn OFF all low power assertions.	<code>`lpa_AssertOff</code>
To turn ON all the assertions excluding the ignored.	<code>`lpa_AssertOn</code>
To register custom assertions	<code>`lpa_register_assertions(lpa_custom_table);</code>
To check if a certain assertion LP_MSG is enabled.	<pre>If (`lpa_id_enabled("LP_MSG")) begin `lpa_log(...); end</pre>
To print custom assertion in MVSIM NATIVE format.	<pre>`lpa_log("MY_RESET_X", \$psprintf(`lpa_msg_format("MY_RESET_X"), reset_n));</pre> <p>Where, the corresponding entry for MY_RESET_X in "lpa_custom_table" is:</p> <pre>'{"MY_RESET_X", "ERROR", "X on reset signal '%s'.", 0}</pre>

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 and `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.

Use Model

The below mentioned steps describe the use model for custom assertions in detail.

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 in MVSIM NATIVE”](#) .

Example:

```
% vcs <options> -upf <upf_file> -lpa_bind  
<custom_bind_file> -sverilog
```

Use Model to Turn OFF Automated Assertions and Retain Custom Assertions

Use `-power=lpa_override` compile-time option to turn OFF automated assertions and retain custom assertions.

Note:

All controllability options are available under override mode. Only the automated low power assertions are turned OFF.

Use Model to Turn Off All Low Power Assertions

Use the `-power=nolpa` compile-time option to turn OFF both automated and custom low power assertions.

Writing Custom Low Power Assertions in MVSIM NATIVE

Follow the below steps to write custom low power assertions in MVSIM NATIVE:

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 will monitor the low power events and parameters provide the static information. Do the following to register custom assertion:

- In the testbench, populate the custom message table of pre-defined type `lpa_msg_table_T`.
- Register the custom assertion ID using the ``lpa_register_assertions` PAVE API.

3. Compile the Custom Bind File and the Checker Module

Use `-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 need to 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) is dynamically varying during the simulation, it must be bound as a port.

2. Creating SV Checker Module and Registering Custom Assertion

Populate the PAVE message table (lpa_msg_table_T) with the custom message and register it using the ``lpa_register_assertions` API.

Example Code:

testbench.sv

```
module testbench;
    import UPF::*;
    import SNPS_LPA_PKG::*;
    ....
    lpa_msg_table_T lpa_custom_table = '{'{"LP_PPN_OFF",
ERROR, "Primary power net '%s' of power domain '%s' turned
OFF.", 0}};

    initial
        `lpa_register_assertions(lpa_custom_table);
        .....
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_LPA_PKG::*;
    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 (`lpa_id_enabled("LP_PPN_OFF"))
                my_assert : assert (PPN_STATE != "OFF") else
                    `lpa_log("LP_PPN_OFF", $psprintf(
`lpa_msg_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.
```

Debugging Low Power Designs Using DVE

You can use DVE to debug your low power simulations. DVE allows you to visualize, trace, and analyze the source of low power events.

It simplifies low power debug by allowing you to visualize the power intent (UPF), and supports features to determine whether unexpected design behavior is caused by functional logic or a power event. Low power debug in DVE is available for all UPF-based low power simulations.

This section describes the following topics:

- [Dumping Low Power Data to VPD](#)
- [Low Power Debug Features](#)

Dumping Low Power Data to VPD

Apart from dumping the design data into VPD, use the following approach to enable dumping of low power data to VPD:

Dump all Low Power Objects

Use any one of the below methods to enable or disable dumping of all low power objects to VPD:

- Call `$vcdpluspowerenableon` or `$vcdpluspowerenableoff` system tasks in the design.
- From UCLI using `dump -power on/off`.
- From UCLI using `call {$vcdpluspowerenableon}` or `call {$vcdpluspowerenableoff}`.

Dump Power Domain State Signals and Power State Table Supply Nets Only

Use any one of the below methods to enable or disable dumping of only power domain state signals and PST supply nets to VPD:

- Call `$vcdpluspowerstateon` or `$vcdpluspowerstateoff` system tasks in the design.
- From UCLI using `dump -powerstate on/off`.
- From UCLI using `call {$vcdpluspowerstateon}` or `call {$vcdpluspowerstateoff}`.

Low Power Debug Features

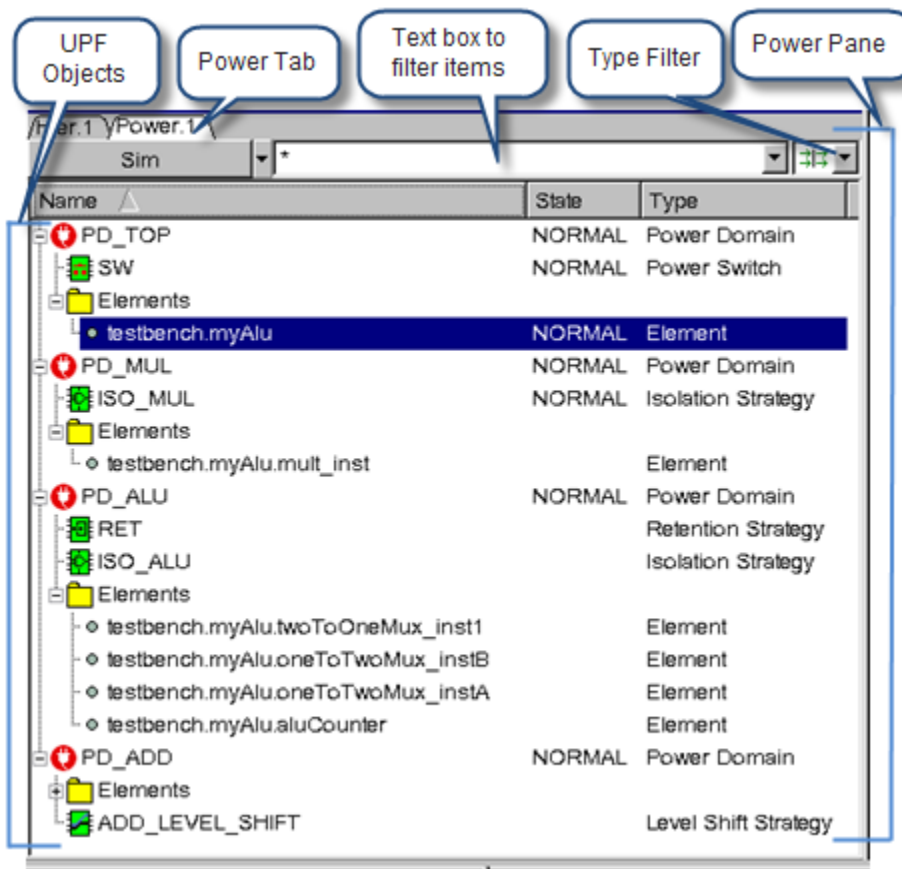
This section describes the following features supported in DVE to debug low power designs:

- [Power Pane](#)
- [Low Power Objects in Hierarchy Pane](#)
- [Marking for Low Power Signals in Data Pane](#)
- [UPF Source View](#)
- [Waveform Shading](#)
- [Power State Table View](#)
- [Shading in Schematic View](#)
- [Power Domain Information for Drivers/Loads](#)

Power Pane

The Power pane displays the objects defined in the UPF file. Select the Power tab to display the Power pane and view the low power objects. The following figure shows the Power pane.

Figure 1-1 Power Pane










The Power pane, shown in the above figure, is a tree view composed of the following columns:

- **Name:** Displays the name of low power object.
- **State:** Displays the state of the power domain. The value is updated during the simulation.
- **Type:** Displays the type of low power object.

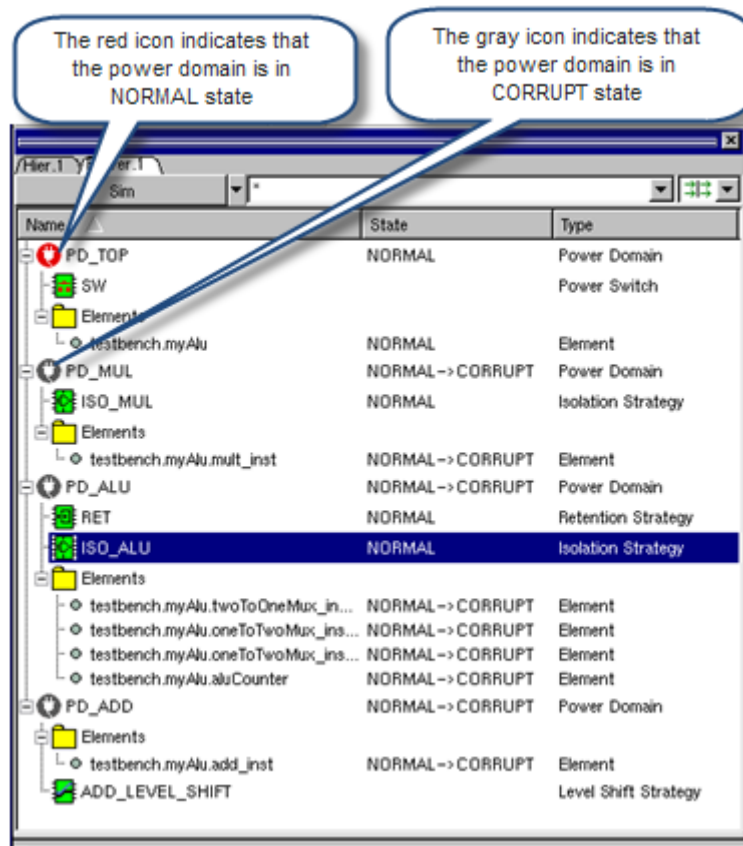
You can detach and reattach the Power tab at any time during your session. To detach the Power tab, toggle the **Power** off by selecting **Window > Panes > Power**. The following table describes the low power objects in the Power pane:

Table 1-6 Low Power Objects in the Power Pane

Object Name	Icon	Type
Power Domain		Power Domain
Power Switch		Power Switch
Isolation Strategy		IsoPol
Retention Strategy		RetPol
LevelShifter Strategy		LSPol
Elements List under Power Domain		Module/Entity
Elements List under Strategy		Module/Entity

The CORRUPT power domain state is represented as a gray icon.
Figure 1-2 shows the CORRUPT and NORMAL power domain states.

Figure 1-2 Power Domain State



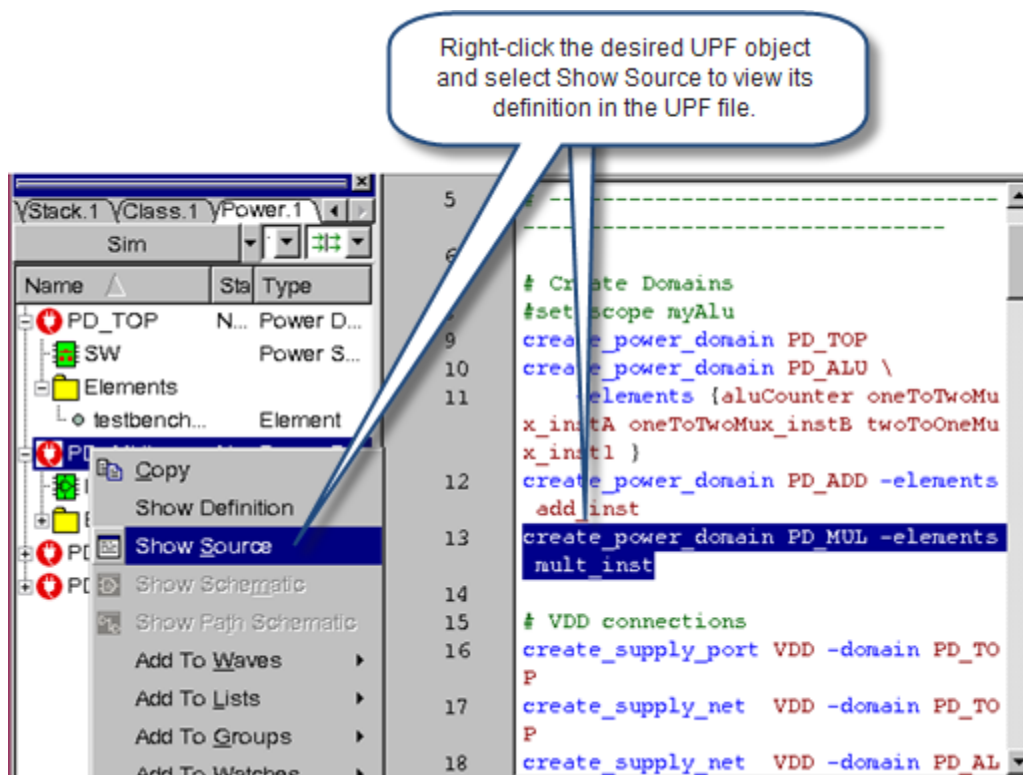
The red icon indicates that the power domain is in NORMAL state

The gray icon indicates that the power domain is in CORRUPT state

Name	State	Type
PD_TOP	NORMAL	Power Domain
SW		Power Switch
Element		
testbench.myAlu	NORMAL	Element
PD_MUL	NORMAL-> CORRUPT	Power Domain
ISO_MUL	NORMAL	Isolation Strategy
Elements		
testbench.myAlu.mult_inst	NORMAL-> CORRUPT	Element
PD_ALU	NORMAL-> CORRUPT	Power Domain
RET	NORMAL	Retention Strategy
ISO_ALU	NORMAL	Isolation Strategy
Elements		
testbench.myAlu.twoToOneMux_in...	NORMAL-> CORRUPT	Element
testbench.myAlu.oneToTwoMux_ins...	NORMAL-> CORRUPT	Element
testbench.myAlu.oneToTwoMux_ins...	NORMAL-> CORRUPT	Element
testbench.myAlu.aluCounter	NORMAL-> CORRUPT	Element
PD_ADD	NORMAL-> CORRUPT	Power Domain
Elements		
testbench.myAlu.add_inst	NORMAL-> CORRUPT	Element
ADD_LEVEL_SHIFT		Level Shift Strategy

To view the source of an UPF object in the Source View, right-click the desired UPF object in the Power pane and select **Show Source**, as shown in [Figure 1-3](#). This option points to the definition in the UPF file.

Figure 1-3 Viewing the Source of an UPF object



Similarly, to switch to scope of an UPF object, right-click the desired UPF object in the Power pane and select **Show Definition**. The scope where UPF object is defined is shown in the Hierarchy pane. For more information on Hierarchy pane, see [“Low Power Objects in Hierarchy Pane”](#).

Filtering the Objects in the Power Pane

You can filter object types such as Isolation Strategy, Retention Strategy, LevelShifter Strategy, and Power Switch in the Power pane, based on the object types mentioned in [Table 1-6](#).

To filter the data, based on object types, click Type Filter and select or clear the desired object types, as shown in [Figure 1-4](#). By default, all object types are ON.

Figure 1-4 Type Filter list in Power pane

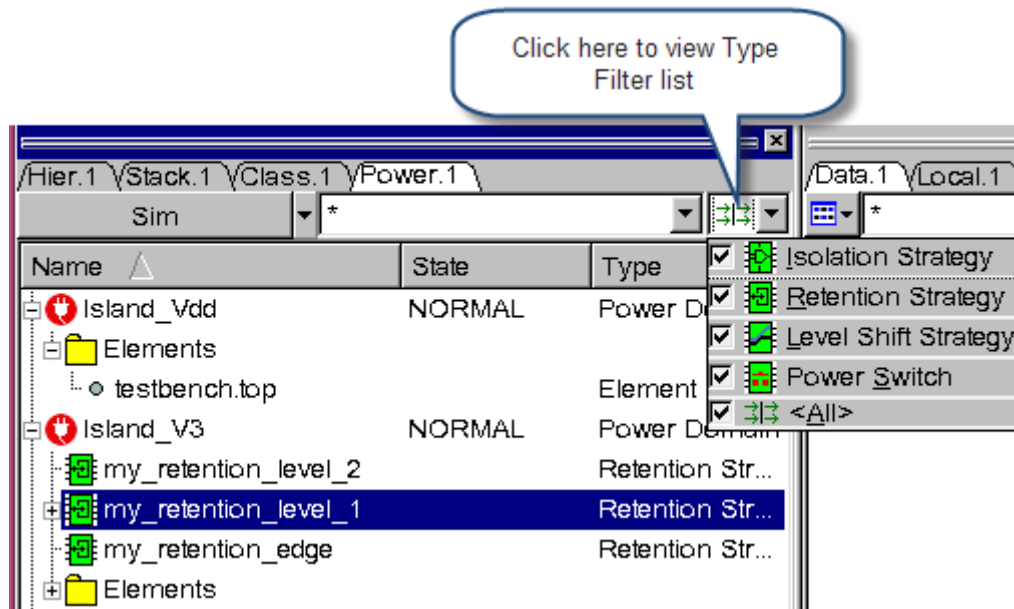


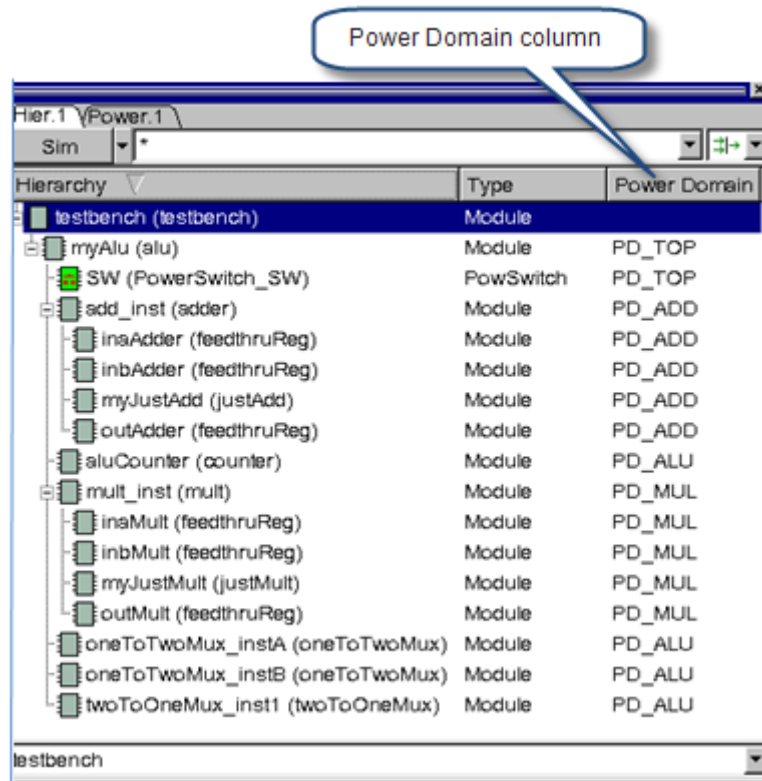
Table 1-7 Mouse Actions in Power Pane

Mouse Action	Command Operations
Hover the mouse cursor on a low power object	Displays tooltip with relevant power domain information. Tooltip shows the name of the object, power domain, and power domain state.

