



Power Aware Simulation User's Manual

ModelSim® SE

Software Version 10.2c

© 2010-2013 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

Chapter 1	
Getting Started With Power Aware Simulation	13
Where Is Power Aware in Your Design Flow?	14
Documentation—Scope and Organization	16
Contents of This Manual	17
How to Use This Manual	17
Related Documentation	18
Chapter 2	
Concepts for Using Power Aware Simulation	19
Power Intent Specification	19
Power Aware Modeling	19
Modeling Corruption	20
Corruption Values	21
Default Corruption Semantics	21
Modeling Isolation	22
Modeling Retention	22
Edge-sensitive and Level-sensitive Control of Retention Models	23
Modeling Bias	24
Chapter 3	
Power Aware Simulation	27
Inputs Required for Power Aware Simulation	27
Commands Used For Power Aware Simulation	27
General Steps for Running Power Aware	28
Power Aware Simulation Flows	28
Using the Standard Flow	29
Compile	29
Optimize	30
Simulate	31
Using the Delayed Optimization Flow	32
Using the No-Optimization Flow	33
Working with Liberty Libraries	35
Liberty Library Models	35
Using Liberty Files with vopt	35
Creating and Saving a Liberty Database	36
Using a Previously Created Liberty Database	36
Using Liberty Files Directly	36
Updating a Liberty Database	37
Refreshing a Liberty Database	37
PDU-Based Simulation	37
Power Aware Simulation Debug	38

Capturing Information for Post-simulation Debug	38
Debugging Designs Containing Liberty Cells	39
Applying Power Intent to the Design	39
Detection of Gate-level Cells	40
Detection of Power Management Cells	40
Automatic Insertion of Power Management Cells	42
Automatic Corruption and Retention of UDPs	42
UDP Corruption and Retention Modes	42
Chapter 4	
Power Aware Reports	45
Generating Reports for Power Aware	45
How to Generate a Report with vopt -pa_genrpt	45
UPF Reports	46
UPF Power Intent Report	47
Example of UPF File and Power Intent Report	49
UPF Static Report	51
Example of UPF Static Report File	51
Static Checking UPF Reports	52
Dynamic UPF Report	53
Architecture Report	54
Power Domain	54
Power Switch	55
Retention Strategy	57
Isolation Strategy	57
Level Shifter Strategy	58
Power State Tables (PSTs)	59
Sample Power Architecture Report	60
Design Elements Report	61
Design Element Scopes and Power Domains	62
Corrupted Signals	63
State Elements	63
Retention Signals	64
Working With A Design Element Report	65
Behavioral Element Reporting	67
Chapter 5	
Automatic Checking	69
Static and Dynamic Checking Overview	69
Level Shifter Checking	69
Isolation Checking	70
Static Checking in Power Aware	70
Usage Notes for Static Checking	71
Debugging Static Checks	71
Static Isolation Checks	72
Static Level Shifter Checks	74
Reporting for a Valid Level Shifter	77
Dynamic Checking in Power Aware	78

Table of Contents

Usage Notes	78
Dynamic Retention Checking	79
Dynamic Isolation Checking	81
Dynamic Level Shifter Checking	84
Operating Voltage for Dynamic Checking	84
Miscellaneous Dynamic Checking	86
Implementing Checking at Gate Level	88
Level Shifting for Gate-Level Checking	88
Isolation for Gate-Level Checking	88
Quick-Reference Comparison of Static and Dynamic Check Arguments	89
Chapter 6	
Power Aware Coverage	93
Power Aware Coverage Collection	93
Collecting Power Aware Coverage Information on Dynamic Checks	93
Collecting Power Aware Coverage Information on States and Transitions	95
Power Aware Coverage Analysis	96
Generating a UCDB File for Power Aware	96
Generating Power Aware Coverage Reports	96
Generating HTML Reports	99
Accessing Coverage Data for Post-Simulation Analysis	99
Power Aware Coverage Report Reference	100
Power Aware Coverage Summary	100
Power Aware Coverage Detailed Report	100
Power Aware Coverage Verbose Report	103
Power Aware Checks Coverage Summary	103
Power Aware Checks Coverage Detailed Report	103
Chapter 7	
Visualization of Power Aware Operations	105
UPF Object Display	105
Visualizing UPF Objects in the GUI	105
Power Aware Schematic Display	107
Top-Down Debugging (From the Test Bench)	108
Bottom-Up Debugging (From the Design Under Test)	108
Usage Notes	109
Schematic Window Visualization for Debugging	109
Power Aware Waveform Display	113
Using Power Aware Highlighting	115
Power State and Transition Display	115
Power State and Transition Concepts	115
Differences Between a Conventional RTL FSM and a PASM	116
Visualizing Power Aware State Machines	116
Power Aware State Machine List Window	117
Power Aware State Machine Viewer Window	119

Appendix A

Power Aware Commands and Options	125
ModelSim Commands Used for Power Aware	126
Using -pa_enable and -pa_disable	128
Additional Commands You Can Use with Power Aware	131
Power Aware Messages	136
Dynamic Power Aware Check Message Control	136
Concepts of pa msg	136
Use Model for pa msg	137
Option Descriptions of pa msg	138
Examples	140
Controlling Power Aware Message Severity During vopt Stage	141
Excluding Design Elements from Power Aware	142
Voltage Level-Shifting (Multi-Voltage Analysis)	145
Power State Tables	145
Example	145
Level Shifter Specification	146
Reporting	146
Threshold Control for Level Shifters	146
Level Shifter Instances	146
Limitations on Level Shifting	147
Restricting Isolation and Level Shifting on a Port	147
Isolation and Level Shifting Behavior	147
How to Apply the -source and -sink Arguments	148
Simulating Designs Containing Macromodels	152
Using UPF Commands	152
Attributes in RTL	153
Liberty File	154
Example of Power Intent on a Hard Macro	154
UPF Commands	155
RTL Attributes	155
Liberty File Attributes	156
Creating Feedthrough For RTL Conversion Functions	156

Appendix B

Model Construction for Power Aware Simulation	159
Assumptions and Advantages	159
Basic Model Structure	159
Named Events in Power Aware	160
Usage Note for Sequence Requirements	162
Attributes	162
Retention Cells and Memories	162
Isolation Cells	163
Level Shifters	163
Model Interface Ports	164
Customizing Activity at Time Zero	164
Example—Register Model	165
Example—Corrupt Model	168