Low Power Objects in Hierarchy Pane

A new column **Power Domain** is added to the Hierarchy Pane to display the corresponding power domain of the design instance. [Figure 1-5](#) shows the Power Domain column.

Figure 1-5 Power Domain Column





Power Domain column

Hierarchy	Type	Power Domain
testbench (testbench)	Module	
myAlu (alu)	Module	PD_TOP
SW (PowerSwitch_SW)	PowSwitch	PD_TOP
add_inst (adder)	Module	PD_ADD
insAdder (feedthruReg)	Module	PD_ADD
inbAdder (feedthruReg)	Module	PD_ADD
myJustAdd (justAdd)	Module	PD_ADD
outAdder (feedthruReg)	Module	PD_ADD
aluCounter (counter)	Module	PD_ALU
mult_inst (mult)	Module	PD_MUL
insMult (feedthruReg)	Module	PD_MUL
inbMult (feedthruReg)	Module	PD_MUL
myJustMult (justMult)	Module	PD_MUL
outMult (feedthruReg)	Module	PD_MUL
oneToTwoMux_instA (oneToTwoMux)	Module	PD_ALU
oneToTwoMux_instB (oneToTwoMux)	Module	PD_ALU
twoToOneMux_inst1 (twoToOneMux)	Module	PD_ALU

The following low power object types are added to the Hierarchy Pane:

Table 1-8 Low Power Object Types in the Hierarchy Pane

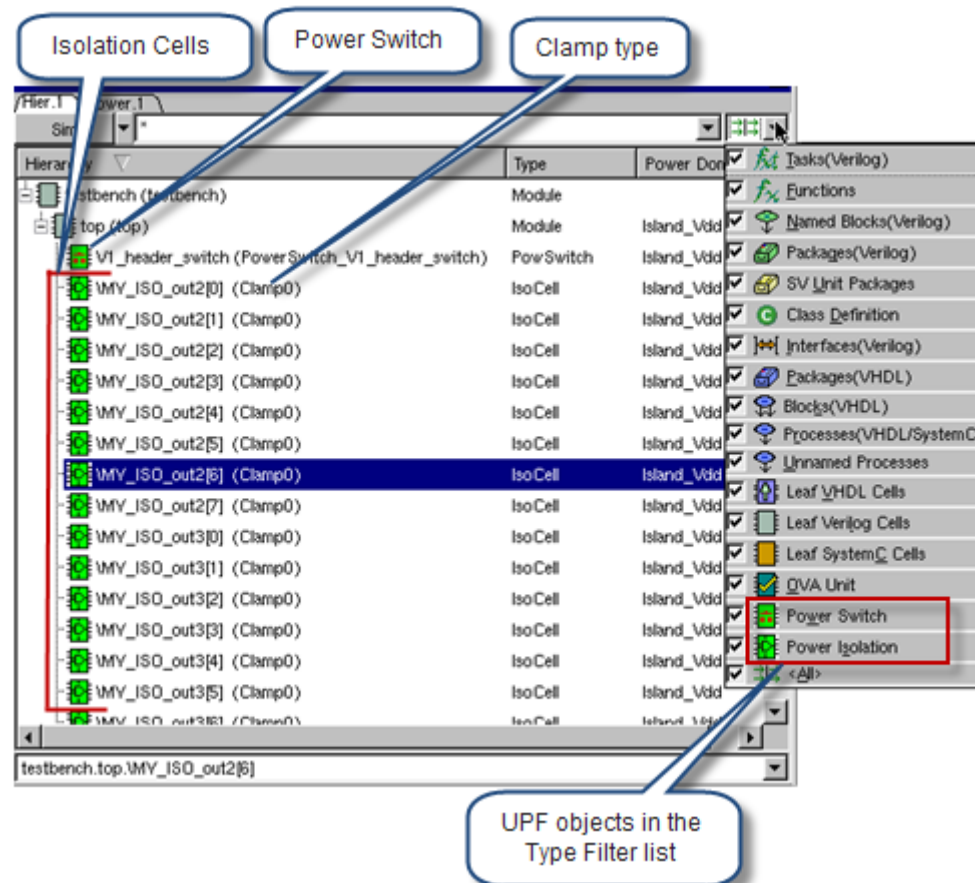
Object Name	Icon	Type
Power Switch		Power Switch
Power Isolation		Isocell

The low power object types mentioned in the above table are added to the Type Filter list, as shown in [Figure 1-6](#).

Note:

The Power Switch and Power Isolation object types are OFF by default.

Figure 1-6 Low Power Objects in the Hierarchy Pane



DVE allows you to view the corresponding power domain of a design instance in the Hierarchy Pane.

To view the corresponding power domain of a design instance:

Right-click the desired design instance in the Hierarchy Pane and select **Show Power Domain**. The control shifts to the corresponding power domain in the Power pane.

DVE allows you to view the source of an UPF object in the Source View.

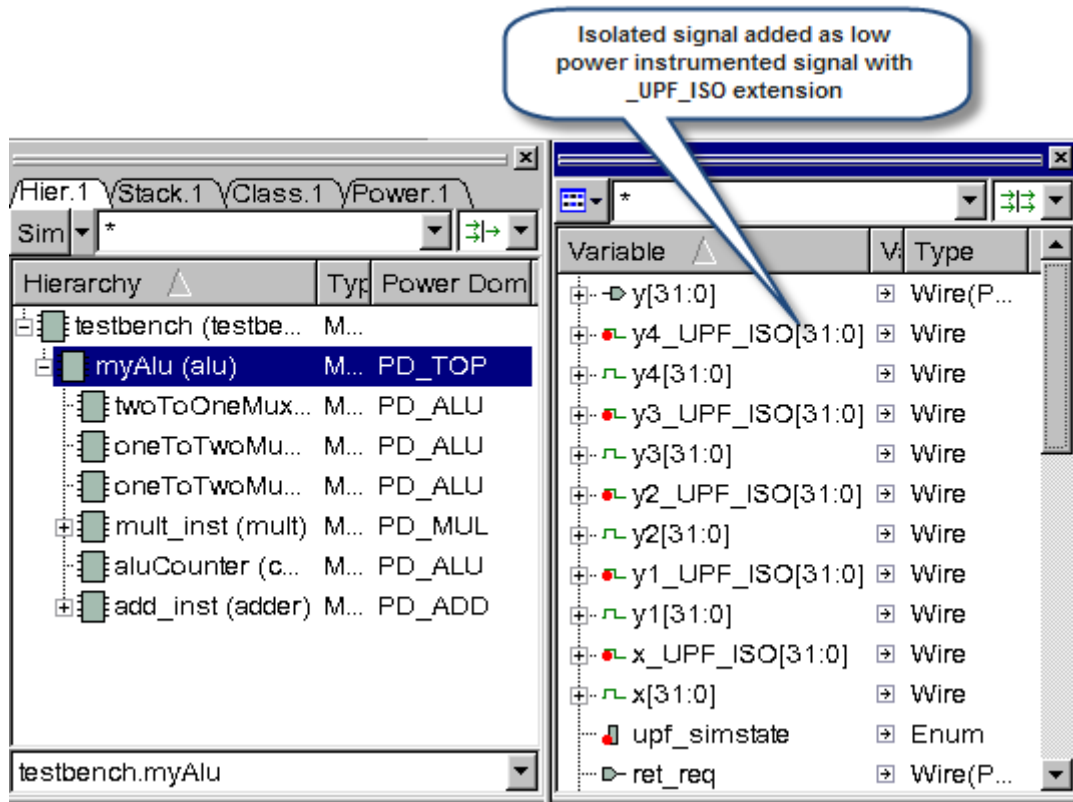
To view the source of an UPF object in the Source View:

Right-click the desired UPF object in the Hierarchy pane and select **Show Source**. This option points to the UPF object definition in the UPF file.

Isolated Signals

You can view the isolated signals in an instance in the Data pane with `_UPF_ISO` extension, as shown in [Figure 1-7](#).

Figure 1-7 Isolated Signals



Shadow Registers

You can view the shadow registers of an instance in the Data pane with `save_____` or `save__` prefix, as shown in [Figure 1-8](#).

Figure 1-8 Shadow Registers

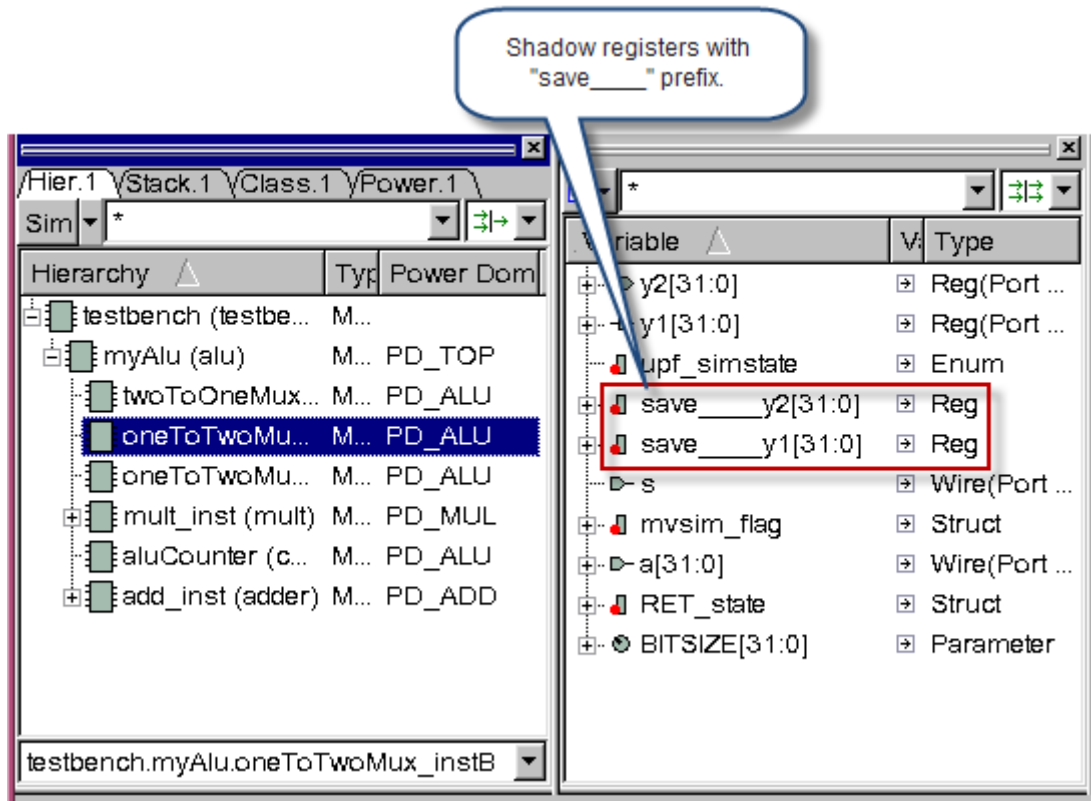


Table 1-9 Mouse Actions in Hierarchy Pane

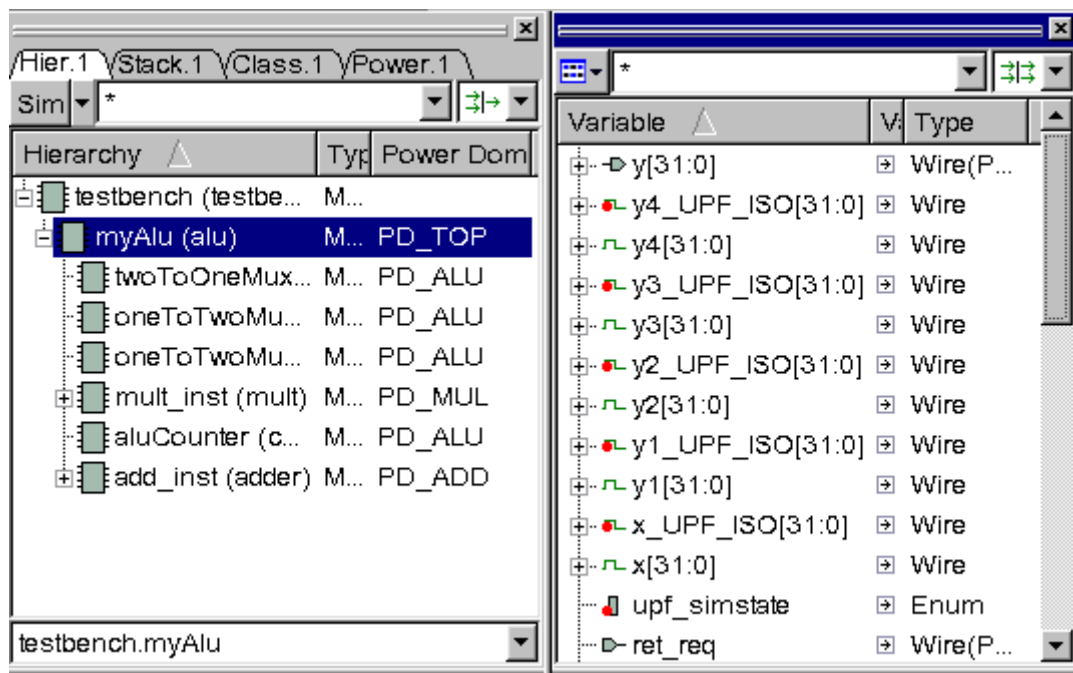
Mouse Action	Command Operations
Hover the mouse cursor on a low power object	Displays tooltip with relevant power domain information. Tooltip shows the name of the object, power domain, and power domain state.

Marking for Low Power Signals in Data Pane

The Data Pane displays the values of signals and variables of the corresponding instance selected in the Hierarchy pane. If you select an instance in the Hierarchy pane, then the Data pane displays the variables of the selected instance. Similarly, Data pane is populated with the necessary data for the objects in the Power pane.

All low power signals (supply nets, upf_simstate, isolated signals, shadow registers) are marked with a red dot, as shown in the following figure, for clear identification.







Figure 1-9 Marking for Low Power Signals in the Data Pane



Low Power Objects in Data Pane

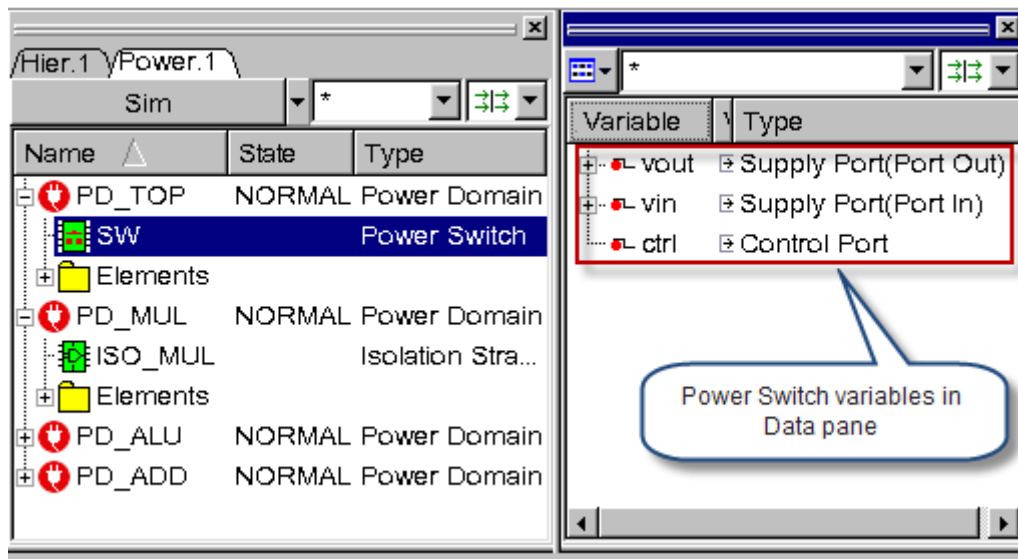
The following table describes the low power objects in the Data Pane:

Table 1-10 Low Power Objects in the Data Pane

Icon	Used by these objects	Type
	Primary Net, Isolation Power Net, Primary Power Net, Retention Power Net	Power Net
	Ground Net, Isolation Ground Net, Primary Ground Net, Retention Ground Net	Ground Net
	Isolation Enable Signal	Logic
	Save Signal	Save Signal
	Restore Signal	Restore Signal
	Control Signal	Control Port
	All Input Supply Nets	Input Supply Port
	All Output Supply Nets	Output Supply Port

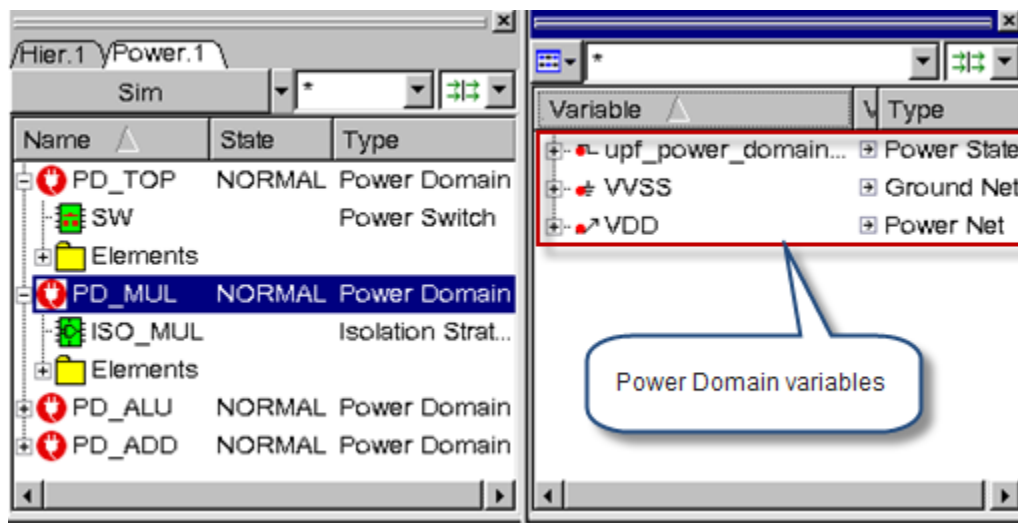
If you select a Power Switch in the Power tab, then the Data pane displays Input Supply Port, Output Supply Port, and Control Port of the Power Switch, as shown in the following figure:

Figure 1-10 Power Switch



If you select a Power Domain in the Power tab, then the Data pane displays Power Net, Ground Net, and simstate of the Power Domain, as shown in [Figure 1-11](#).