Appendix C

UPF Commands and Reference..... 169

- Unified Power Format (UPF)..... 169
 - Using a UPF File as Part of Power Aware Simulation..... 170
- UPF Standards..... 170
 - Version 1.0 of the UPF Standard..... 171
 - Version 2.0 of the UPF Standard: IEEE Std 1801-2009..... 171
 - Syntax and Semantic Differences Between UPF 1.0 and UPF 2.0..... 171
 - UPF 2.0 Syntax Differences..... 172
 - UPF 2.0 Semantic Differences..... 172
- Supported UPF Commands..... 172
 - add_domain_elements..... 177
 - add_port_state..... 178
 - add_power_state..... 179
 - add_pst_state..... 183
 - associate_supply_set..... 184
 - bind_checker..... 185
 - connect_logic_net..... 188
 - connect_supply_net..... 189
 - connect_supply_set..... 190
 - create_composite_domain..... 191
 - create_hdl2upf_vct..... 197
 - create_logic_net..... 198
 - create_logic_port..... 199
 - create_power_domain..... 200
 - create_power_switch..... 202
 - create_pst..... 203
 - create_supply_net..... 204
 - create_supply_port..... 205
 - create_supply_set..... 206
 - create_upf2hdl_vct..... 207
 - find_objects..... 208
 - load_simstate_behavior..... 210
 - load_upf..... 213
 - load_upf_protected..... 214
 - map_isolation_cell..... 215
 - map_level_shifter_cell..... 218
 - map_retention_cell..... 219
 - name_format..... 220
 - query_design_attributes..... 221
 - query_port_state..... 222
 - query_power_domain..... 223
 - query_power_state..... 224
 - query_power_switch..... 225
 - query_pst..... 226
 - query_pst_state..... 227
 - query_supply_net..... 228
 - query_supply_port..... 229

save_upf	230
set_design_attributes	232
set_design_top	233
set_domain_supply_net	234
set_isolation	235
set_isolation_control	239
set_level_shifter	240
set_partial_on_translation	243
set_pin_related_supply	246
set_port_attributes	247
set_power_switch	249
set_retention	250
set_retention_control	252
set_scope	253
set_simstate_behavior	254
upf_version	259
Supported UPF Package Functions	259
Detailed Support for supply_net_type	260
Accessing Generate Blocks in UPF	261
Limitation	261
Supported UPF Attributes	261
Specifying Attributes	262
Limitations	263
Attributes in VHDL or SystemVerilog	263
Specifying Supply Nets in UPF	263
Format of Assigned Net Values	264
Changing the Default Supply State Values for VHDL Models	264
Supported UPF Extensions	265
Using -pa_upfextensions	265
UPF Supply Connections	269
Implicit Connections	269
Explicit Connections	270
Explicit Connections to HDL Ports	270
Examples	270
Explicit Connections to 1-bit HDL Ports	271
Explicit Connections to Supply Ports of Power Switch	271
Automatic Connections	272
Automatic Connections for Supply Nets	272
Automatic Connections for Supply Sets	273
Power State Composition	275
Determining State Dependency with add_power_state Arguments	276
Power State Reporting	278
Value Conversion Tables	280
Using VCT Commands	280
Examples	280
Limitations	281
Predefined VCTs Supported from the UPF Standard	281
Connections Using Value Conversion Tables (VCTs)	284
Simulation Semantics for UPF Supply Connections	286

Table of Contents

Supply Nets	287
Resolving Drivers on a Supply Net	287
Example	288
Defining Isolation	289
Method 1: Isolation is already explicitly present	289
Method 2: Isolation needs to be added	289
Specifying Isolation Cells	290
Limitations	291
Defining Retention	291
-retention_supply_set	292
-no_retention	293
-use_retention_as_primary	295
Appendix D	
Power Configuration File Reference.....	299
Power Specification File	299
Formats	299
Using a PCF as Part of Power Aware Verification	299
PCF Syntax and Contents	300
Basic PCF Statement Types	300
Statement Termination	301
Header Statement	301
Context Statements	302
Scope Statement	303
Variable Statement	303
Include Statement	304
Corruption Extent Statement	304
Power Statements	305
Region Definitions	306
Power Model Mapping Statement	310
Mapping Statement Precedence	312
Specifying Default Model Mappings	313
Retention Statement	314
Corruption Semantics	314
Voltage Domains	315
Comments	316
Regular Expressions and Variables	316
Rule Precedence	316
Appendix E	
Supplemental Information.....	319
Power Aware Verification of ARM-Based Designs	319
Abstract	319
Introduction	320
Active Power Management	320
Power Management Techniques	320
Power Management Specification	321
Power Management Architecture	322

Operating Modes	322
Power Domains	323
Power Distribution	324
Power States	326
Isolation and Level Shifting	326
State Retention	328
Power Managed Behavior	328
Power Control Logic	329
Power Aware Verification Flow	329
Verifying the Power Management Architecture	330
Verifying Power Managed Behavior	331
Verifying Power Control Logic	332
Summary	332
Acknowledgements	333
References	333

List of Tables

Table 4-1. Generating UPF Reports for Power Aware	46
Table 5-1. Static Isolation Checks	72
Table 5-2. Static Level Shifter Checks	74
Table 5-3. Dynamic Retention Checks	79
Table 5-4. Dynamic Isolation Checks	81
Table 5-5. Dynamic Level Shifter Checks	85
Table 5-6. Miscellaneous Dynamic Checks	86
Table 5-7. Static and Dynamic Checks Comparison	89
Table 7-1. Power Aware State Machine List Window Columns	118
Table 7-2. Power Aware State Machine List Window Popup Menu	119
Table 7-3. PA State Machines Menu	119
Table 7-4. Power Aware State Machine Viewer Window Popup Menu	122
Table 7-5. FSM View Menu, Specific to Power Aware State Machines	122
Table A-1. Power Aware Arguments for vopt	126
Table A-2. Power Aware Arguments for vsim	127
Table A-3. Power Aware Actions for vopt -pa_enable and -pa_disable	128
Table C-1. List of Supported UPF Commands	173
Table C-2. Supported UPF Package Functions for VHDL	259
Table C-3. Supported UPF Package Functions for SystemVerilog	259
Table C-4. Supported UPF Attributes	262
Table C-5. Power Aware Actions for vopt -pa_upfextensions	266

Chapter 1

Getting Started With Power Aware Simulation

Note



The functionality described in this chapter requires an additional license feature for ModelSim SE. Refer to the section “[License Feature Names](#)” in the Installation and Licensing Guide for more information, or contact your Mentor Graphics sales representative.

Some designs require that you minimize dynamic and static power consumption. A common low-power design technique—power gating—involves switching off certain portions of the design when their operation is not needed and restoring power when operation is needed again. Other low-power design techniques include the use of multi-voltage supplies, state retention, isolation, and level shifting.

ModelSim supports Power Aware simulation for VHDL or Verilog designs that use the above techniques in both register transfer level (PA-RTL) and gate-level (PA-GL) simulation.