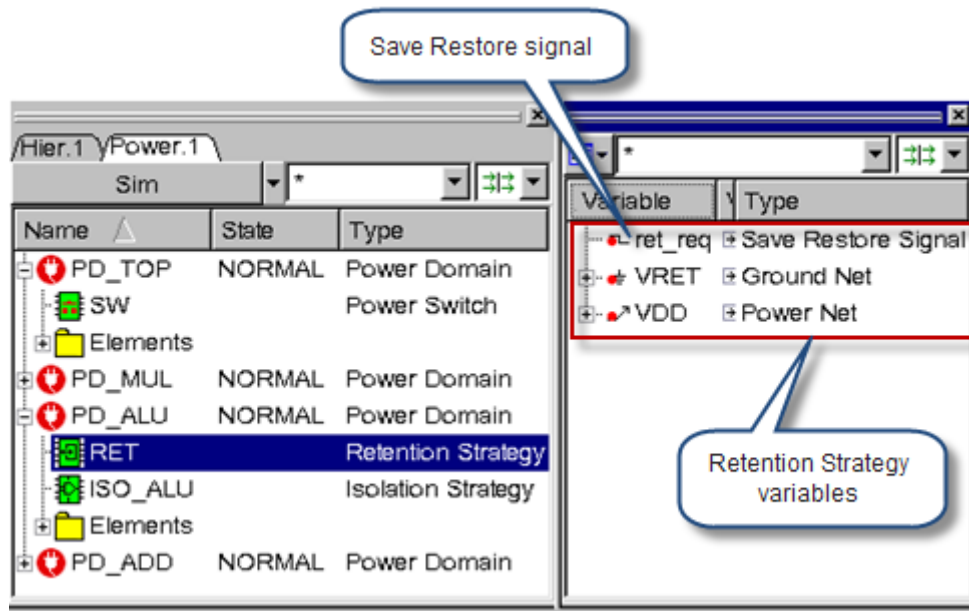
Figure 1-11 Power Domain



If you select Retention Strategy in the Power tab, then the Data pane displays the Retention Power Net, Retention Ground Net, and Save and Restore signals of the Retention Strategy, as shown in [Figure 1-12](#).

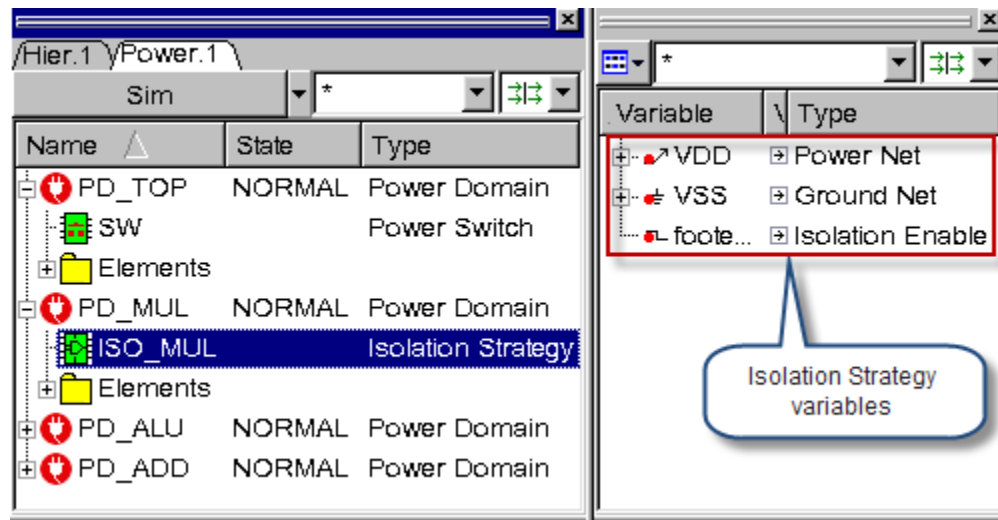
If you use same signal as both Save and Restore signals for the retention policy, then the type of that signal is displayed as **Save Restore Signal** in Data pane, as shown in the following figure:

Figure 1-12 Retention Strategy



Similarly, if you select Isolation Strategy in the Power tab, then the Data pane displays Isolation Power Net, Isolation Ground Net, and Isolation Enable of the Isolation Strategy, as shown in [Figure 1-13](#).

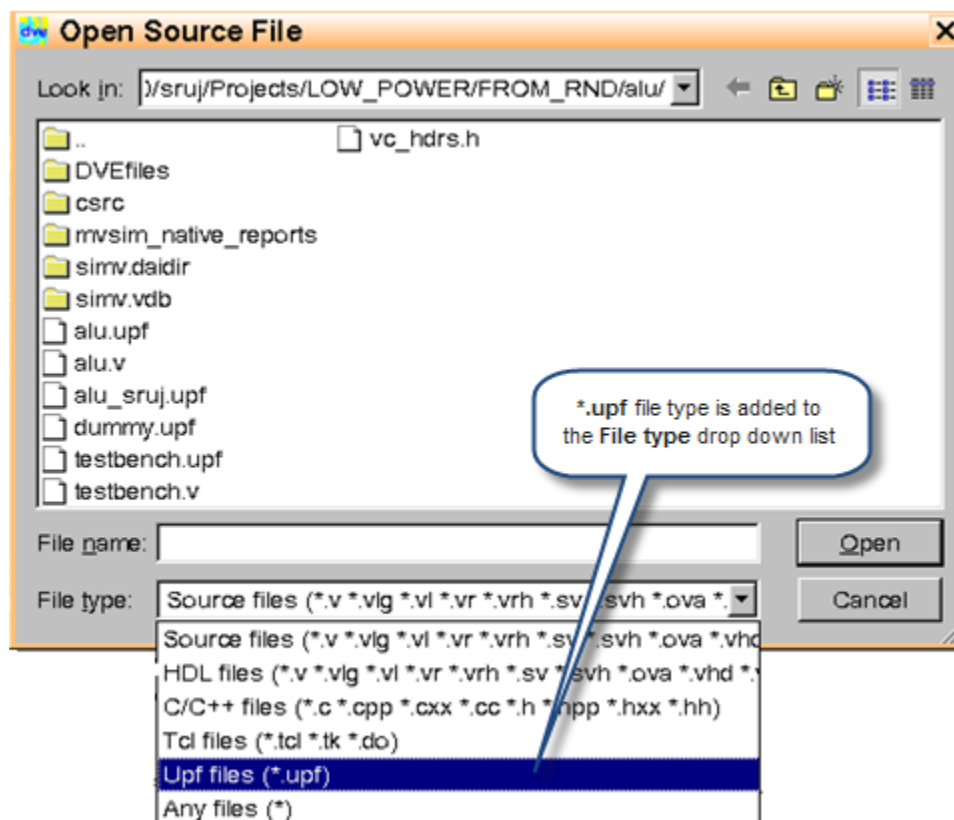
Figure 1-13 Isolation Strategy



UPF Source View

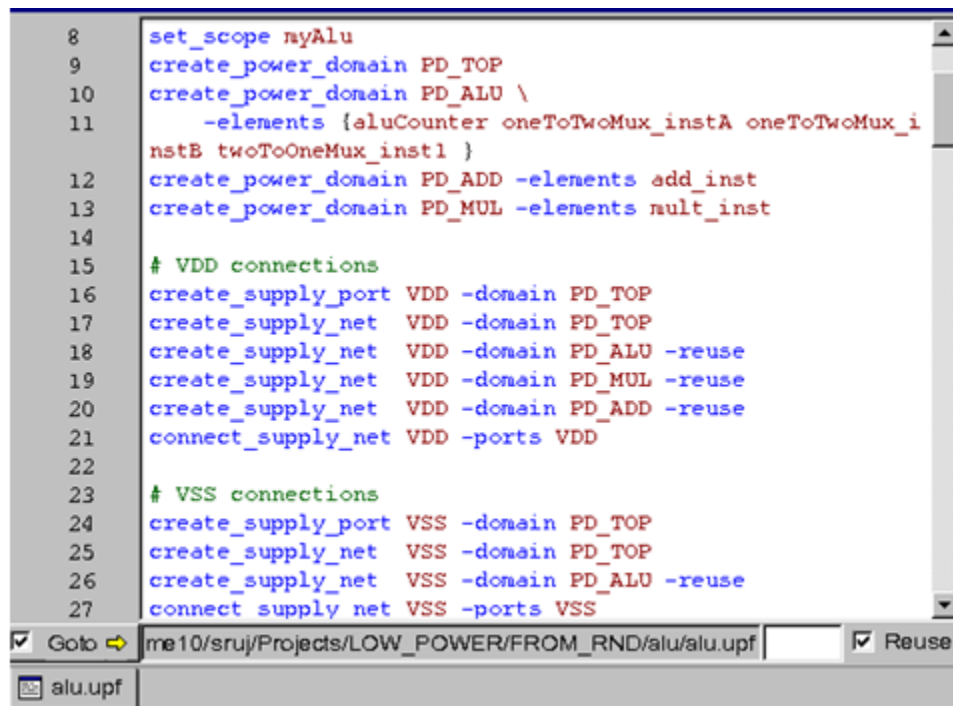
A new file type `Upf files (*.upf)` is added to **Open Source File** dialog box to open UPF files in Source Window.

Figure 1-14 Opening UPF Files in Source Window



DVE supports syntax highlighting/coloring of UPF keywords in *.upf files, as shown in the following figure:

Figure 1-15 Syntax Highlighting of UPF Keywords



```
8  set_scope myAlu
9  create_power_domain PD_TOP
10 create_power_domain PD_ALU \
11     -elements {aluCounter oneToTwoMux_instA oneToTwoMux_i
nstB twoToOneMux_inst1 }
12 create_power_domain PD_ADD -elements add_inst
13 create_power_domain PD_MUL -elements mult_inst
14
15 # VDD connections
16 create_supply_port VDD -domain PD_TOP
17 create_supply_net VDD -domain PD_TOP
18 create_supply_net VDD -domain PD_ALU -reuse
19 create_supply_net VDD -domain PD_MUL -reuse
20 create_supply_net VDD -domain PD_ADD -reuse
21 connect_supply_net VDD -ports VDD
22
23 # VSS connections
24 create_supply_port VSS -domain PD_TOP
25 create_supply_net VSS -domain PD_TOP
26 create_supply_net VSS -domain PD_ALU -reuse
27 connect supply net VSS -ports VSS
```

me10/sruj/Projects/LOW_POWER/FROM_RND/alu/alu.upf

al.u.upf

Reuse

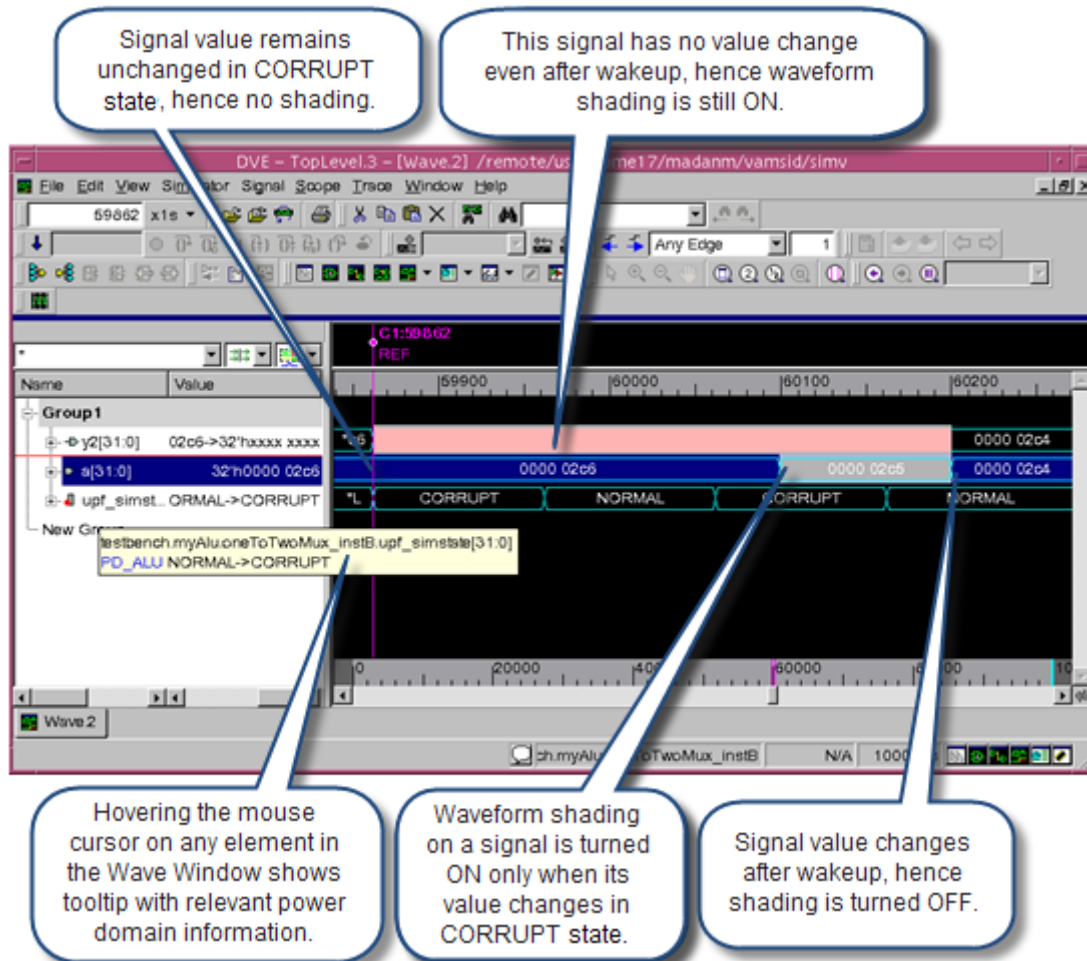
Waveform Shading

Waveform shading is used to distinguish if the value on a signal is the result of its corresponding power domain going to CORRUPT state.

- Waveform shading is turned ON if a value change is detected on a signal during CORRUPT state, indicating that the current value on the signal is an effect of the power being turned OFF.
- Waveform shading will remain ON till a value change is detected after the power domain goes to NORMAL state. It gives an indication that the signal has not yet been initialized upon wake up.
- No waveform shading will be seen if there is no value change detected during the CORRUPT state, thus indicating that there has been no effect of power OFF on this signal.

The following figure describes waveform shading in DVE:

Figure 1-16 Waveform Shading



DVE allows you to view the following in the Wave View:

- Isolation Shading
- Isolation Power Off Shading
- Input Toggle Shading

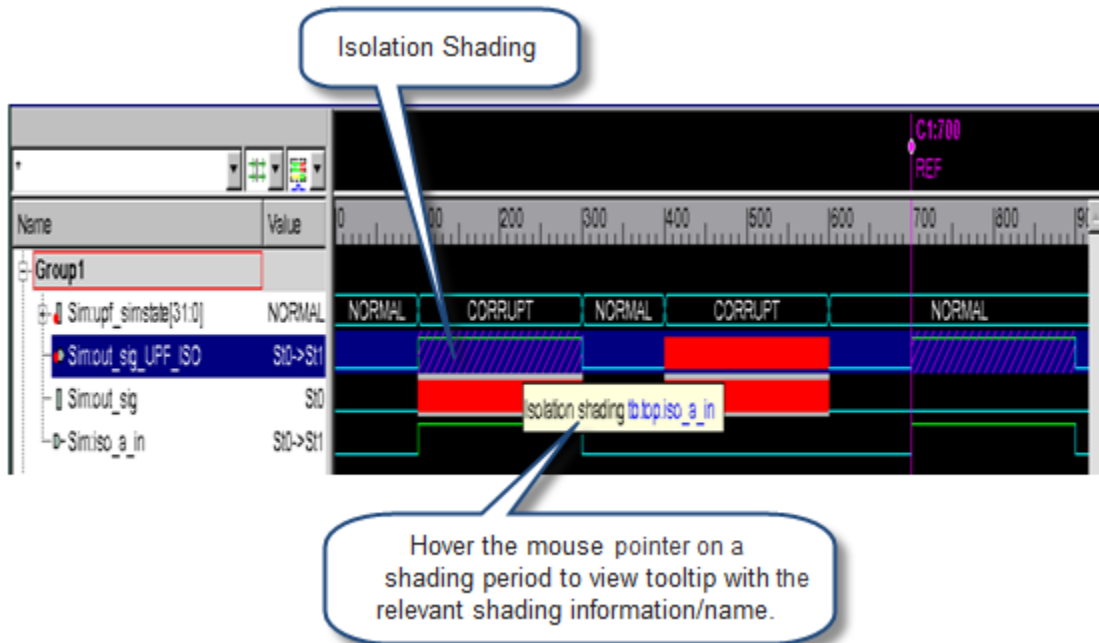
- `set_related_supply_net` (SRSN) Shading

Isolation Shading Support

DVE provides the waveform shading for isolated nets/ports in Wave View, as shown in [Figure 1-17](#). This allows you to distinguish whether the value changes on the net/port is due to the applied isolation.

Hover the mouse pointer on a shading period to view tooltip with the relevant shading information/name, as shown in [Figure 1-17](#).

Figure 1-17 Isolation Shading According to Isolation Control Signals

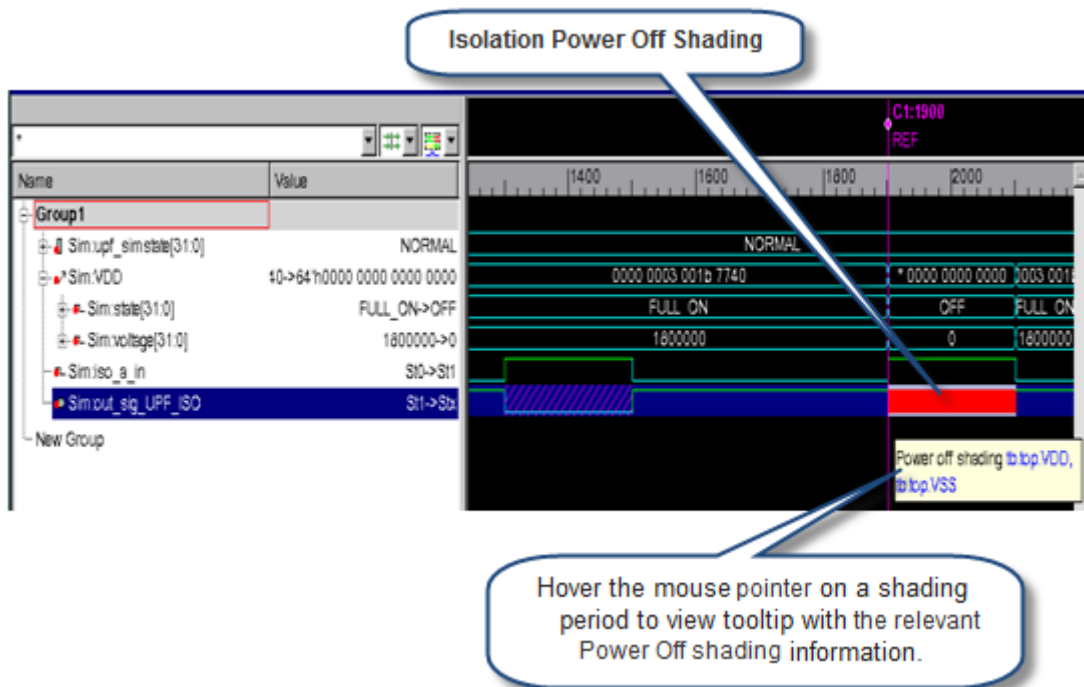


In the previous figure, `iso_a_in` is the isolation enabled with sense high. `out_sig` is isolated, therefore, isolation shading on `out_sig_UPF_ISO`.

Shading Support for the isolation_power_off Shading

DVE supports the shading of isolated nets/ports when the isolation power or ground nets are turned off, as shown in [Figure 1-18](#). This allows you to distinguish whether the value changes on an isolated net/port is due to the turning off of the isolation supplies.

Figure 1-18 Isolation Power Off Shading

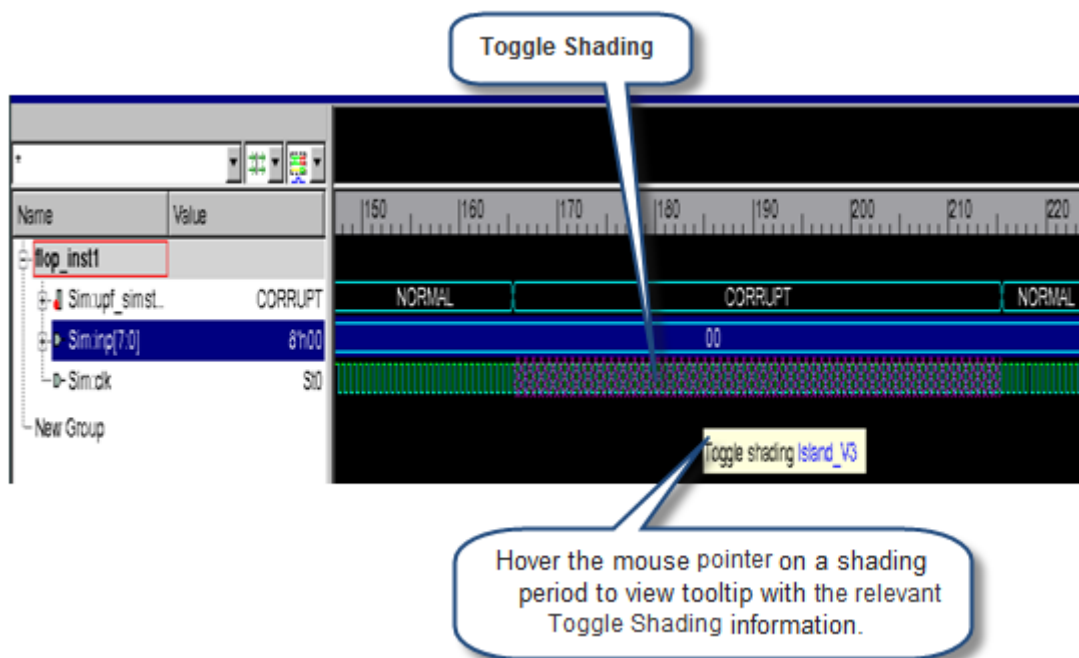


In the previous figure, VDD is the isolation power net for out_sig_UPF_ISO.

Support for Toggle Shading on Input Ports

DVE supports the shading of input ports when there is a toggle during the power off period, as shown in [Figure 1-19](#).

Figure 1-19 Toggle Shading for Input Ports



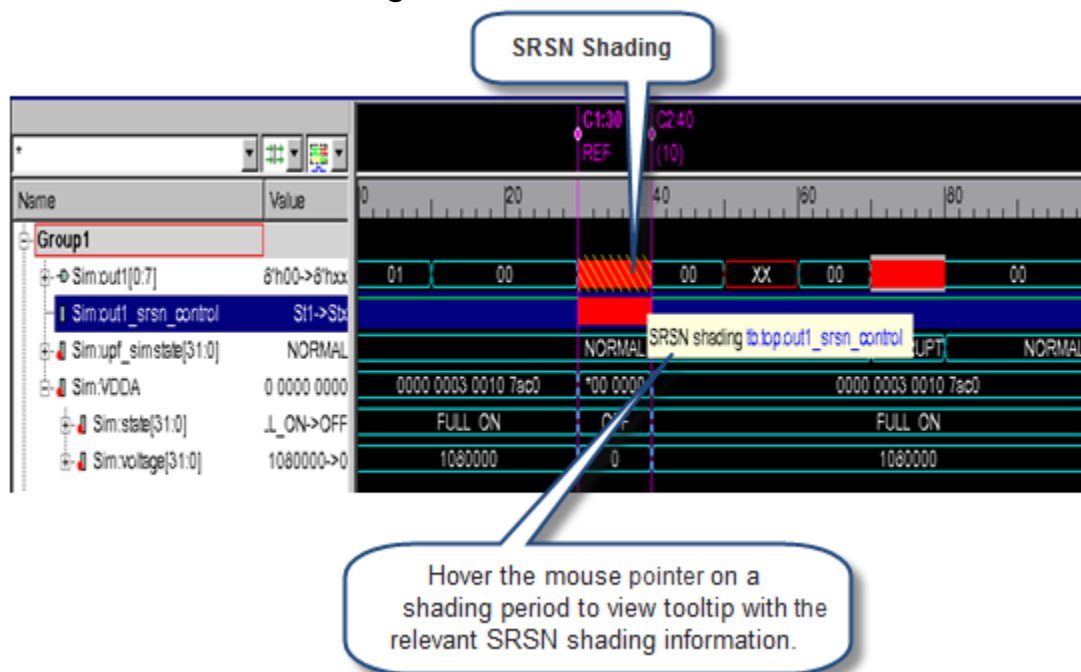
In the previous figure, `clk` is the input, which is toggling when the domain is in the `CORRUPT` state. No toggle on `inp`, therefore no shading.

Shading Support for `set_related_supply_net (SRSN)` on Ports

As part of the shading support, the output of the inserted SRSN buffer is shaded, as shown in [Figure 1-20](#), when the SRSN supplies go off.

The `*_srsn_control` signal indicates the state of SRSN supplies.

Figure 1-20 SRSN Shading



In the previous figure, VDDA is the related supply for out1. When VDDA is OFF, the SRSN shading is applied.

Note:

Waveform shading is enabled/disabled only if there is a value change on ports/nets.

Primary Power Off Shading



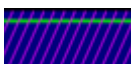

From this release, DVE does not support the corruption shading of input ports and elements marked as `always_on`.

Since NLP does not corrupt input ports and `always_on` elements explicitly, the Corruption shading has been disabled for these.

Configuring the Waveform Shading Pattern Color

You can set different color for a waveform shading pattern. [Table 1-11](#) lists default shading pattern colors of different types of waveform shading.

Table 1-11 Default Shading Pattern Color

Shading type	Default pattern color	Shading pattern
Power off shading	Gray	
SRSN shading	Yellow	
Isolation shading	Pink	
Toggle shading	Pink	

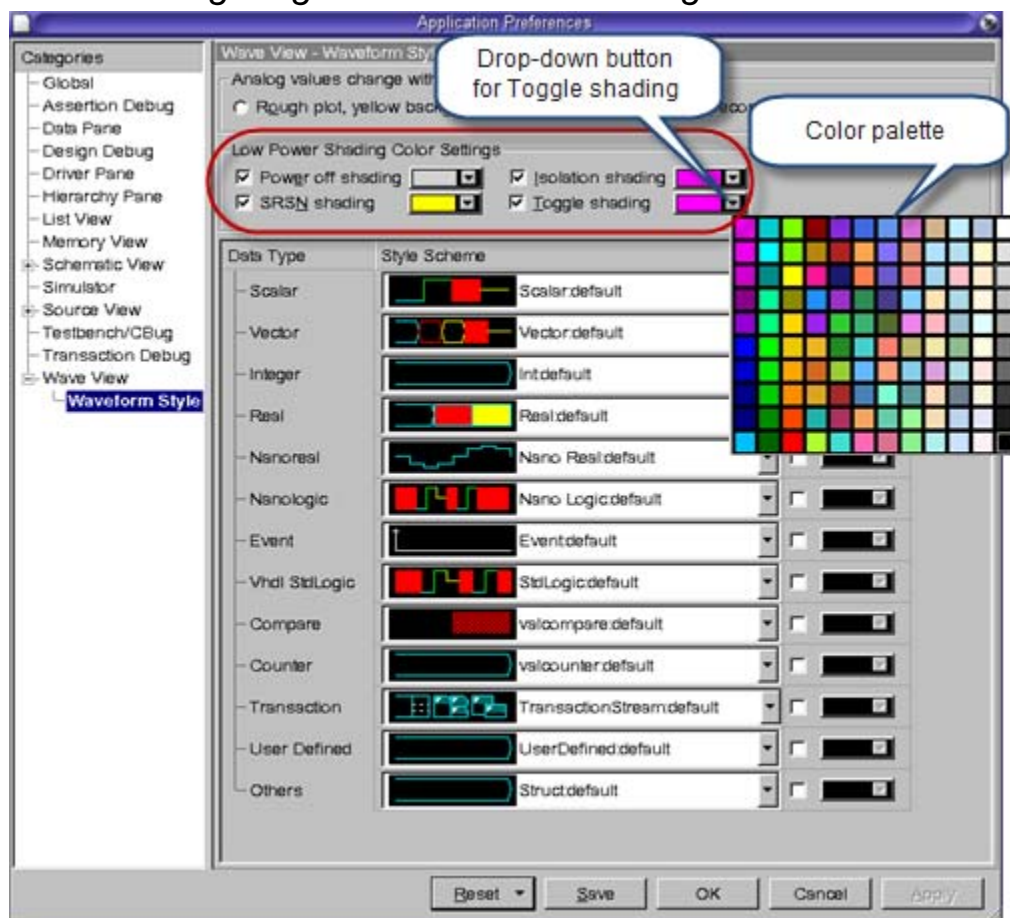
Perform the following steps to change the shading pattern color of a waveform shading:

1. Select **Edit > Preferences**.

The Applications Preferences dialog box appears.

2. In the **Wave View > Waveform Styles** category, **Low Power Shading Color Settings** region, click the drop-down button of a low power shading to open the color palette, as shown in [Figure 1-21](#).

Figure 1-21 Configuring the Waveform Shading Pattern Color



3. Select a color from the color palette and click **Apply**.
4. Click **OK**.

Limitations

DVE does not support the following features:

- The Signal Highlight feature when an isolation cell is part of the Schematic View
- Waveform shading for isolation with bit-select ports
- SRSN shading for bit/part select of array in `object_list`

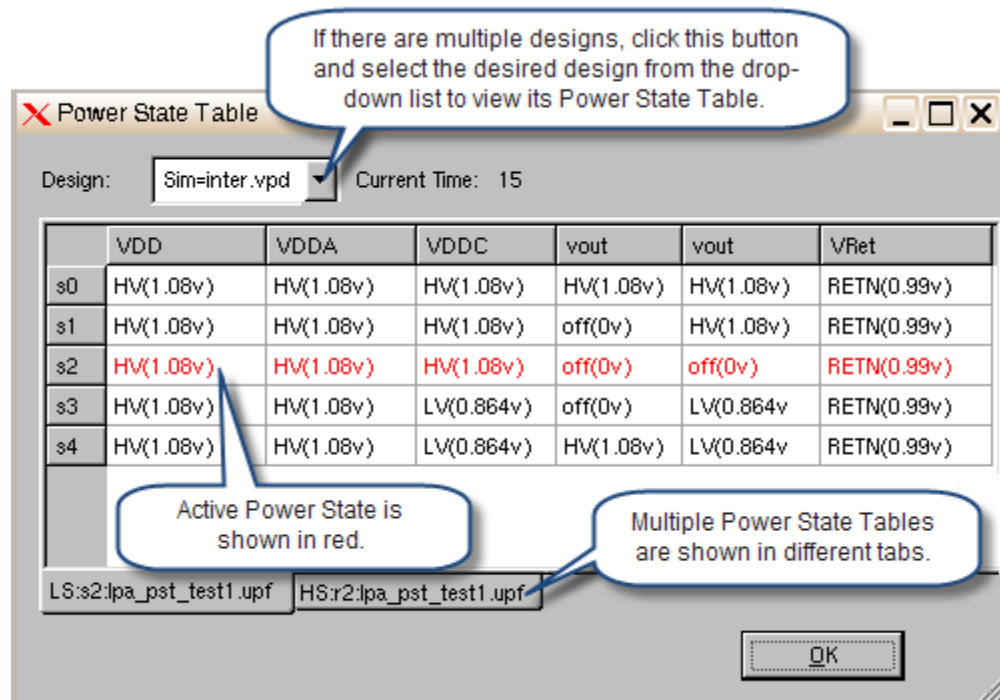
- Isolation shading when clamp type is latch
- Power Off shading and Input Toggle shading on ports of a cell (model with .db) is observed

Power State Table View

Use the Power State Table dialog box to view the power state tables of a design loaded in DVE. To open the Power State Table, click **View** > **Power State Table**. Figure 1-22 shows the Power State Table dialog box.

If there are multiple designs, you can use the **Design** drop-down menu and select the desired design to view its power state table. In case of multiple PSTs, each PST is shown in a different tab.

Figure 1-22 Power State Table

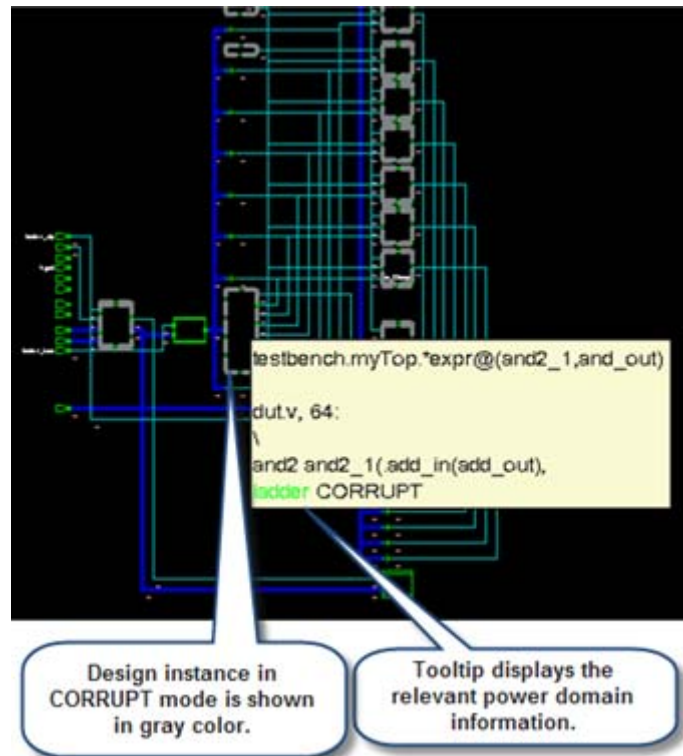


Shading in Schematic View

The Schematic Window automatically shades the design instances in gray when corresponding power domain is in `CORRUPT` state. It helps you to view the states of source and sink power domains of a crossover.