To apply Power Aware simulation, you use your conventional ModelSim simulation flow, along with some power-specific options to the `vopt` and `vsim` commands and a power intent specification written using IEEE 1801 Unified Power Format (UPF).

With Power Aware simulation, you can perform functional verification of low power designs together with the power management structures defined by your power intent. Different types of power gating designmanagement structures can be verified, such as:

- Multiple switchable power domains with a single voltage
- Multiple switchable power domains with different (fixed) voltages per domain
- Power domains that can be put into bias mode for state retention with low static leakage

To verify these structures, you create a UPF [Power Intent Specification](#) that includes the following:

- Definition of the power domains within your design and the instances that belong to each power domain.
- Definition of the power states of those power domains, their corresponding primary supplies, and the simstates associated with primary supply states.

- Definition of strategies for inserting, powering, and controlling retention cells in a power domain, to retain state during power down.
- Definition of strategies for inserting, powering, and controlling isolation cells and level shifter cells at power domain boundaries, to mediate interactions between power domains in different states or at different voltage levels.
- Definition of supply networks, including supply ports, nets, and power switches required to provide power to each domain.

Where Is Power Aware in Your Design Flow?

Before you begin to use Power Aware to perform a power aware simulation on your RTL design, you should evaluate where you are in your overall design flow. [Figure 1-1](#) shows an approximation of a typical design sequence and where Power Aware might occur in that sequence.

Before running Power Aware simulation, you should have worked through the following stages of your design flow:

- Design creation
- RTL architecture
- Formal verification
- Definition of power intent

After running a Power Aware RTL simulation, you would typically use the results to make appropriate topology or performance changes to your power-sensitive design blocks. After a gate-level simulation (GLS), you would typically make library cell changes based on performance characteristics.

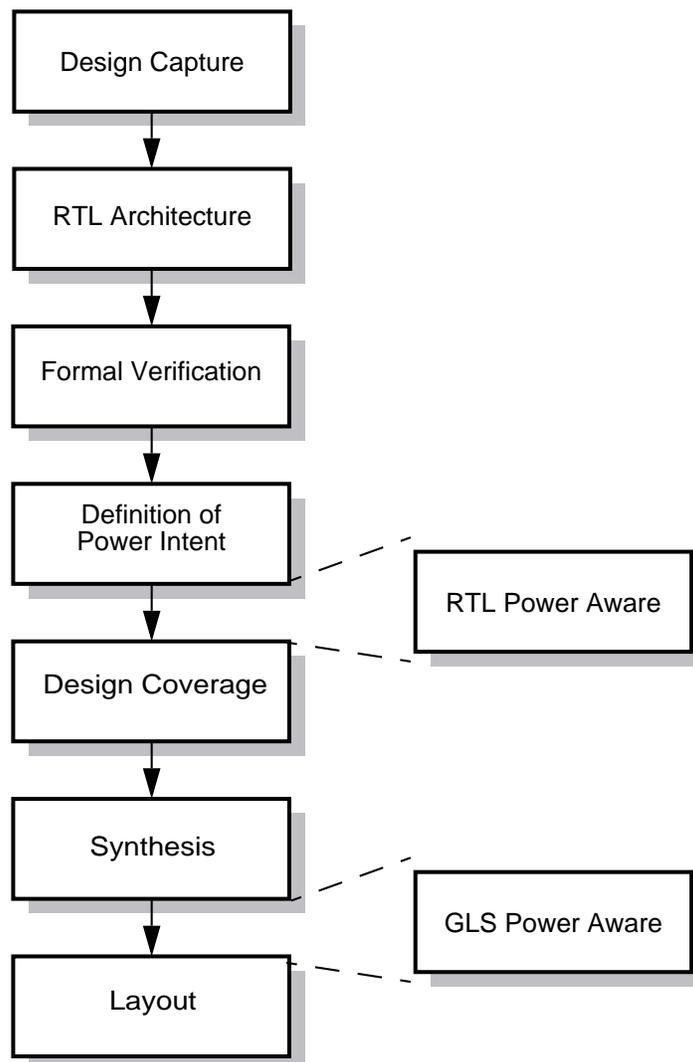
The scope of Power Aware as a low-power solution spans multiple manifestations of design architecture.

- **Power Aware Simulation** — Simulation that includes active power management elements and their behavior. The power management architecture is specified in UPF; the behavior of those elements is inferred from the UPF specification. This relates specifically to Power Aware Simulation.
- **Power Aware Verification** — Collaborative usage of various products and methods for verifying that a design operates correctly under active power management. These include power aware simulation (for verifying the correct operation of the power management architecture), formal verification (for verifying correct operation of power control logic), and hardware/software co-verification (for verifying that software power control interacts correctly with power control logic). This relates to all of the components of the Questa Verification Platform that can be used for Power Aware

Verification, including Questa PASim, Questa ADMS, Questa Formal, Questa Codelink, and Questa VM.

- **Power Efficient Design** — Design of hardware that involves active power management. This includes design decisions involved in allocating power budget, partitioning the design into power domains, and defining the power management architecture, the power control logic, and the power control software, as well as products used for verification and implementation of such designs. This relates to the entire range of Mentor Graphics products that are involved in the design, verification, and implementation of low-power IP, chips, and systems.

Figure 1-1. Typical Location of Power Aware Simulation in Design Flow



Documentation—Scope and Organization

Power Aware simulation augments normal HDL simulation capabilities with the ability to specify, model, and simulate the effects of active power management logic that will be added to the design during implementation.

The success of applying Power Aware features depends on understanding the structure of your design and having the ability to run ModelSim, plus—more generally—your goals of using simulation and verification software products.

The purpose of this manual is to provide basic usage and reference information on how to run a Power Aware simulation in ModelSim. The primary focus is on how to define the power intent of your design and then apply your conventional ModelSim simulation flow to verify power gating behavior.

Note that there are some areas related to Power Aware operation that this manual is not intended to cover:

- UPF standards — The [UPF Commands and Reference](#) appendix lists the UPF commands and arguments that are currently supported for both v1.0 and v2.0 of the UPF standard. However, complete usage information from those standards is not duplicated in this manual.
- Basic and advanced ModelSim usage — Please refer to the other manuals of ModelSim documentation for information on operations related to Power Aware simulation, such as: command usage, the graphical user interface (GUI), design optimization, waveform analysis, and finite state machines.
- Power design — Reporting power estimation or creating an RTL architecture for optimized power implementation.

Specifically, the scope of this manual falls into the following broad areas:

Usage

- Terminology definitions
- Basic operating instructions for Power Aware simulation
- ModelSim commands specific to Power Aware
- Differences between RTL and GLS
- UPF commands for the power specification file
- Reporting of results

Design

- General discussion of low-power design and Power Aware simulation.