Hovering the mouse cursor on a design instance displays tooltip with relevant power domain information, as shown in the following figure:

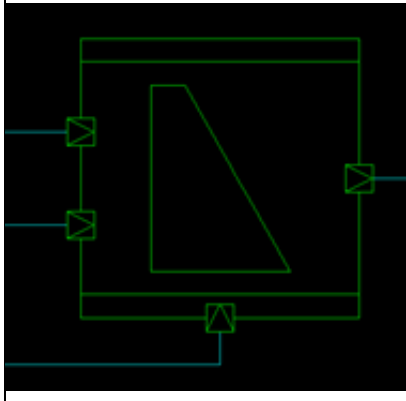
Figure 1-23 Schematic Window



Symbol for Isolation Cell in Schematic

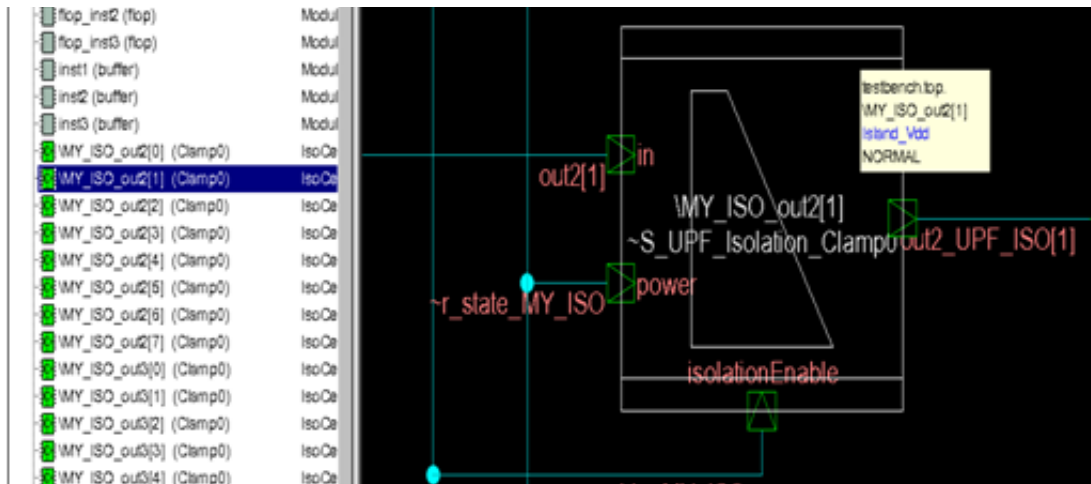
In the previous versions of DVE, isolation cell in schematic views (for Verilog portions of the design only) are represented as rectangular blocks. DVE now provides a new schematic symbol (shown in [Table 1-12](#)) to represent isolation cells.

Table 1-12 Isolation Symbols in the Schematic View

Symbol	Symbol Name
	Isolation Cell

[Figure 1-24](#) shows the previous isolation cell in the Schematic View.

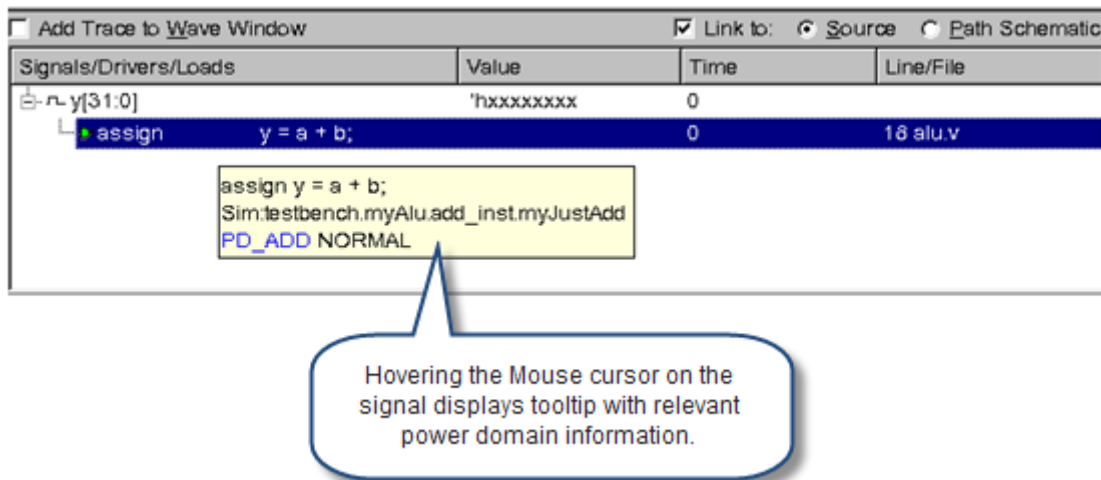
Figure 1-24 Isolation Cells in the Schematic



Power Domain Information for Drivers/Loads

The Drivers/Loads Pane displays all drivers/loads that possibly contributed to a signal value. Hovering the mouse cursor on the signal displays its power domain information in a tooltip, as shown in the following figure:

Figure 1-25 Drivers/Loads Pane



Coverage Support for Low Power Objects

You can use the `-power=coverage` compile-time option to automatically create covergroups for low power objects based on the power intent. The covergroups are created for the following low power objects:

- PST states/transitions
- Power Switch State 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 connected SupplySet
- Supply Set Power State 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`

You can use the `describe_state_transition` UPF command to describe the desired state transitions or sequences.

By default, MVSIM native mode assumes that all state transitions between legal and user-defined states are valid transitions. You can also use this command to constraint the set of valid transitions and to define illegal transitions.

Note:

If `describe_state_transition` is specified on an object, then the transitions are not generated automatically.

Syntax

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

Options

transition_name

Name of the set of transitions for the object defined using this command.

Object

MVSIM native mode supports following objects:

- PST
- Power Switch — on/on_partial/error/off states are considered.

- Supply Set — For Supply Sets, MVSIM native mode 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`.
- Supply Net — These are states defined using `add_port_state` 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.
- Supply Port

`-from, -to`

These options take a list of states. 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`

If `-from` and `-to` is used, then `-through` can be used to define intermediate states between `-from` and `-to` states to create a sequence. This is a MVSIM native mode specific option to capture a state sequence.

`-paired`

Takes a paired list of transitions.

`-illegal | -legal`

This option is used to mark the transitions defined in the command as illegal/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`.

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.

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: $\{\{s1, s4\} \{s2 s4\} \{s3 s4\} \{s5 s4\}\}$

Coverage Reports

DVE Coverage Report

DVE coverage allows you to group all the low power coverage items together in DVE, so that it does not interfere with user-defined functional covergroups. Perform the following steps to view the DVE Coverage GUI, as shown in [Figure 1-26](#).

1. Perform the below command:

```
% dve -cov -dir simv.vdb &
```

2. Select **File > Import User Defined Groups**
3. Double-click the **mvsim_native_reports** folder in the **Import User Defined Groups** dialog box, select **lp_covergroups.tcl**, and then click **Load** to view the DVE Coverage GUI.

Figure 1-26 DVE Coverage GUI with Low Power Cover Groups

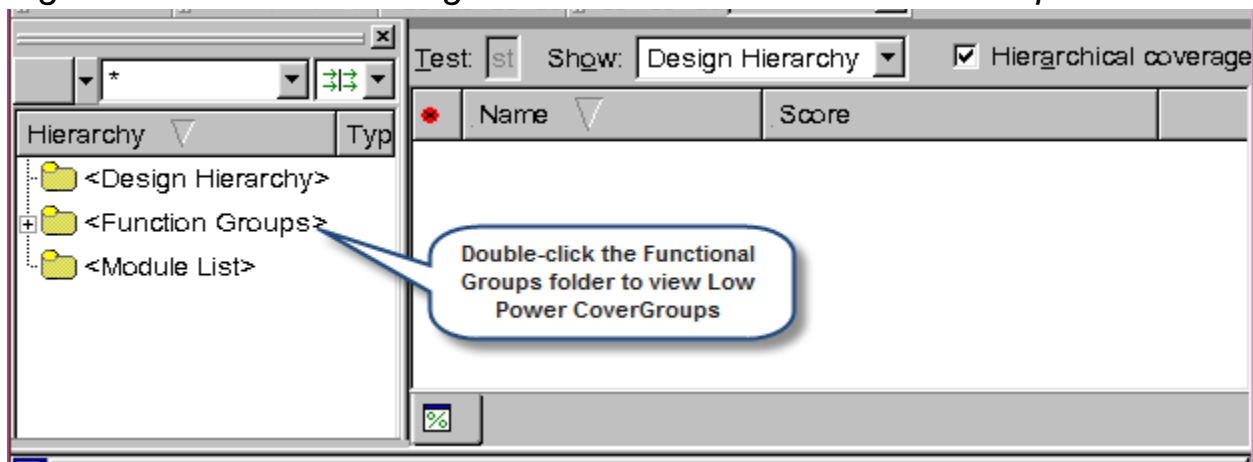
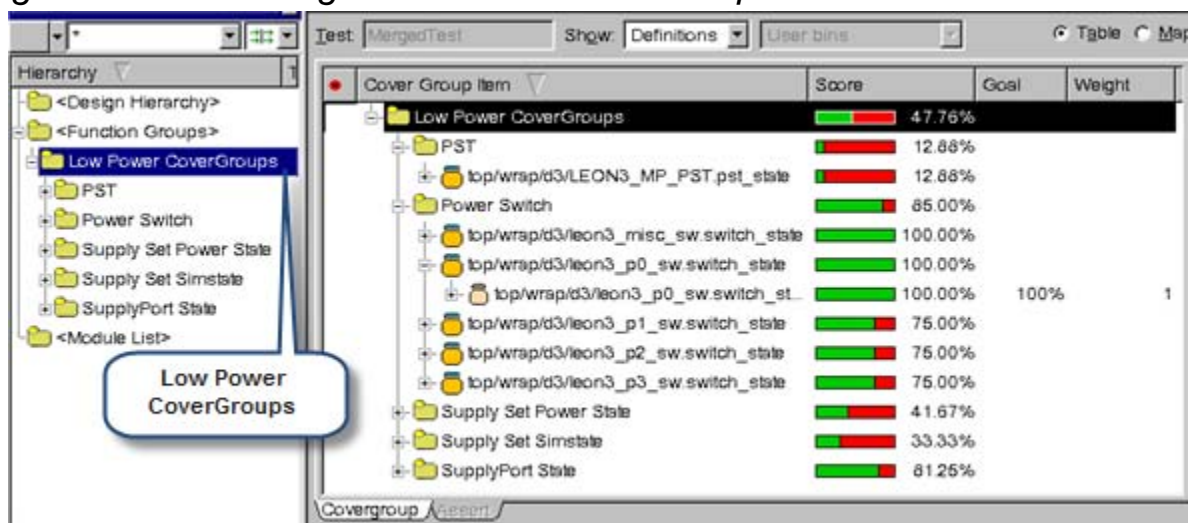


Figure 1-27 Viewing Low Power Cover Groups



Click a transition in Hierarchy Pane to view its definition in Source View, as shown in Figure 1-28.

Figure 1-28 Cross-linked with Definition in the UPF File

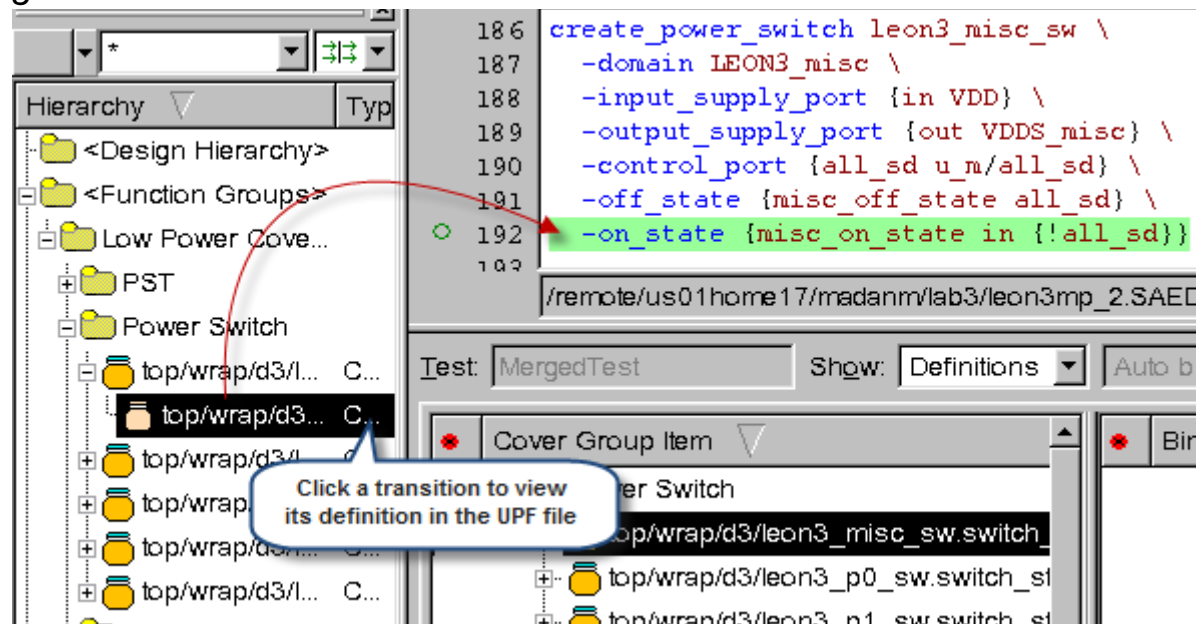


Figure 1-29 shows the PST states in DVE. One bin per PST state is created by default. Illegal Bin is created for Undefined/Illegal PST State.

Figure 1-29 Default States/Transitions – PST States

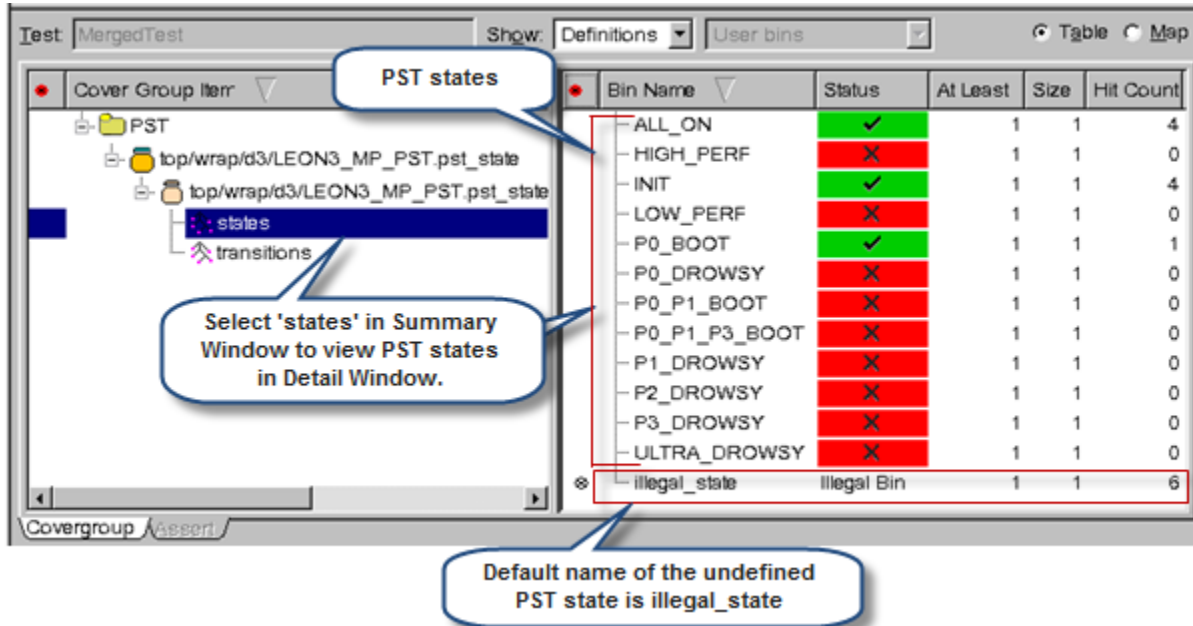
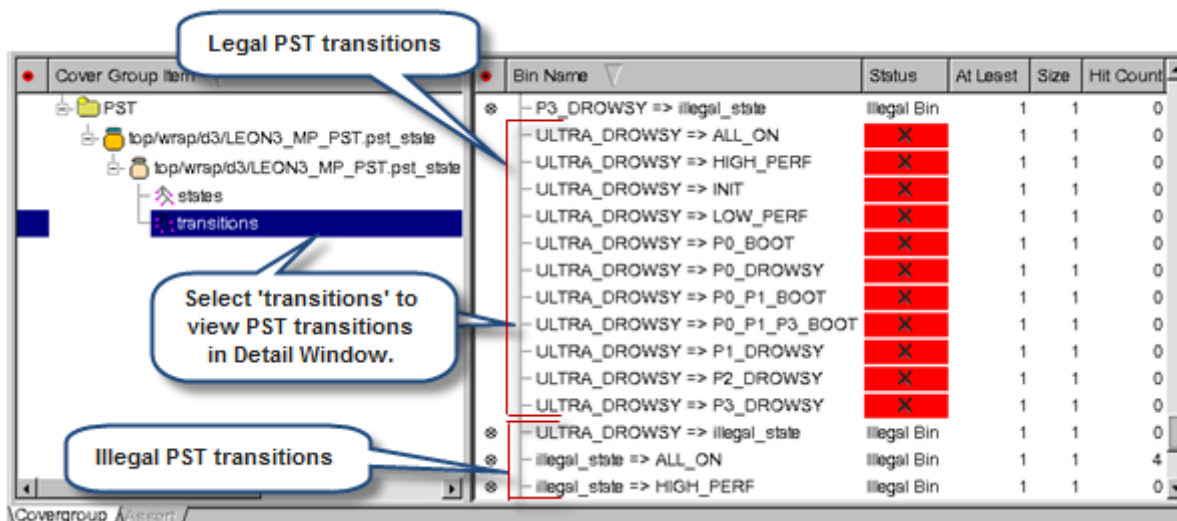


Figure 1-30 shows the PST transitions. One bin per legal state transition is created by default. Illegal Bin is created for transition to/ from Undefined/Illegal PST State.

Figure 1-30 Default States/Transitions – PST Transitions



URG Report

Perform the below command to view URG report:

```
% urg -dir simv.vdb
```

```
% mozilla $cwd/urgReport/dashboard.html &
```

Figure 1-31 shows the sample URG report.

Note:

Grouping of low power coverage items is not supported in URG.

Figure 1-31 Sample URG Report

SCORE	WEIGHT	GOAL	NAME
100.00	1	100	test_top/DUT/LEON3_p3.primary.simstate
100.00	1	100	test_top/DUT/LEON3_p1.primary.simstate
100.00	1	100	test_top/DUT/LEON3_p2.primary.simstate
100.00	1	100	test_top/DUT/leon3_misc_sw.switch_state
100.00	1	100	test_top/DUT/LEON3_p0.primary.simstate
100.00	1	100	test_top/DUT/leon3_p0_sw.switch_state
100.00	1	100	test_top/DUT/leon3_p1_sw.switch_state
100.00	1	100	test_top/DUT/leon3_p2_sw.switch_state
100.00	1	100	test_top/DUT/leon3_p3_sw.switch_state
100.00	1	100	test_top/DUT/LEON3_misc.primary.simstate
40.00	1	100	test_top/DUT/LEON3_misc.primary.power_state
40.00	1	100	test_top/DUT/LEON3_p0.primary.power_state
37.50	1	100	test_top/DUT/LEON3_p1.primary.power_state
37.50	1	100	test_top/DUT/LEON3_p2.primary.power_state
37.50	1	100	test_top/DUT/LEON3_p3.primary.power_state
25.00	1	100	test_top/DUT/LEON3_p1.default_retention.simstate
25.00	1	100	test_top/DUT/TOP.default_retention.simstate
25.00	1	100	test_top/DUT/LEON3_p2.default_retention.simstate

Usage Example of describe_state_transition

Consider the below 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 1-32 shows PST states for the above PST description.

Figure 1-32 PST States

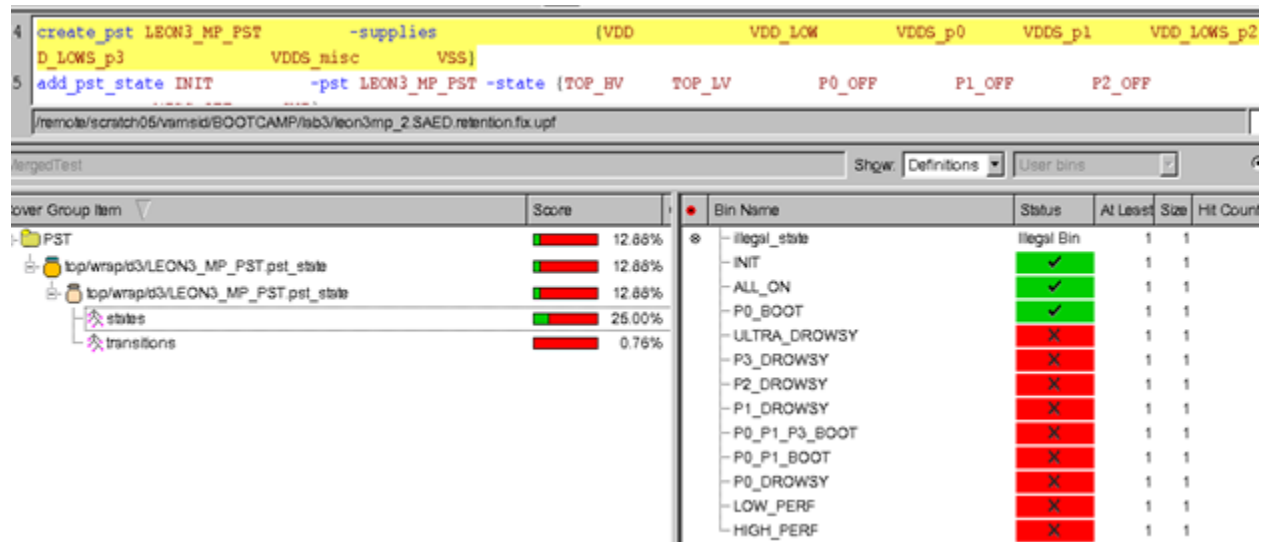
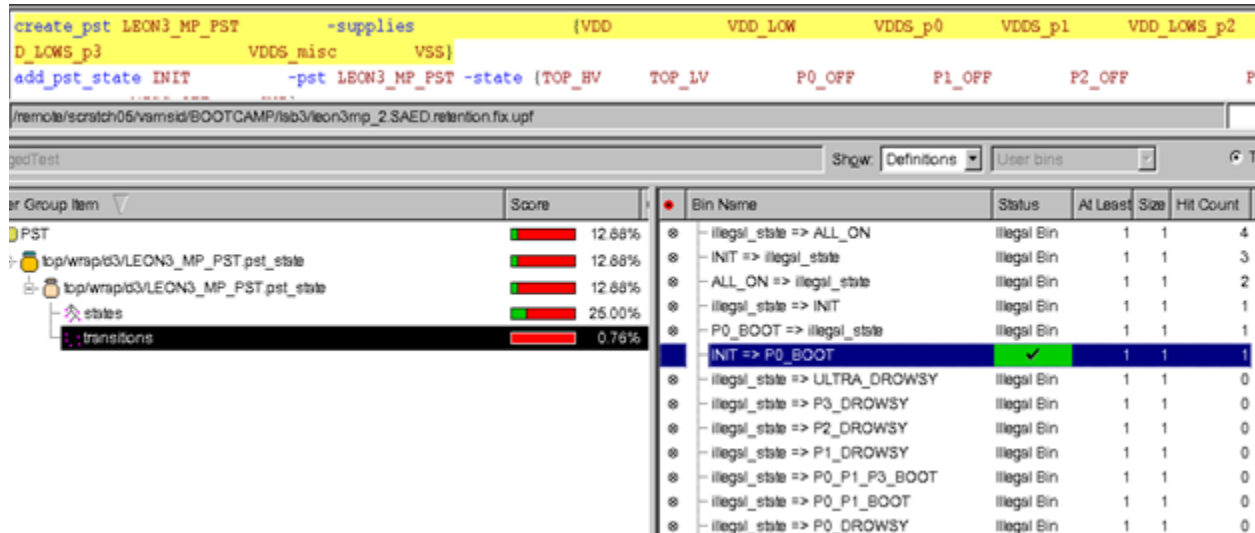


Figure 1-33 shows PST transitions for the above PST description.

Figure 1-33 PST Transitions



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 1-34) 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 1-34 *describe_state_transition* – Marking a Transition Illegal

Cover Group Item	Score	Goal	Bin Name	Status	At Least	Size	Hit Count
PST	12.50%		-INIT => P0_BOOT	Illegal Bin	1	1	1
top/wrap163/LEON3_MP_PST.pst_state	12.50%						
top/wrap163/LEON3_MP_PST.pst_state	12.50%						
states	25.00%						
transitions	0.00%						

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 1-35](#).

```
describe_state_transition pst_seq \
  -object LEON3_MP_PST \
  -from {INIT} -to {ALL_ON} -through {P0_BOOT} \
  -legal
```

Figure 1-35 *describe_state_transition* – Generating a Sequence

Cover Group Item	Score	Goal	Weight	Bin Name	Status	At Least	Size	Hit Count
PST	12.50%			-INIT => P0_BOOT => ALL_ON	X	1	1	0
top/wrap163/LEON3_MP_PST.pst_state	12.50%							
top/wrap163/LEON3_MP_PST.pst_state	12.50%	100%	1					
states	25.00%	100%	1					
transitions	0.00%	100%	1					

Guidelines for Migrating to MVSIM Native Mode from MVSIM

This section describes the guidelines you need to follow for migrating from MVSIM to MVSIM native mode.

This section contains the following topics:

- [“Migrating from MVSIM to MVSIM Native Mode”](#)
- [“Specifying Power Top for Simulations”](#)
- [“Using Supply Functions”](#)
- [“Mapping MVSIM archpro.ini Settings to MVSIM Native Mode”](#)
- [“Using Wildcard Characters”](#)
- [“Managing Corruption in MVSIM Native Mode”](#)

Migrating from MVSIM to MVSIM Native Mode

Perform the following steps to migrate from MVSIM to MVSIM native mode:

1. Run the vector without the `-upf` switch on the MVSIM native mode release. This step takes care of any VCS migration issues.
2. Run the vector with the `-upf` switch in an always-on mode and comment power switches in UPF. This step makes sure that the internal instrumentation is correct.
3. Run the vector with intended power activities like shutdown corruption, and so on.
4. Debug by passing MVSIM logs and dump, if available.

Specifying Power Top for Simulations

In MVSIM, the power top for simulations is specified using the `mvdbgen -top` switch. In MVSIM native mode, 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` switch, as shown in the following table:

Table 1-13 Power Top for Simulations in MVSIM native mode

MVSIM	MVSIM native mode
<code>mvdbgen -top dut_top</code>	<code>set_design_top testbench/../../ dut_top</code>
<code>mvdbgen -top dut_top</code>	<code>-power_top dut_top</code>

Using Supply Functions

These functions are called in the testbench to drive the supply pads in the supply network. Only supply pads, essentially supply ports created at the power top, can be driven. Real or reg-based voltages are not supported. The following table describes the usage of supply functions in MVSIM native mode:

Table 1-14 Using Supply Functions in MVSIM native mode

Behavior	MVSIM	MVSIM native mode
Activate	<code>VDD =1.2 ;</code>	<code>supply_on("VDD",1.2);</code>
Shutdown	<code>VDD =0.0 ;</code>	<code>supply_off("VDD");</code>

Note:

- You must specify `import UPF::*;` in the module in which supply functions are being used.

- You must use `supply_off("VDD")` to switch OFF a domain in MVSIM native mode. The `supply_on("VDD", 0.0)` function call switches OFF a domain in MVSIM, but not in MVSIM native mode.
- You can use the below `set_design_attributes` command in the UPF or runtime config file(-power <config_file>), to switch OFF the domain when `supply_on("VDD", 0.0)` is called on a primary power of the domain.

```
set_design_attributes -attribute
SNPS_voltage_based_supply_on TRUE
```

Handling VRMs in State-based sim

You must modify the Voltage Regulator Models(VRMs) using the `supply_on` or `supply_off` functions. You must call the `supply_off` function after the voltage is ramped to 0.0, and not when the control signal is changed. Similarly, you must call the `supply_on` function during voltage ramp-up.

Mapping MVSIM archpro.ini Settings to MVSIM Native Mode

In MVSIM, all settings are mentioned in the archpro.ini file. In MVSIM native mode, these settings are mapped either to UPF commands or VCS command-line options, as given in the following sections:

- [“Mapping archpro.ini Settings to Corresponding MVSIM Native Mode Usage”](#)
- [“Mapping archpro.ini Settings to Corresponding VCS Command-line Options in MVSIM Native Mode”](#)

Mapping archpro.ini Settings to Corresponding MVSIM Native Mode Usage

This section describes the mapping of archpro.ini settings to corresponding usage in MVSIM native mode.

always_on_modules

Use this setting to specify modules that need to be always ON.

MVSIM	MVSIM native mode
always_on_modules	set_design_attributes

Use Model:

```
set_design_attributes -models <MODULE_NAME> -  
attribute UPF_dont_touch TRUE
```

allow_upf_edge_retention

Use this setting to specify edge sensitive retention in UPF.

MVSIM	MVSIM native mode
allow_upf_edge_retention	Default

Custom Retention Settings

Use this setting to model custom retention schemes.

MVSIM	MVSIM native mode
retention_scheme_c*rff	-assert_*_mutex
retention_scheme_sam_c*rff	-save_condition/ -restore_condition/ -retention_condition

disable_30_70

Use this setting to disable 30-70 conversion in PROTECTED simulation mode.

MVSIM	MVSIM native mode
disable_30_70	Default

module_always_on_signals

Use this setting to specify signals that need to be always ON.

MVSIM	MVSIM native mode
module_always_on_signals	set_design_attributes

Use Model:

```
set_design_attributes -elements {reg1 reg2 reg3} -  
models <MODULE_NAME> -attribute UPF_dont_touch TRUE
```

Where, reg1, reg2, and reg3 are variables in the MODULE_NAME module.

reinit_elements.

MVSIM	MVSIM native mode
reinit_elements	set_design_attributes

Use Model:

```
set_design_attributes -models <MODULE_NAME> -  
attribute SNPS_reinit TRUE
```

For more information on reinit_elements, see [“Initial Block Retriggering”](#).

Mapping archpro.ini Settings to Corresponding VCS Command-line Options in MVSIM Native Mode

This section describes the mapping of archpro.ini settings to corresponding VCS command-line options in MVSIM native mode.

consider_contassigns_as_pass_through

Use this setting to ignore continuous assigns in crossover detection.

MVSIM	MVSIM native mode
consider_contassigns_as_pass_through	-power=always_on

Note:

This VCS command-line option is applicable for the Verilog portion of the design.

force_edge_sensitive_retention

This setting forces the retention to be edge based, even if level sensitive retention is specified in UPF.

MVSIM	MVSIM native mode
allow_upf_edge_retention	-power=ret_level_as_edge

simulation_mode

Use this setting to switch between transparent and protected modes.

Behavior	MVSIM	MVSIM native mode
transparent	simulation_mode=TRANSPARENT	Not Supported
protected	simulation_mode=PROTECTED	Default

Using Wildcard Characters

MVSIM native mode supports the `find_objects` UPF command for wildcard character usage in UPF. The following table describes the usage in MVSIM and MVSIM native mode:

TOOL	Usage
MVSIM	<code>create_power_domain ISLAND_VDD -elements {U1*}</code>
MVSIM native mode	<code>set x [find_objects. -pattern {U1*} -object_type inst -transitive TRUE] * create_power_domain ISLAND_VDD -elements \$x</code>

Managing Corruption in MVSIM Native Mode

You can set the environment variable `APF_COMPAT` at runtime with MVSIM-MVSIM native mode to mimic MVSIM-PLI like corruption.

Use `setenv APF_COMPAT 1` to:

- Corrupt integers to 0 instead of `x`
- Skip corruption of reals
-

Differences between MVSIM and MVSIM Native Mode

The following points describe the differences in behavior between MVSIM and MVSIM native mode:

- Design object names must use ‘/’ as hierarchy delimiter in MVSIM native mode. This is consistent with rest of the Eclipse flow. MVSIM allows both ‘.’ and ‘/’ to be used as hierarchy delimiter. MVSIM native mode does not support ‘.’ as hierarchy delimiter in UPF.

Note:

You must use ‘/’ as a hierarchy delimiter for generate instance names in MVSIM native mode.

- Design scope names must use ‘[]’ when specifying generate statement scopes in MVSIM native mode. MVSIM allows both ‘[]’ and ‘()’ to be used when specifying generate statement scopes in the hierarchical paths.
- MVSIM native mode generates an error, if you use the supply-net name which is similar to the existing supply-net name in RTL.
 - For MVSIM native mode, you must use `-reuse` in UPF to indicate the reuse of an existing net. MVSIM implicitly connects the two nets together and drives the UPF net using the RTL net.
- For constants in port map, MVSIM native mode corrupts ‘b0’ only when `VSS` is OFF, and not when `VDD` is OFF. MVSIM native mode corrupts ‘b1’ only when `VDD` is OFF, and not when `VSS` is OFF. MVSIM does not corrupt constant port maps.
 - Use the `-power=apfcompat` compile-time option in MVSIM native mode to mimic MVSIM behavior.

Gate-level Netlist Simulations (Simulating with Liberty (.db) Files)

In MVSIM native mode, gate-level netlist simulations are not the same as RTL simulations. In addition to the design (gates and models) and the UPF, MVSIM native mode can also read in some special attributes about the cells from their .db representations. The following sections describe gate-level netlist simulations (GLS) with MVSIM native mode:

- [“Using Connect Supply Net and Library Mapping”](#)
- [“Supported .db Attributes”](#)
- [“Power Aware and Non-Power Aware Models”](#)
- [“Simulation Behavior for Power Aware Models”](#)
- [“Implicit Supply Net Connections”](#)
- [“Value Conversion Tables \(VCTs\)”](#)
- [“Multi-rail Macro Simulation”](#)
- [“Limitations of Gate-level Netlist Simulations”](#)

Using Connect Supply Net and Library Mapping

Library mapping is used to determine a matching library cell for a Verilog module-based architecture. A module is considered as a cell, if a matching library cell for the module exists in a specified database.

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.

Use Model

Use the following search path and link library command in the configuration file. You can specify this file as an argument to the `-power_config <file_name>` compile-time option or specify the model name from the UPF command.

```
db_search_path = {path}
```

```
db_link_library = {.db files}
```

Use the following command to run the testcase:

```
vcs -upf out.upf -power_config lp_config
```

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. MVSIM native mode propagates the state and voltage values of the UPF supply net to HDL supplies based on the connection.

Supported .db Attributes

MVSIM native mode supports the following .db attributes. For more information on these attributes, see the *Library Compiler LRM* (Liberty Format). These attributes help MVSIM native mode to accurately simulate the power-aware behavior of the netlist and UPF.

- **Logic Pins** — These are the input and output ports of the cells, the same as the ports listed in the model of the cell. MVSIM native mode expects the logic pins of the .db and the model to match exactly. This is how MVSIM native mode maps a cell model with the appropriate .db. If the module names match but the logic pins do not, MVSIM native mode 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 be 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** — This 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 MVSIM native mode supports:

- always_on cells
- isolation cells
- level shifter cells
- retention cells
- power switch cells

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 PG pins are present in a model, they must exactly match what is present in the .db representation. Otherwise, MVSIM native mode treats it as a non-power aware model.

Note:

- MVSIM native mode does not correctly recognize PG pins of type internal.
- MVSIM native mode does not use PG pins for type nwell, pwell, deepnwell, and deeppwell.

In a power aware model, the PG pins can be:

- input/output ports of the model
- reg/wire
- supply_net_type
- supply0/supply1 type

MVSIM native mode uses standard VCTs, as defined in the LRM, for conversion between various types. The default MVSIM native mode 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, MVSIM native mode 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, MVSIM native mode corrupts the outputs of the cell based on the power-down function. If there is no power-down function, MVSIM native mode uses the `related_pg_pin` attribute to corrupt the ports. You can override the above default behavior by using the following `set_design_attributes` commands:

- To force MVSIM native mode to corrupt the ports for all cells even when the models are power aware:

```
set_design_attributes -attribute \  
SNPS_override_pbp_corruption TRUE
```

- To force MVSIM native mode not to corrupt the ports for all cells even when the models are non-power aware:

```
set_design_attributes -attribute \  
SNPS_override_pbp_corruption FALSE
```

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, then 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 (ISOs and ELSes) and retention cells are exceptions to this rule. If the UPF contains isolation policies, MVSIM native mode tries to match isolation policies with the implemented ISO and ELS cells in the netlist. Once an ISO cell is matched with a policy, the `isolation_power_net` and the `isolation_ground_net` specified in the policy are implicitly connected to the PG pins of the ISO cell.