- HDL models used for Power Aware simulation.

Flow

- General discussion of low-power analysis as part of overall design flow.
- The distinctions between RTL and gate-level simulations, and the advantages of each at different points in the flow.

Contents of This Manual

This manual contains both an introductory overview of the Power Aware simulation capabilities and detailed reference information about Power Aware simulation features and usage.

- Chapters 2 through 6 provide an introductory overview of basic usage.
- Appendices A through F provide more detailed reference information.

How to Use This Manual

If you are just beginning to learn about Power Aware simulation, read the following chapter:

- [Concepts for Using Power Aware Simulation](#)

If you are looking for an overview of Power Aware simulation concepts and capabilities and how they are used, read the following chapter:

- [Power Aware Simulation](#)

If you are looking for an overview of the information provided by ModelSim during power aware simulation, to help you understand how active power management is working in your design, identify problem areas, and track what has been verified, read the following chapters:

- [Power Aware Reports](#)
- [Automatic Checking](#)
- [Visualization of Power Aware Operations](#)
- [UPF Commands and Reference](#)

For more detailed reference information about specific topics such as UPF and its use, Power Aware simulation commands and flows, report formats, messages, read the appropriate appendices of this manual.

Related Documentation

Other documents that may be useful for understanding Power Aware simulation include the following:

- *ModelSim User's Manual*

This manual explains how to use ModelSim for simulation of hardware designs. It contains descriptions of basic ModelSim usage, especially simulation, optimization, debugging, assertions, and GUI visualization of design source, schematics, and waveforms.

- *ModelSim Reference Manual*

This manual contains a comprehensive listing and description of ModelSim commands, arguments, and values.

- *ModelSim Tutorial*

This manual presents an introductory tutorial on the use of ModelSim for Power Aware simulation. It provides a simple exercise on how to run a Power Aware simulation on RTL design data for the Interleaver example provided as part of the ModelSim installation. This exercise includes creating a UPF power intent specification, defining isolation and retention for the power domain, and evaluating simulation results (waveforms and reports).

- *IEEE 1801™-2009 Standard for Design and Verification of Low Power Integrated Circuits*

This IEEE standard defines the Unified Power Format (UPF), a notation used for specifying power intent for HDL designs.

Chapter 2

Concepts for Using Power Aware Simulation

This chapter provides a brief description of basic usage elements for running Power Aware simulation in the following sections:

- [Power Intent Specification](#)
- [Power Aware Modeling](#)
- [Modeling Corruption](#)
- [Modeling Isolation](#)
- [Modeling Retention](#)
- [Modeling Bias](#)

Power Intent Specification

To perform Power Aware simulation, you need to provide a power intent specification that defines the planned power management architecture for the design. A power intent specification is written using commands defined in the IEEE 1801 Unified Power Format (UPF) standard — For more information on this format, refer to [Unified Power Format \(UPF\) definition](#).

Power Aware Modeling

Verilog and VHDL both make the fundamental assumption that all logic is powered on at the beginning of simulation and remains powered on throughout simulation. However, this assumption is no longer true for most complex devices being designed today. In these systems, power is typically provided to a given portion of a chip only when that part needs to function. This involves inclusion of additional hardware to control power, save state when power is turned off, and mediate interactions between portions of the system that are powered differently.

Power Aware simulation makes it possible to model these power management aspects of a system in simulation, even before the power management features have been implemented in the design. To do so, additional logic is included in the simulation model. This additional logic does the following:

- Defines the power management architecture to be imposed on the design
- Implements the behavior of power management elements
- Adapts the behavior of the design itself to reflect changes in power

To run Power Aware simulation, the normal build process for constructing the simulation model is modified so that this additional logic can be added.

Power Aware simulation provides default Verilog models for the behavior of power management elements and the behavior of the design under active power management. The default models are provided in your installation directory at the following location:

```
<install_dir>/verilog_src/upf_pack
```

With the model mapping capability defined in the power intent specification, you can simulate power on/power off and retention using a model that accurately describes the power down/power up sequence, power down/up behavior, as well as the save and restore sequence and behavior based on actual silicon. You can use Power Aware simulation for both Register Transfer Level (RTL), Gate-Level (GL), and mixed RTL/GL designs.

Power aware simulation involves adding power intent to the design under test, as appropriate. For an RTL design with no power management content, this may involve inferring power domains, retention cells, isolation cells, level shifters, and the power supply network from the UPF power intent specification. For a GL design that has been created by a synthesis tool that reads and implements UPF, some of the power management content may already be present, and therefore less of the power management content may need to be inferred from the UPF specification.

Application of power intent to the design includes addition of functionality to model the corruption of signal values when insufficient power is provided for normal operation. It also includes addition of functionality to model the saving and restoring of system state that will be performed by state retention elements in the power managed system. Addition of this functionality is done automatically as part of the preparation for power aware simulation.



It is also possible for you to construct custom models for these behaviors. Refer to [Appendix B, Model Construction for Power Aware Simulation](#) for more information on Power Aware modeling.

Modeling Corruption

Corruption refers to the change of a signal from its current value to a corrupted value when a power domain is insufficiently powered to operate correctly in a given mode.

Corruption refers to the situation where the value of a signal becomes unpredictable when the power supply for the element driving that signal is disconnected, changes to OFF, or drops below some threshold. Corruption of a signal is represented by assigning a particular value to the signal. The corruption value depends upon the type of the signal and is user-definable. Corruption is typically applied to the drivers of signals and will propagate to all sinks of the signal that have not been isolated from their source.

When a design instance is turned off, every sequential element within the powered-down instance and every signal driven from within the powered-down instance is corrupted. As long as the power remains off, no additional activity takes place within the powered down instance—all processes within the powered down instance become inactive, regardless of their original sensitivity list. Events that were scheduled before the power was turned off and whose target is inside a powered down instance have no effect.

When a design instance is turned on (restored), corruption of sequential elements and signals within the powered down element ends.

Continuous assignments once again become sensitive to changes to their right-hand side expressions, and other combinational processes (such as an `always_comb` block in SystemVerilog) resume their normal sensitivity list operation. All continuous assignments and other combinational processes are evaluated at power up to ensure that constant values and current input values are properly propagated. Sequential elements will be re-evaluated on the next clock cycle after power up.

Corruption Values

Signals are corrupted by assigning them their default initial value (such as X for 4-state types). Default corruption values for Verilog and SystemVerilog are:

- 4-state logic types: 'X'
- 2-state logic types: '0'
- SystemVerilog user-defined types: SystemVerilog default value

Default corruption values for VHDL are:

- Logic types: 'X'
- Real types: 0.0
- For any type T: T`LEFT

Default Corruption Semantics

During power aware simulation, if the driver of a net is powered down, then the driver's output is corrupted, and this corrupted value propagates to all sinks of that net.