If there is an explicit CSN, that connectivity is supported. Similarly, if there is a retention policy, MVSIM native mode tries to associate the retention cell with a policy. If there is no explicit CSN, the rails in the policy implicitly drive the backup PG pins of the retention cell.

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 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 ports in the same scope, the direction of the verilog port is input
 - For 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 ports in the same scope, the direction of the verilog port is output
- For 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 connections from UPF to an HDL model or from the HDL model to 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
- 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 1-15 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

Table 1-15 Supported VCTs

	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
	UPF2SV_LOGIC ^	FULL_ON - 1 OFF - 0 PARTIAL_ON - X UNDETERMINED - X
	UPF_GNDZERO2SV_LOGIC ^^	FULL_ON - 0 OFF - 1 PARTIAL_ON - X UNDETERMINED - X

* **SV_LOGIC2UPF** is the default HDL2UPF VCT used for power ports.

** **SV_LOGIC2UPF_GNDZERO** is the default HDL2UPF VCT used for ground ports.

^ **UPF2SV_LOGIC** is the default UPF2HDL VCT used for power ports.

^^ **UPF_GNDZERO2SV_LOGIC** is the default UPF2HDL VCT used for ground ports.

Attributes Supported for Overriding the Default VCTs

Table 1-16 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 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

- Defining custom VCTs is not supported.
- For HDL2UPF, `-vct` with `connect_supply_net` cannot be used for supply net connections to a DB matched cell port.

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 UPF supply ports to HDL supplies.

Use Model

Setup

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 MRM cells in the `-power_config` file, which is similar to `c_s_n` and library mapping feature.

- UPF compliant liberty DB's for all MRM's.

Note:

Only liberty DB's are supported. Library files(`.lib`) are not supported.

- UPF with `connect_supply_net` statements for 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 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 of 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 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 `c_s_n` and library mapping.

Note:

DB matching is done for Verilog portion of the design only.

Limitations of Gate-level Netlist Simulations

Note the following limitations:

- Simulation of an ICC-generated netlist and UPF is not supported. That is, if there are power switches instantiated in the design, then MVSIM native mode does not simulate correctly.

- Policy to cell association is not supported. For instance, MVSIM native mode may instrument isolation even if an ISO cell is present. Moreover, the supplies to the ISO cell may not be driven correctly. MVSIM native mode dumps the isolation association report in the `mvsim_native_reports` or directory for debug.
- Timing simulation (sdf) is not supported.
- Liberty version 11.09 is not supported (10.12 is supported).
- If the .db has attributes on the bits of a bus or bundle, then MVSIM native mode does not handle it correctly.

Synopsys Low Power Flow Consistency

This section consists of the following topics:

- “`-simulation_only` Support”
- “Synopsys Low Power Flow Consistency Check”
- “Handling of Constants for Corruption and Isolation”
- “Error Checking for Isolation Strategy with `-location parent` on Top-level Ports”
- “Error for Conflicting Port State Names in the `add_port_state` Command”

-simulation_only Support

With the `-power=sim_only` compile-time option, the below list of UPF commands/options/attributes will result in parsing error, if they are not part of the UPF file that is loaded with `load_upf <upf_file> -simulation_only`.

Error code for UPF command/option:

Error- [UPF_UMFCO] Unsupported mode for Command/
option

Error code for attribute:

Error- [UPF_UMFCA] Unsupported mode for Attribute

Table 1-17 Options Supported with -simulation_only

Commands	Options/Attributes
bind_checker	All options and custom options
load_upf	-model
create_power_domain	-simulation_only
set_design_attributes	Simulation-only: UPF_dont_touch SNPS_override_pbp_corruption SNPS_reinit SNPS_dont_reinit Non simulation-only: (SNPS_default_power_upf2sv_vct, SNPS_default_ground_upf2sv_vct, SNPS_default_power_upf2vh_vct, SNPS_default_ground_upf2vh_vct, SNPS_default_power_sv2upf_vct, SNPS_default_ground_sv2upf_vct, iso_source, iso_sink, related_supply_default_primary)

Synopsys Low Power Flow Consistency Check

You can use the `-power=snps_lp_flow` compile-time option to maintain consistency of UPF across the Synopsys Low Power Flow. With this option, the below list of UPF commands/options will result in an error as these are not supported across all the tools in Synopsys Low Power Flow.

Use Model

```
%vcs <vcs_options> -upf <upf_file> -  
power=snps_lp_flow
```

Error code for UPF command/option:

```
Error- [UPF_UMFCO] Unsupported mode for Command/  
option
```

Table 1-18 Options that will error out with `-power=snps_lp_flow`

Commands	Options
add_power_state	-update
set_isolation	-clamp_value Z
set_isolation_control	-location automatic
set_level_shifter	-source, -sink
set_retention_control	-save_signal/-restore_signal <posedge/ negedge>
set_simstate_behaviour	

Handling of Constants for Corruption and Isolation

MVSIM native mode 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.

- [Default Behavior](#)
- [Supply Based Corruption of Constants](#)
- [Avoid Corruption for Constants in Continuous Assignments](#)
- [Skip Corruption of Constants](#)

Default Behavior

All literal constants in port map, or all variables that are assigned to literal constants using continuous assign statements, are treated as if they are powered by the Power Domain where the declaration/assignment is made.

All constants are corrupted like any other logic when either Power or Ground of the domain goes OFF.

Since constants are corrupted like any other logic, they would also need isolation. Isolation strategy can be specified on all constant driven paths which cross a domain boundary. Constants in the port map of cell mapped (macro) instances will not be corrupted unless there is isolation on that port.

Supply Based Corruption of Constants

This is the default behavior in the previous releases. 1'b0 is corrupted if VSS (ground) is OFF and 1'b1 gets corrupted if VDD (power) is OFF. The old behavior can be achieved by using the `-power=const_corruption_old` compile-time option.

Note:

Isolation will not be applied on constants.

Avoid Corruption for Constants in Continuous Assignments

The `-power=pass_thru_const` compile-time option skips the corruption of LHS of continuous assignments with constant literal in RHS, treating them as pass through.

```
assign net = 1'b0;  
assign net1 = 1'b1;
```

Note:

Any isolation policy defined will still be applicable on constants.

Skip Corruption of Constants

The `-power=apfcompat` compile-time option skips the corruption on constants in port map and continuous assignments with constants.

Note:

Any isolation policy defined will still be applicable on constants.

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.

MVSIM Native generates an error message when an isolation strategy with location “parent” is specified on 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'.
```

Error for Conflicting Port State Names in the `add_port_state` Command

The `add_port_state` command adds 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.

Starting from this release, 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 below 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 "Error- [UPF_DUPPS] Duplicate Port States Error" message.

`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}  
..
```

Appendix

This section consists of the following topics:

- [Low Power Assertions](#)
- [Using `bind_checker` Command](#)

Low Power Assertions

This topic 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 Assertions](#)
- [List of Low Power Assertions \(LP_MSG\)](#)
- [Predefined Data Types](#)
- [Supported UPF Query Commands](#)

List of Automated Assertions

[Table 1-19](#) lists all the automated assertions.

Table 1-19 List of Automated Assertions

ID	Severity	Message
LP_PD_INIT_STATE	INFO	Power domain '%s' started in %s state.
LP_PD_STATE_CHANGE	INFO	Power domain '%s' state changed from %s to %s.
LP_PPN_INIT_STATE	INFO	Primary power net '%s' of power domain '%s' started in %s state.
LP_PPN_INIT_VALUE	INFO	Primary power net '%s' of power domain '%s' started with voltage %f V.
LP_PGN_INIT_STATE	INFO	Primary ground net '%s' of power domain '%s' started in %s state.
LP_PGN_INIT_VALUE	INFO	Primary ground net '%s' of power domain '%s' started with voltage %f V.
LP_PPN_STATE_CHANGE	INFO	State of the primary power net '%s' of power domain '%s' changed from %s to %s.
LP_PGN_STATE_CHANGE	INFO	State of the primary ground net '%s' of power domain '%s' changed from %s to %s.
LP_PPN_VALUE_CHANGE	INFO	Voltage of the primary power net '%s' of power domain '%s' changed from %f V to %f V.
LP_PGN_VALUE_CHANGE	INFO	Voltage of the primary ground net '%s' of power domain '%s' changed from %f V to %f V.
LP_ISOEN_INIT_VALUE	INFO	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' started with value '%v'. Isolation sense specified is '%s'.
LP_ISOEN_CHANGE	INFO	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' changed from %v to %v. Isolation sense specified is '%s'.
LP_ISO_APPLIED	INFO	Isolation applied for strategy '%s' of power domain '%s'.
LP_ISO_REMOVED	INFO	Isolation removed for strategy '%s' of power domain '%s'.

ID	Severity	Message
LP_ISOPN_INIT	INFO	Isolation power net '%s' for isolation strategy '%s' of power domain '%s' started in state %s.
LP_ISOPN_STATE	INFO	State of isolation power net '%s' for isolation strategy '%s' of power domain '%s' changed from %s to %s.
LP_ISOGN_INIT	INFO	Isolation ground net '%s' for isolation strategy '%s' of power domain '%s' started in state %s.
LP_ISOGN_STATE	INFO	State of isolation ground net '%s' for isolation strategy '%s' of power domain '%s' changed from %s to %s.
LP_PST_INIT_STATE	INFO	Design started with '%s' state in pst '%s'.
LP_PST_STATE_CHANGE	INFO	Design state changed from '%s' to '%s' in pst '%s'.
LP_PSW_CTRL_VALUE	INFO	Signal '%s' connected to control port '%s' of power switch '%s' started with a value '%v'.
LP_PSW_ISP_INIT_STATE	INFO	Supply net '%s' tied to input supply port '%s' of power switch '%s' started with state %s.
LP_PSW_ISP_INIT_VALUE	INFO	Supply net '%s' tied to input supply port '%s' of power switch '%s' started with voltage %f V.
LP_PSW_OSP_INIT_STATE	INFO	Supply net '%s' tied to output supply port '%s' of power switch '%s' started with state %s.
LP_PSW_OSP_INIT_VALUE	INFO	Supply net '%s' tied to output supply port '%s' of power switch '%s' started with voltage %f V.
LP_PSW_INIT_STATE	INFO	Power switch '%s' started in %s (%s) state.
LP_PSW_STATE_CHANGE	INFO	State of power switch '%s' changed from %s (%s) to %s (%s).
LP_PSW_CTRL_CHANGE	INFO	Signal '%s' connected to control port '%s' of power switch '%s' changed from '%v' to '%v'.

ID	Severity	Message
LP_PSW_OSP_STATE	INFO	State of supply net '%s' tied to output supply port '%s' of power switch '%s' changed from %s to %s.
LP_PSW_OSP_VALUE	INFO	Voltage of supply net '%s' tied to output supply port '%s' of power switch '%s' changed from %f V to %f V.
LP_PSW_ISP_STATE	INFO	State of supply net '%s' tied to input supply port '%s' of power switch '%s' changed from %s to %s.
LP_PSW_ISP_VALUE	INFO	Voltage of supply net '%s' tied to input supply port '%s' of power switch '%s' changed from %f V to %f V.
LP_RETPN_INIT	INFO	Retention power net '%s' for retention strategy '%s' of power domain '%s' started with %s state.
LP_RETPN_STATE	INFO	State of retention power net '%s' for retention strategy '%s' of power domain '%s' changed from %s to %s.
LP_RETGN_INIT	INFO	Retention ground net '%s' for retention strategy '%s' of power domain '%s' started with %s state.
LP_RETGN_STATE	INFO	State of retention ground net '%s' for retention strategy '%s' of power domain '%s' changed from %s to %s.
LP_SAVE_INIT	INFO	Save signal '%s' for retention strategy '%s' of power domain '%s' started with a value '%v'. Save sense specified is '%s'.
LP_SAVE_CHANGE	INFO	Save signal '%s' for retention strategy '%s' of power domain '%s' changed from '%v' to '%v'. Save sense specified is '%s'.
LP_SAVE_ENABLED	INFO	Save operation enabled for retention strategy '%s' of power domain '%s'. Save signal '%s' is at an active value '%v'.
LP_SAVE_DISABLED	INFO	Save operation disabled for retention strategy '%s' of power domain '%s'. Save signal '%s' changed to an inactive value '%v'.

ID	Severity	Message
LP_RESTORE_INIT	INFO	Restore signal '%s' for retention strategy '%s' of power domain '%s' started with a value '%v'. Restore sense specified is '%s'.
LP_RESTORE_CHANGE	INFO	Restore signal '%s' for retention strategy '%s' of power domain '%s' changed from '%v' to '%v'. Restore sense specified is '%s'.
LP_RESTORE_ENABLED	INFO	Restore operation enabled for retention strategy '%s' of power domain '%s'. Restore signal '%s' is at an active value '%v'.
LP_RESTORE_DISABLED	INFO	Restore operation disabled for retention strategy '%s' of power domain '%s'. Restore signal '%s' changed to an inactive value '%v'.
LP_PPN_STATE_GLITCH	WARNING	Multiple changes at same time on State of the primary power net '%s' of power domain '%s'. State changed from %s -> %s -> %s.
LP_PGN_STATE_GLITCH	WARNING	Multiple changes at same time on State of the primary ground net '%s' of power domain '%s'. State changed from %s -> %s -> %s.
LP_PPN_VALUE_GLITCH	WARNING	Multiple voltage changes at same time on the primary power net '%s' of power domain '%s'. Voltage changed from %f V -> %f V -> %f V.
LP_PGN_VALUE_GLITCH	WARNING	Multiple voltage changes at same time on the primary ground net '%s' of power domain '%s'. Voltage changed from %f V -> %f V -> %f V.
LP_ISO_INIT_OFF	WARNING	Isolation strategy '%s' of power domain '%s' started in CORRUPT state.
LP_ISOEN_INIT_INVALID	WARNING	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' started with an invalid value '%v'.

ID	Severity	Message
LP_ISOEN_GLITCH	WARNING	Glitch on isolation enable '%s' of isolation strategy '%s' of power domain '%s'. Isolation enable changed from %v -> %v -> %v.
LP_ISOPN_OFF	WARNING	State of isolation power net '%s' for isolation strategy '%s' of power domain '%s' changed to %s.
LP_ISOGN_OFF	WARNING	State of isolation ground net '%s' for isolation strategy '%s' of power domain '%s' changed to %s.
LP_ISO_SEQFAIL1	WARNING	Isolation enable '%s' for isolation strategy '%s' is not active when the source power domain '%s' changed to %s state.
LP_ISO_SEQFAIL2	WARNING	Isolation removed before the source power domain '%s' turned ON. Isolation strategy : '%s'.
LP_PST_INIT_ILLEGAL	WARNING	Design started with '%s' state in pst '%s'.
LP_PST_STATE_ILLEGAL	WARNING	Design state changed from '%s' to '%s' in pst '%s'.
LP_SAVE_RETPWR_OFF	WARNING	Save operation disabled. Retention power for retention strategy '%s' of power domain '%s' turned OFF while save signal '%s' is at an active level '%v'.
LP_RET_PWR_OFF	WARNING	Retention power for retention strategy '%s' of power domain '%s' turned OFF. Shadow registers are corrupted.
LP_RESTORE_PRIMPWR_OFF	WARNING	Restore operation disabled. Primary power of power domain '%s' turned OFF while restore signal '%s' of retention strategy '%s' is at an active level '%v'.
LP_ISOEN_INVALID	ERROR	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' changed to an invalid value '%v'.
LP_PSW_CTRL_INVALID	ERROR	Signal '%s' connected to control port '%s' of power switch '%s' started with an invalid value '%v'.

ID	Severity	Message
LP_PSW_CTRL_ERROR	ERROR	Signal '%s' connected to control port '%s' of power switch '%s' changed from '%v' to an invalid value '%v'.
LP_RESTORE_X	ERROR	Restoring X on the registers of power domain '%s'. Retention power for retention strategy '%s' is in OFF state when restore is applied.
LP_RESTORE_NOSAVE	ERROR	Restoring X on the registers of power domain '%s'. Restore operation is enabled without a save event, after retention power is turned ON. Retention Strategy : '%s', Restore Signal : '%s', Save Signal : '%s'.
LP_ASSERT_R_MUTEX	ERROR	'%s' defined for retention strategy '%s' of power domain '%s' failed. Signal '%s' ('%s') is not mutually exclusive with the restore signal '%s' ('%s').
LP_ASSERT_S_MUTEX	ERROR	'%s' defined for retention strategy '%s' of power domain '%s' failed. Signal '%s' ('%s') is not mutually exclusive with the save signal '%s' ('%s').
LP_ASSERT_RS_MUTEX	ERROR	'%s' defined for retention strategy '%s' of power domain '%s' failed. %s '%s' ('%s') and %s '%s' ('%s') are not mutually exclusive.

List of Low Power Assertions (LP_MSG)

Table 1-20 lists all the automated low power assertions.

Table 1-20 Low Power Assertions (LP_MSG)

Context	ID	Severity	Message
HDL2UPF Connections	LP_SNET_HDL_SNET_UPDATE	INFO	HDL Supply Net '%s' updated to value '%s'.
	LP_SNET_HDL_SNET_CONN_UPDATE	INFO	HDL Supply Port '%s' driving UPF Supply Net '%s' updated to value '%s'.
Clock/Reset Wiggle	LP_CLOCK_WIGGLE, LP_RESET_WIGGLE	WARNING	Clock '%s' wiggling in instance '%s' during shutdown for the power domain '%s' at time %0t. Reset '%s' wiggling in instance '%s' during shutdown for the power domain '%s' at time %0t.
	LP_COA_CLOCK_WIGGLE, LP_COA_RESET_WIGGLE	WARNING	Clock '%s' wiggling in instance '%s' during standby for the power domain '%s' at time %0t. Reset '%s' wiggling in instance '%s' during standby for the power domain '%s' at time %0t.
Isolation	LP_ISO_EN	INFO	Isolation enabled for isolation strategy '%s' of power domain '%s'.
	LP_ISO_EN_PWRDN	INFO	Isolation enabled for isolation strategy '%s' when Power Domain '%s' is OFF.
	LP_ISO_DIS	INFO	Isolation disabled for isolation strategy '%s' of power domain '%s'.

Context	ID	Severity	Message
	LP_ISO_DIS_PWRDN	INFO	Isolation disabled for isolation strategy '%s' when Power Domain '%s' is OFF.
	LP_ISOEN_INIT_VALUE	INFO	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' started with value '%s'. Isolation sense specified is '%s'.
	LP_ISO_INIT_OFF	WARNING	Isolation strategy '%s' of power domain '%s' started in CORRUPT state.
	LP_ISOEN_INIT_INVALID	WARNING	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' started with an invalid value '%s'.
	LP_ISOEN_CHANGE	INFO	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' changed from '%s' to '%s'. Isolation sense specified is '%s'.
	LP_ISOEN_INVALID	ERROR	Isolation enable '%s' of isolation strategy '%s' of power domain '%s' changed to an invalid value '%s'.
	LP_ISOPN_INIT	INFO	Isolation power net '%s' for isolation strategy '%s' of power domain '%s' started in state %s.
	LP_ISOPN_STATE	INFO	State of isolation power net '%s' for isolation strategy '%s' of power domain '%s' changed from %s to %s.
	LP_ISOGN_INIT	INFO	Isolation ground net '%s' for isolation strategy '%s' of power domain '%s' started in state %s.

Context	ID	Severity	Message
	LP_ISOGN_STATE	INFO	State of isolation ground net '%s' for isolation strategy '%s' of power domain '%s' changed from %s to %s.
	LP_ISOPN_OFF	WARNING	State of isolation power net '%s' for isolation strategy '%s' of power domain '%s' changed to %s.
	LP_ISOGN_OFF	WARNING	State of isolation ground net '%s' for isolation strategy '%s' of power domain '%s' changed to %s.
	LP_ISO_SEQFAIL1	WARNING	Isolation enable '%s' for isolation strategy '%s' is not active when the source power domain '%s' changed to %s 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: %s, File: %s, Line: %s."
	LP_COA_CORRUPT_CA	WARNING	"LHS of cont-assign corrupted due to activity on RHS in simstate 'CORRUPT_ON_ACTIVITY' . Instance: %s, File: %s, Line: %s."
	LP_COA_CORRUPT_GATE	WARNING	"Output of primitive gate corrupted due to activity on inputs in simstate 'CORRUPT_ON_ACTIVITY' . Instance: '%s', File: '%s', Line: '%s'."

Context	ID	Severity	Message
	LP_COA_CORRUPT_UDP	WARNING	"Output of UDP corrupted due to activity on inputs in simstate 'CORRUPT_ON_ACTIVITY' . Instance: %s, File: %s, Line: %s."
	LP_COA_CORRUPT_SIGNAL	WARNING	Signal %s corrupted due to activity.
	LP_COA_CORRUPT_VARIABLE	WARNING	Variable %s corrupted due to activity.
Message Control	LP_CFG_FILE_ERR	ERROR	"Cannot open specified config file: %s, please check for the file existence and file permissions and rerun the simulation."
	LP_MSG_SUMMARY	INFO	
	LP_MSG_ONOFF	INFO	Message '%s' is now '%s'.
	LP_MSG_NF	WARNING	" '%s', '%s': Message '%s' referenced in function '%s' not found."
	LP_MSG_INVALID_ARGUMENT	WARNING	" '%s', '%s': Argument '%s' in function '%s' is invalid. Valid values are '%s'."
	LP_MSG_SEV	INFO	Severity of message '%s' is now '%s'.
	LP_MSG_REG	INFO	Message '%s' is registered successfully.
	LP_MSG_INT	WARNING	" '%s', '%s': '%s' is an internal message. 'lp_msg_print' can be used only with messages registered through 'lp_msg_register'."
	LP_MSG_AE	WARNING	" '%s', '%s': Message '%s' in function '%s' already Exists. Already registered message cannot be registered again."

Context	ID	Severity	Message
Power Domain	LP_PD_INIT_STATE	INFO	Power domain '%s' started in '%s' state.
	LP_PD_STATE_CHANGE	INFO	Power domain '%s' state changed from '%s' to '%s'.
	LP_PD_TRANS	INFO	Power Domain '%s' transitioned to state '%s'
	LP_PD_PWRUP	INFO	Power Domain '%s' powered up.
	LP_PD_PWRDN	INFO	Power Domain '%s' powered down.
	LP_MACRO_PD_PWRUP	INFO	Power Domain '%s' powered up.
	LP_MACRO_PD_PWRDN	INFO	Power Domain '%s' powered down.
	LP_PPN_INIT_STATE	INFO	Primary power net '%s' of power domain '%s' started in %s state.
	LP_PPN_INIT_VALUE	INFO	Primary power net '%s' of power domain '%s' started with voltage %s V.
	LP_PGN_INIT_STATE	INFO	Primary ground net '%s' of power domain '%s' started in %s state.
	LP_PGN_INIT_VALUE	INFO	Primary ground net '%s' of power domain '%s' started with voltage %s V.
	LP_PPN_STATE_CHANGE	INFO	State of the primary power net '%s' of power domain '%s' changed from %s to %s.
	LP_PGN_STATE_CHANGE	INFO	State of the primary ground net '%s' of power domain '%s' changed from %s to %s.

Context	ID	Severity	Message
	LP_PPN_VALUE_CHANGE	INFO	Voltage of the primary power net '%s' of power domain '%s' changed from %s V to %s V.
	LP_PGN_VALUE_CHANGE	INFO	Voltage of the primary ground net '%s' of power domain '%s' changed from %s V to %s V.
Power Switch	LP_PSW_CTRL_INIT_INVALID	ERROR	Signal '%s' connected to control port '%s' of power switch '%s' started with an invalid value '%s'.
	LP_PSW_CTRL_INIT_VALUE	INFO	Signal '%s' connected to control port '%s' of power switch '%s' started with a value '%s'.
	LP_PSW_ISP_INIT_STATE	INFO	Supply net '%s' tied to input supply port '%s' of power switch '%s' started with state %s.
	LP_PSW_ISP_INIT_VALUE	INFO	Supply net '%s' tied to input supply port '%s' of power switch '%s' started with voltage %s V.
	LP_PSW_OSP_INIT_STATE	INFO	Supply net '%s' tied to output supply port '%s' of power switch '%s' started with state %s.
	LP_PSW_OSP_INIT_VALUE	INFO	Supply net '%s' tied to output supply port '%s' of power switch '%s' started with voltage %s V.
	LP_PSW_INIT_STATE	INFO	Power switch '%s' started in %s (%s) state.
	LP_PSW_STATE_CHANGE	INFO	State of power switch '%s' changed from %s (%s) to %s (%s).

Context	ID	Severity	Message
	LP_PSW_CTRL_CHANGE	INFO	Signal '%s' connected to control port '%s' of power switch '%s' changed from '%s' to '%s'.
	LP_PSW_CTRL_INVALID	ERROR	Signal '%s' connected to control port '%s' of power switch '%s' changed from '%s' to an invalid value '%s'.
	LP_PSW_OSP_STATE	INFO	State of supply net '%s' tied to output supply port '%s' of power switch '%s' changed from %s to %s.
	LP_PSW_OSP_VALUE	INFO	Voltage of supply net '%s' tied to output supply port '%s' of power switch '%s' changed from %s V to %s V.
	LP_PSW_ISP_STATE	INFO	State of supply net '%s' tied to input supply port '%s' of power switch '%s' changed from %s to %s.
	LP_PSW_ISP_VALUE	INFO	Voltage of supply net '%s' tied to input supply port '%s' of power switch '%s' changed from %s V to %s V.
	LP_PSW_SUPPLY_CHANGE	INFO	Output Supply Port of Power Switch '%s' changed to state = '%s' and voltage = %s volt(s) with enable expression '%s'.
	LP_PSW_INIT	INFO	Output Supply Port of Power Switch '%s' initialized to state = '%s' and enable expression '%s'.
PST	LP_PST_INIT_STATE	INFO	Design started with '%s' state in pst '%s'.
	LP_PST_INIT_ILLEGAL	WARNING	Design started with '%s' state (illegal) in pst '%s'.

Context	ID	Severity	Message
	LP_PST_STATE_CHANGE	INFO	Design state changed from '%s' to '%s' in pst '%s'.
	LP_PST_STATE_ILLEGAL	WARNING	Design state changed from '%s' to '%s' (illegal) in pst '%s'.
Random Corruption	LP_DOMAIN_RANDOM_SEED	INFO	Random corruption seed specified for domain '%s' is '%s'.
	LP_DESIGN_RANDOM_SEED	INFO	Random corruption seed is not specified for domain '%s'. Using random corruption seed '%s' specified for design.
	LP_AUTO_RANDOM_SEED	INFO	Random corruption seed is not specified for domain '%s'. Using automatic generated random corruption seed '%s'.
Retention	LP_RETPN_INIT	INFO	Retention power net '%s' for retention strategy '%s' of power domain '%s' started with %s state.
	LP_RETGN_INIT	INFO	Retention ground net '%s' for retention strategy '%s' of power domain '%s' started with %s state.
	LP_RETPN_STATE	INFO	State of retention power net '%s' for retention strategy '%s' of power domain '%s' changed from %s to %s.
	LP_RETGN_STATE	INFO	State of retention ground net '%s' for retention strategy '%s' of power domain '%s' changed from %s to %s.

Context	ID	Severity	Message
	LP_RET_PWR_OFF	WARNING	Retention power for retention strategy '%s' of power domain '%s' turned OFF. Shadow registers are corrupted.
	LP_SAVE_ON	INFO	Save asserted for retention strategy '%s' of power domain '%s'.
	LP_SAVE_OFF	INFO	Save deasserted for retention strategy '%s' of power domain '%s'.
	LP_SAVE_RETPWRDN	ERROR	Save event triggered for retention strategy '%s' of power domain '%s' when retention power is OFF.
	LP_RESTORE_ON	INFO	Restore asserted for retention strategy '%s' of power domain '%s'.
	LP_RESTORE_OFF	INFO	Restore deasserted for retention strategy '%s' of power domain '%s'.
	LP_RESTORE_RETPWRDN	ERROR	Restore event triggered for retention strategy '%s' of power domain '%s' when retention power is OFF.
	LP_SAVE_INIT	INFO	Save signal '%s' for retention strategy '%s' of power domain '%s' started with a value '%s'. Save sense specified is '%s'.
	LP_SAVE_INIT_INVALID	WARNING	Save signal '%s' for retention strategy '%s' of power domain '%s' started with an invalid value '%s'.

Context	ID	Severity	Message
	LP_SAVE_CHANGE	INFO	Save signal '%s' for retention strategy '%s' of power domain '%s' changed from '%s' to '%s'. Save sense specified is '%s'.
	LP_SAVE_INVALID	WARNING	Save signal '%s' for retention strategy '%s' of power domain '%s' changed from '%s' to '%s'.
	LP_SAVE_PRIMPWR_OFF	WARNING	Save operation enabled for retention strategy '%s' of power domain '%s' while the primary power of domain is OFF. Save signal '%s' is at an active level '%s'.
	LP_RESTORE_INIT	INFO	Restore signal '%s' for retention strategy '%s' of power domain '%s' started with a value '%s'. Restore sense specified is '%s'.
	LP_RESTORE_INIT_INVALID	WARNING	Restore signal '%s' for retention strategy '%s' of power domain '%s' started with an invalid value '%s'.
	LP_RESTORE_CHANGE	INFO	Restore signal '%s' for retention strategy '%s' of power domain '%s' changed from '%s' to '%s'. Restore sense specified is '%s'.
	LP_RESTORE_INVALID	WARNING	Restore signal '%s' for retention strategy '%s' of power domain '%s' changed from '%s' to '%s'.

Context	ID	Severity	Message
	LP_RESTORE_PRIMP WR_OFF	WARNING	Restore operation disabled. Primary power of power domain '%s' turned OFF while restore signal '%s' of retention strategy '%s' is at an active level '%s'.
	LP_RESTORE_NOSAVE	ERROR	" Restoring X on the registers of power domain '%s'. Restore operation is enabled without a save event, after retention power is turned ON. Retention Strategy: '%s', Restore Signal: '%s', Save Signal: '%s'."
Retention assert_mutex	LP_ASSERT_R_MUTEX	ERROR	'assert_r_mutex' defined for retention strategy '%s' of power domain '%s' failed. Restore signal '%s' ('%s') and signal '%s' ('%s') are not mutually exclusive.
	LP_ASSERT_S_MUTEX	ERROR	'assert_s_mutex' defined for retention strategy '%s' of power domain '%s' failed. Save signal '%s' ('%s') and signal '%s' ('%s') are not mutually exclusive.
	LP_ASSERT_RS_MUTEX	ERROR	'assert_rs_mutex' defined for retention strategy '%s' of power domain '%s' failed. Restore signal '%s' ('%s') and save signal '%s' ('%s') are not mutually exclusive.
Supply Sets	LP_SS_INIT	INFO	Supply Set '%s' initialized to state '%s' with simstate '%s'.
	LP_SS_ILLEGAL	WARNING	Supply Set '%s' initialized to illegal state '%s' with simstate '%s'.

Context	ID	Severity	Message
	LP_SS_STATE_CHANGE	INFO	Supply Set '%s' transitioned to state '%s' with simstate '%s'.
	LP_SS_STATE_CHANGE_ILLEGAL	WARNING	Supply Set '%s' transitioned to illegal state '%s' with simstate '%s'.
	LP_SS_EXPR_MISMATCH	ERROR	Supply expression's value '%s' does not match with Logic expression value '%s' for state '%s' of Supply Set '%s'.
	LP_SHUTOFF_EXPR_MISMATCH	ERROR	External Shutoff Expression of Power Domain '%s' is %s but existing %s of the domain is %s.
UPF::supply_on/supply_off	LP_SUPPLY_PAD_NOT_FOUND	WARNING	Supply Pad '%s' not found in 'UPF::%s' function.
	LP_SUPPLY_PAD_CALL	INFO	Supply Pad function 'UPF::%s' called for Supply Pad '%s'. Supply Pad value changed to '%s'.
	LP_SUPPLY_PAD_VOLTAGE	WARNING	Supply Pad function 'UPF::%s' called for Power Supply Pad '%s' with '0' voltage. Supply Pad value changed to '%s'.

Predefined Data Types

You can use the following predefined data types of UPF and SNPS_LPA_PKG packages to write custom low power assertions. You must import the UPF and SNPS_LPA_PKG package before using these data types.

Type	Definition	Description
lpa_severity_T (SNPS_LPA_PKG)	<pre>typedef enum { INFO, WARNING, ERROR, FATAL } lpa_severity_T;</pre>	Four levels of severity are defined for low power assertions. Any low power assertion with severity FATAL will terminate the simulation.
lpa_msg_table_T (SNPS_LPA_PKG)	<pre>typedef struct { string id; lpa_severity_T severity; string format; bit disabled; } lpa_msg_table_T[];</pre>	<p>Low power assertions are captured in a consistent format as part of the message table. Message table comprises of four different fields.</p> <p>id: string type to capture the assertion ID.</p> <p>severity: lpa_severity_T type as defined above.</p> <p>format: string type to capture the complete message that needs to be printed. It should include the format specifiers like %d, %s as needed.</p> <p>disabled: bit type that indicates whether an assertion is disabled/enabled. Set it to 1 to disable an assertion.</p>

Type	Definition	Description
lpa_log_mode_T (SNPS_LPA_PKG)	<pre>typedef enum { DEFAULT, CUSTOM, DEFAULT_CUSTOM } lpa_log_mode_T;</pre>	<p>Three modes are provided for logging low power assertions.</p> <p>DEFAULT: Log in the simulation log file specified by '-l' VCS option and STDOUT.</p> <p>CUSTOM: Log only in the log file specified by lpa_logfile ini variable.</p> <p>DEFAULT_CUSTOM: Log in both the simulation log file and the file specified by 'lpa_logfile' config setting including STDOUT.</p>
supply_net_type (UPF)	<pre>typedef struct packed { state int voltage; } supply_net_type;</pre>	<p>While binding UPF supply net/port to a checker port, the corresponding port in the checker module must be declared as supply_net_type.</p> <p>supply_net_type.state returns the current state of the supply net. The states can be OFF, UNDETERMINED, PARTIAL_ON, and FULL_ON.</p> <p>supply_net_type.voltage returns the voltage representation in micro volts.</p>

Supported UPF Query Commands

You can use the following UPF query commands to write custom low power assertions.

Note:

The names of 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.	UPF::supply_net_type/ 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> command</code> . 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[ind ex]. Index 0 represents undetermined state. Array of all state_names must be generated from 'state_name' value of 'states' key.	SV int

* state_type will be 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 other fanout fanin faninout parent sibling)	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	<ol style="list-style-type: none"> 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} <p>Sense is either high, low, posedge, or negedge.</p>	<ol style="list-style-type: none"> 1. NA 2. "" 3. {SV wire/SV string SV string}
assert_s_mutex	<p>{net_name sense}</p> <p>Sense is either high, low, posedge, or negedge.</p>	{SV wire/SV string SV string}
domain	Full hierarchical name of the power domain.	SV string
restore_signal	<p>{restore_signal sense}</p> <p>restore_signal is returned in the name alias format, use 'query_original_name' tcl proc to get its simple name.</p> <p>Sense is either high, low, posedge, or negedge.</p>	{SV wire/string SV string}
save_signal	<p>{save_signal sense}</p> <p>save_signal is returned in the name alias format, use 'query_original_name' tcl proc to get its simple name.</p> <p>Sense is either high, low, posedge, or negedge.</p>	{SV wire/string SV string}

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*]
[-bind_to module]
[-ports {{port_name net_name/upf_name*}}]
[-parameters {{parameter_name constant_expression/
upf_name*}}]
[-name_prefix prefix]
[-name_suffix suffix]
```

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 instance.

-elements *element_list*

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

-bind_to *module*

The SystemVerilog module for which all instances are the target of this command. If **-bind_to** is specified, an instance of checker is created in every instance of the module.

-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 }}*

This allows parameter passing to the checker module.

-name_prefix *prefix*

In case the checker is bound to policy, 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 policy, 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 UPF `bind_checker` command, UPF names must be prefixed with '@' for distinction.

For example, consider the below 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 binding specified with `-elements` must be any UPF object like Power Domain, Power Switch, Isolation Strategy, 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.