To understand how corruption will occur in a given design, it is necessary to understand which elements of the design represent or contain drivers.

- In RTL code, any statements involving arithmetic or logical operations or conditional execution are interpreted as representing drivers and cause corruption when powered down. Unconditional assignments and Verilog **buf** primitives do not represent drivers

and therefore do not cause corruption when powered down, but they may propagate corrupted signals from upstream drivers.

- In Gate-Level code, all cell instances are interpreted as containing drivers. As a result, buffer cell instances in a gate level netlist will cause corruption when powered down. Refer to the section "[Detection of Gate-level Cells](#)" for information on the detection of any particular model as a gate-level cell.

Modeling Isolation

Isolation is used to ensure correct logical and electrical interactions between a powered down domain that is the source of a signal and a powered up domain that is the receiver of that signal. Isolation ensures that the receiving power domain sees a stable, known logic value, while at the same time preventing so-called "crowbar current" due to an indeterminate source value partially enabling both transistors of a CMOS inverter at the same time.

Note



Isolation cells themselves must be powered in order to function correctly. If the power supply for an isolation cell is turned off, the isolation output is corrupted regardless of the state of the source power domain.

A given power domain may be powered off while another domain is operating in normal mode. Isolation ensures the following:

- Powered-down regions do not drive unknown values into the rest of the design (isolation on outputs).
- The rest of the design receives values that are functionally correct (isolation on inputs).

Modeling Retention

Retention consists of saving the value of a design element in a power domain prior to switching off the power to that element, then restoring that value after power to the element is switched back on.

The `set_retention` (and in UPF 1.0 `set_retention_control`) UPF command determines which registers in a power domain need to be retention registers and set the corresponding save and restore signals for the retention registers.

In UPF, you specify a retention strategy where state preservation is required:

- Latch, flip-flop, or memory retention
- Retention power supply
- Retention controls to trigger retention

The general sequence for specifying retention in UPF is:

1. Define your power domains:

```
create_power_domain
```

2. Specify the retention strategy—a set of registers in the domain requiring retention:

```
set_retention
```

3. Specify the retention control signals for the strategy:

```
set_retention (in UPF 2.0)
set_retention_control (in UPF 1.0)
```

In simulation, two additional processes are added for each register that is to be retained, to model the retention behavior:

- One is sensitive to the save signal in accordance to the save sense
- The second is sensitive to the restore signal in accordance to the restore sense.

A retention memory is also created for each sequential element that needs to be retained.

Edge-sensitive and Level-sensitive Control of Retention Models

Power Aware simulation provides default Verilog models for retention cells that support both edge-sensitive and level-sensitive detection of input control signals for save and restore functions. Note that there are separate models for single and dual control signals:

- Single control signal — uses opposite (inverted) edge or level of one input signal to initiate save and restore.
- Dual control signals — uses edge or level of two different input signals to initiate save and restore.

Automatic Model Selection

Based on the retention control signals specified in the UPF specification, Power Aware simulation automatically selects an edge-sensitive or level-sensitive model for retention. Automatic selection occurs according to the following conditions:

- If both `save_signal` and `restore_signal` are level-sensitive and the same signal is used for both, the Single control, level-sensitive model is selected.
- If both `save_signal` and `restore_signal` are level-sensitive and two different signals are used for save and restore: the Dual control, level-sensitive model is selected.
- If any control signal (save or restore) is edge-sensitive and the same signal is used for both save and restore, the Single control, edge-sensitive model is selected.

- If either `save_signal` or `restore_signal` is edge-sensitive and two different signals are used for save and restore, the Dual control, edge-sensitive model is selected.

Level-sensitive Retention Model Protocol Example

The level-sensitive model accurately duplicates the behavior of a level-sensitive Liberty cell. Based on the save/restore level, the retention register switches between normal and retention operations as follows:

- When save is active, normal register behavior occurs—a balloon latch keeps latching the output of the register.
- When save is de-asserted, a balloon latch on the register output saves the register value.
- When restore is active, the saved value is retained.
- When restore is de-asserted, the retained value is loaded into the register again.

The save or restore events are defined as trailing edge of the level-sensitive event. So, for this command, the register output is saved when `save_restore` goes from high to low (the save event), and the retained value is transferred to the register output at the low to high (the restore event) transition of the `save_restore` signal.

The following is an example of the protocol followed by level-sensitive model:

- In the save phase, normal register operation happens: $D \rightarrow Q$ at clock edges. At the save event (defined above), the register output gets latched.
- In the restore phase, D has no effect on Q , so Q gets the retained value. At the restore event, normal operation resumes and Q will get new value of D from next active edge of clock.
- The register output gets corrupted when the primary power or retention power goes off.
- On primary power-up (and also if retention power is on), retention behavior or normal behavior of the register resumes.
- Retention power off corrupts the register output and the retained value, regardless of the primary power.
- For dual control signals, the retained value and register value both get corrupted when save and restore signals are simultaneously active.

Modeling Bias

Power Aware simulation provides support for bias modes that allow for state retention in a different manner. In the power intent specification, you can specify bias power states of the primary supply of a power domain to simulate bias functionality. When you use a bias mode, the power domain is powered on but running with reduced functionality. Depending upon the

particular bias mode involved, activity inside the domain may corrupt the contents of the domain.

Implementing a bias mode performs retention behavior without inserting explicit retention registers. This saves in area usage and also helps to reduce the leakage power. However, the electrical characteristics of the domain during this period prohibit normal logic functioning and thus timing requirements may not be met.

Chapter 3

Power Aware Simulation

Inputs Required for Power Aware Simulation	27
Commands Used For Power Aware Simulation	27
Power Aware Simulation Flows	28
Working with Liberty Libraries.	35
PDU-Based Simulation	37
Power Aware Simulation Debug.	38
Applying Power Intent to the Design	39

Inputs Required for Power Aware Simulation

Power aware simulation requires a complete HDL representation of the design, including simulation models for leaf-level instances. The HDL representation may be expressed in Verilog, SystemVerilog, or VHDL code, or any combination of these languages, and it may be synthesizable RTL code, behavioral RTL code, Gate-Level netlist, or any combination of these forms. Power Aware simulation supports all of the above.

For designs that are, or that include, a Gate-Level netlist, and for designs that include instances of hard macros, a Liberty library may be available that specifies the characteristics of the standard cells and/or hard macros used in the design. Liberty models typically include power-related information that must be considered during power aware simulation. Power Aware simulation reads Liberty libraries to collect this information and takes it into account in building and executing a power aware simulation model.

Power aware simulation also requires a specification of power intent for the design to be simulated. Power intent is specified using the IEEE 1801 Unified Power Format (UPF). Power Aware simulation supports both UPF 1.0 and UPF 2.0.

Commands Used For Power Aware Simulation

You invoke Power Aware simulation with the same commands used for conventional ModelSim simulation, although the optimization (vopt) and simulation (vsim) commands have additional arguments.

For conventional ModelSim simulation, you use the following commands:

- **vlog** or **vcom** — Used to compile Verilog, SystemVerilog, or VHDL source code. For Power Aware simulation, these commands are used in the same manner.

- **vopt** — Used to enable or disable optimization in the HDL design. For Power Aware simulation, the vopt command also processes the UPF power intent specification.
- **vsim** — Used to run simulation on the HDL design. For Power Aware simulation, the vsim command also applies Power Aware simulation semantics to the HDL design.

i For a complete listing of the Power Aware arguments, beyond what is discussed in this section, provided for these commands, refer to [ModelSim Commands Used for Power Aware](#).

General Steps for Running Power Aware

1. Map your libraries:

```
vlib <library_name>  
vmap work <library_name>
```

2. Compile your VHDL or Verilog design files (ignore statements within translate_off/on and synthesis_off/on pragmas):

```
vcom <design_files>  
vlog <design_files>
```

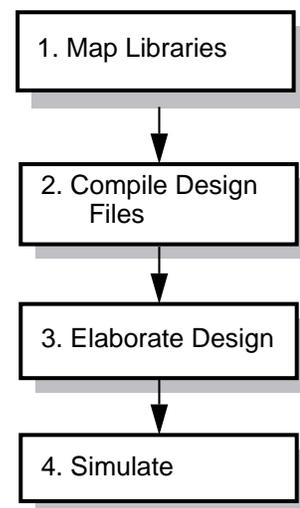
3. Elaborate your top-level design and apply the UPF power intent to the design:

```
vopt <design_top> -pa_upf <upf_file> -o  
<optimized_output>
```

There are multiple methods of using the vopt command to prepare and optimize a Power Aware simulation..

4. Simulate the Power Aware version of your design.

```
vsim -pa <optimized_output>
```



Power Aware Simulation Flows

There are three major flows available for use with Power Aware simulation, including:

- The standard (three-step) flow, where you explicitly invoke an optimization step.
- The delayed optimization (two-step) flow, where optimization is invoked implicitly before simulation.
- The no-optimization flow, where you do not invoke optimization either explicitly or implicitly.

The first two flows are strongly recommended. The optimization step used in these flows, whether invoked explicitly or implicitly, provides the following advantages for power aware simulation:

- Dramatically better performance for Power Aware simulation of RTL designs.
- Support for Power Aware simulation of Gate-Level and Mixed RTL/Gate-Level designs.
- More robust support for isolation of arbitrary SystemVerilog data types.
- Visualization of isolation and level-shifter cells in dataflow and schematic windows.
- Power management objects in structure, waveform, and object windows.
- Support for coverage data collection for power states and transitions and power aware checks.
- Support for user-defined power aware assertions using the UPF `bind_checker` command.
- Automatic power aware testplan generation.

The no-optimization flow is not recommended, except for certain very specific applications that do not yet support the optimized flows. One such application is the use of Power Aware simulation together with Questa ADMS.

Using the Standard Flow

This section describes the three-step standard command flow used to perform Power Aware simulation for RTL, Gate-level, or mixed RTL/Gate-level designs. This flow consists of the following sequence of operations:

1. [Compile](#)
2. [Optimize](#)
3. [Simulate](#)

Compile

Compile your design by running either the `vcom` (for VHDL) or the `vlog` (for Verilog) command as you would for any VHDL or Verilog/SystemVerilog design.

 **Note** Do not use the `-novopt` argument with either `vcom` or `vlog` when you compile for Power Aware simulation.

For more information on using `vcom`, refer to “[Compiling a VHDL Design—the vcom Command](#)” in the User’s Manual.

For more information on using vlog, refer to “[Invoking the Verilog Compiler](#)” in the User’s Manual.

Usage

```
vcom <files>  
vlog <files>
```

Optimize

After you compile the design, use the vopt command with the following arguments:

- pa_top (optional) Specifies the hierarchical name of the design top instance. If unspecified, the default is the topmost instance of the design hierarchy.
- pa_upf Specifies the name of the UPF file containing the power intent specification. Power Aware simulation reads the UPF file and generates information required to run Power Aware simulation.
- pa_lib (optional) Specifies a destination library in which the power aware information will be stored. Note that you must use the vlib command to create the library, then use the library name as the value for this argument. If unspecified, the default is the work library.
- o Specifies the name for the resultant optimized design.

Note that you can also specify any other vopt arguments for conventional optimization to create the optimized design. This sequence is similar to the conventional [three-step optimization flow](#).

You can write the UPF power intent specification in one of two ways: either with respect to the top of the design-under-test (DUT), or with respect to the top of the testbench (TB). If the UPF specification is written with respect to the DUT top, then you must specify vopt -pa_top <pathname> to specify the path from the top module down to and including the DUT instance.

Example 1

If the UPF specification is written with respect to the top of the design-under-test (DUT), and this module is instantiated in the testbench top module TB as TB.dut, then you need to invoke vopt as follows:

```
vopt TB -pa_top TB/dut -pa_lib work -pa_upf DUT.upf -o SimModel [other  
vopt args]
```

where

- TB — the name of the top-level module of the test bench.
- -pa_top TB/dut — the path from the test bench top down to the design top instance.
- -pa_upf DUT.upf — the name of the UPF file written with respect to the DUT top-level module.

- `-pa_lib work` — the name of the destination library.
- `-o SimModel` — the name of the optimized output for your vsim run.
- `[other args]` — any other vopt arguments used to control optimization.

Example 2

If the UPF specification is written with respect to the top of the testbench (TB), then you need to invoke vopt as follows:

```
vopt TB_top -pa_upf TB.upf -pa_lib work -o SimModel [other vopt args]
```

where

- `TB_top` — the name of the top-level module of the test bench.
- `-pa_upf TB.upf` — the name of the UPF file written with respect to `TB_top`.
- `-pa_lib work` — the name of the destination library.
- `-o SimModel` — the name of the simulation-ready output file to be generated.
- `[other vopt args]` — any other vopt arguments used to control optimization.

Usage Notes

Power Aware simulation involves register/latch detection in order to identify state elements in the design that need to be corrupted and may need to have their state retained during power down. However, your design may contain code that is not at the appropriate level of abstraction for register/latch detection, so you may need to exclude such code from power intent processing. Also, you may want to exclude parts of the design from Power Aware simulation for other reasons (see “[Excluding Design Elements from Power Aware](#)” in Appendix A).

ModelSim provides additional vopt arguments for Power Aware simulation that you can use when generating the optimized design. For more information on these arguments, refer to the section “[ModelSim Commands Used for Power Aware](#)”.

A simulation model created for Power Aware simulation contains specific Power Aware simulation artifacts and cannot be used for normal simulation.

Simulate

After you compile and optimize the design, run the vsim command on the optimized design using the `-pa` argument to perform Power Aware simulation.

- | | |
|----------------------|--|
| <code>-pa</code> | Enables PA simulation, including features for gate-level simulation. |
| <code>-pa_lib</code> | (optional) Instructs Questa SIM to load Power Aware information from the specified library. If unspecified, defaults to the current directory. |

Example

```
vsim SimModel -pa -pa_lib work [other vsim args]
```

where

- SimModel — the name of the simulation-ready output file generated by vopt.
- -pa — invokes simulation in Power Aware mode.
- -pa_lib work — loads power aware information from library work.
- [other vsim args] — any other vsim arguments used to control simulation.

Using the Delayed Optimization Flow

This section describes the two-step, delayed optimization command flow used to perform Power Aware simulation for RTL, Gate-Level, or mixed RTL/Gate-Level designs. This flow consists of the following sequence of operations:

1. Compile
2. Simulate

In conventional simulation, you can perform what is referred to as a [two-step optimization flow](#). In this flow, you do not use the -o argument for vopt, so that optimization is delayed until you invoke the vsim command, at which point it is done implicitly.

Note



This delayed optimization flow is also supported for Power Aware simulation. This flow may be useful in certain cases, such as automated scripts that assume only two steps are involved in building and running a simulation.

Compile

Compile your design by running either the vcom (for VHDL) or the vlog (for Verilog) command as you would for any VHDL or Verilog/SystemVerilog design.

Note



Do not use the -novopt argument with either vcom or vlog when you compile for Power Aware simulation.

For more information on using vcom, refer to “[Compiling a VHDL Design—the vcom Command](#)” in the User’s Manual.

For more information on using vlog, refer to “[Invoking the Verilog Compiler](#)” in the User’s Manual.

Usage

```
vcom <files>  
vlog <files>
```

Simulate

To implement simulation for this flow, run the `vsim` command on the test bench using the `-pa` argument, along with the `-voptargs` argument (specifying a UPF file) to invoke Power Aware simulation. When you invoke `vsim` on the test bench top module, it implicitly performs UPF processing and optimizes the design before beginning the Power Aware simulation.

Example

```
vsim TB_top -pa  
-voptargs="-pa_upf test.upf -pa_lib work [other vopt args]"  
[other vsim args]
```

where

- `TB_top` — is the name of the top-level module of the test bench.
- `-pa` — invokes simulation in Power Aware mode.
- `-voptargs` — instructs Questa SIM to apply arguments for the `vopt` command. For this flow, specify the following `vopt` arguments:
 - `-pa_upf <filename>` — to cause `vopt` to apply the power intent specification.
 - `-pa_lib work` — the name of the destination library.
 - `[other vopt args]` — any other `vopt` arguments used to control UPF processing or optimization.
- `[other vsim args]` — any other `vsim` arguments used to control simulation.

Using the No-Optimization Flow

In conventional simulation, you can perform simulation without optimizing the design at all. This no-optimization flow is also supported for Power Aware simulation.

In this flow, the `vopt` step is still performed, but only to process the UPF specification; you do not use the `-o` argument for `vopt`, so optimization is not performed until you invoke the `vsim` command, use the `-novopt` argument to prevent implicit optimization from being performed for the simulation session.

Note



This flow is **not recommended** for use except for certain very specific applications that do not yet support the optimized flows. One such application is the use of Power Aware simulation together with Questa ADMS.

Compile

Compile your design by running either the `vcom` (for VHDL) or the `vlog` (for Verilog) command as you would for any VHDL or Verilog/SystemVerilog design.

For more information on using `vcom`, refer to “[Compiling a VHDL Design—the vcom Command](#)” in the User’s Manual.

For more information on using `vlog`, refer to “[Invoking the Verilog Compiler](#)” in the User’s Manual.

Usage

```
vcom <files>  
vlog <files>
```

Process UPF

After you compile your design, run the `vopt` command on the test bench top with the `-pa_upf` argument and `TB.upf` but without the `-o` argument, to process the UPF power intent specification and apply it to the design. You must also specify the `-pa_enable=nonoptimizedflow` option. Power Aware simulation reads and processes the UPF file without generating an optimized output.

Example

```
vopt TB_top -pa_upf TB.upf -pa_lib work -pa_enable=nonoptimizedflow [other  
vopt args]
```

where

- `TB_top` — the name of the top-level module of the test bench.
- `-pa_upf TB.upf` — the name of the UPF specification written with respect to the test bench top-level module.
- `-pa_lib work` — the name of the destination library.
- `-pa_enable=nonoptimizedflow` — required for any flow using `-novopt`
- `[other vopt args]` — any other `vopt` arguments used to control UPF processing or optimization.

Simulate

After you have compiled your design and processed the UPF file, run the `vsim` command on the test bench using the `-pa` and `-novopt` arguments to invoke Power Aware simulation with no optimization on the design.

Example

```
vsim TB_top -pa -novopt -pa_lib work [other vsim args]
```

where

- TB_top — is the name of the top-level module of the test bench.
- -pa — invokes simulation in Power Aware mode.
- -novopt — instructs ModelSim to disable optimization.
- -pa_lib work — the name of the destination library.
- [other vsim args] — any other vsim arguments used to control simulation.

Working with Liberty Libraries

Liberty Library Models

Liberty libraries define standard cells and macro cells that can be used in a system. Liberty models include information that defines how power supplies provided to instances of such cells are used to power logic receiving inputs or driving outputs of those cell instances. Liberty models also include other information pertaining to power management cells.

The information contained in a Liberty library may be of use in both RTL and Gate-Level simulations as well as in mixed RTL/GL simulations. For example, Liberty models of hard macros may be of use in applying power intent to the RTL portion of a design, and information about standard cells in a Liberty library may be of use in applying power intent to the gate-level portion of a design.

Power Aware simulation uses the information present in the Liberty library to create proper connection of the supplies and the control signals that are defined in the UPF power intent specification. Power Aware simulation also uses such information to assist in automatic recognition of power management cells that are present in a design.

Liberty libraries can be preprocessed to extract power-related information into a database that can be used during UPF processing in preparation for power aware simulation. This preprocessing step enables most efficient use of Liberty libraries when the same library will be used many times. Liberty libraries can also be read directly during the UPF processing step for a single simulation run.

Using Liberty Files with vopt

You can specify one or more Liberty files to use with the vopt command with the -pa_libertyfiles argument. By default, using Liberty files creates an internal database in which power-related data extracted from a Liberty library is stored.

You can control how this database is used with the following vopt arguments:

- -pa_dumplibertydb
- -pa_libertyupdate
- -pa_libertyrefresh
- -pa_loadlibertydb

Creating and Saving a Liberty Database

Use the vopt command to parse the Liberty files and create a saved database containing information extracted from the Liberty files that will be used in power aware simulation.

Note that you can specify multiple files with the -pa_libertyfiles argument by separating file names with a comma.

```
vopt -pa_libertyfiles=a.lib,b.lib -pa_dumplibertydb=lib_db
```

where:

- -pa_libertyfiles — specifies the Liberty files to read.
- -pa_dumplibertydb — specifies the name of the Liberty database to be saved for future use.

Using a Previously Created Liberty Database

Use the vopt command again with other Power Aware arguments to include information from a previously created Liberty database in the UPF processing for a design.

```
vopt design_top -pa_upf compile.upf -pa_lib work -pa_loadlibertydb=lib_db  
[other vopt args]
```

where

design_top	the name of the top-level module of the test bench.
-pa_upf compile.upf	the name of the UPF file written with respect to the design top module.
-pa_lib work	the name of the destination library.
-pa_loadlibertydb	the name of the Liberty database from which to load information.

Using Liberty Files Directly

Liberty files can also be read in directly when processing UPF power intent, without creating a database first.

The following command

```
vopt design_top -pa_upf compile.upf -pa_lib work -  
pa_libertyfiles=a.lib,b.lib
```

analyzes *a.lib* and *b.lib* files and creates an internal database for these libraries. This internal database is used for Liberty data in a Power Aware analysis; the internal database is deleted at the end of the vopt run.

Updating a Liberty Database

If a Liberty database already exists for a Liberty library, you can replace the existing database with a new one using the `-pa_libertyupdate` argument.

Assuming that a Liberty database has been created already at location `/home/user/libdbs/LVT` for Liberty files *a.lib*, *b.lib*, the command:

```
vopt -pa_libertyfiles=a.lib,b.lib -pa_dumplibertydb=/home/user/libdbs/LVT  
-pa_libertyupdate
```

which analyzes *a.lib* and *b.lib* files again and overwrites the previous database with a new one. This can be used when a *.lib* file has been edited to correct an error and needs to be reprocessed.

Refreshing a Liberty Database

If a Liberty database has been created already with a previous version of Power Aware simulation, the database can be modified to work with a later version of Power Aware simulation by using the `-pa_libertyrefresh` argument.

```
vopt design_top -pa_upf compile.upf -pa_lib work -pa_loadlibertydb=lib_db  
-pa_libertyrefresh [other vopt args]
```

This will refresh 'lib_db' to be used with current version of ModelSim.

PDU-Based Simulation

In conventional simulation, it is often desirable to optimize the design-under-test (DUT) separately from the test bench, so that you can use the same optimized DUT model (pre-optimized design unit, or PDU) with multiple test benches. You can also apply this approach to run Power Aware simulation, provided that the DUT appears at the same location in each test bench.

To use this approach, include the `-pa_defertop` argument on the vopt command (in the three-step, standard flow) or in the `-voptargs` argument to vsim (in the two-step, delayed optimization flow). For example,

```
vopt DUT_top -pa_defertop -pa_upf DUT.upf [other vopt args]
```

where

- DUT_top — specifies the name of the top-level module of the design under test.
- -pa_defertop — specifies that the location of the DUT will be defined later.
- -pa_upf DUT.upf — specifies the name of the UPF file written with respect to the DUT top-level module.
- [other vopt args] — any other vsim arguments used to control simulation.

Then in simulation, include the -pa_top argument on the vsim command, for example:

```
vsim TB_top -pa_top <DUT_path> -pa [other vsim args]
```

where

- TB_top — is the name of the top-level module of the test bench.
- -pa_top <DUT_path> — is the path from the test bench top to the design top instance.
- -pa — invokes simulation in Power Aware mode.
- [other vsim args] — any other vsim arguments used to control simulation.

Limitations

Only one module can be specified as a PDU with -pa_top in a given simulation.

Power Aware Simulation Debug

This section provides additional information on setting up your Power Aware simulation for post-sim debug.

Capturing Information for Post-simulation Debug

Power Aware simulation allows you to create a database of information that for analyzing the results after completion of a live simulation.

This is most useful when you need to debug a power aware simulation that has been run in a regression setup or another situation where you are not able to debug a live simulation.

The database contains information that allows you to perform supply network debugging and information for colorizing power information in the Dataflow window.

To create the post-simulation debug database, use the **-pa_enable=supplynetworkdebug** option on the vopt command (in the three-step, standard flow) or in the -voptargs argument to the vsim command (in the two-step, delayed optimization flow), for example:

```
vopt top -pa_upf top.upf -o DebugOut -pa_enable=supplynetworkdebug  
[other vopt args]
```

Then, when you run simulation, add the **-pa_debugdir <directory>** option, which creates the necessary debug information for you to access post-simulation, for example:

```
vsim -pa_debugdir padebug -pa DebugOut -do "log -r /*; run -all"
```

After simulation has completed, load the WLF file and debug database to perform any debug activities on the results:

```
vsim -view vsim.wlf -pa_debugdir padebug -pa
```

Debugging Designs Containing Liberty Cells

To enable debugging of designs containing Liberty cells, you need to specify the location of the Liberty library file(s) and you also need to specify the **-debugdb** argument as part of the **vopt** command. For example:

```
vopt -libertyfiles=a.lib,b.lib -debugdb
```

This enables schematic viewing and causality analysis using Liberty logic cell definitions.

Note



You can also enable debugging and schematic viewing by setting the **MTI_LIBERTY_PATH** environment variable to the directory location containing Liberty library source (.lib) files prior to running the **vopt** step.

Applying Power Intent to the Design

When you run a mixed RTL and gate-level design through the power aware flow the following steps are performed automatically:

1. Detect gate-level cells in the mixed netlist and perform the application of corruption based on the corresponding UPF strategies.
2. Detect the power aware cells in the design and match them with corresponding UPF strategies at that point in the design and run and flag power aware checks to report the valid cases and error out for anomalous cases.
3. Insert power aware cells wherever required by the corresponding UPF strategy and not already present in the netlist.
4. Detect UDPs and apply corruption and retention based on the corresponding UPF strategy.

Detection of Gate-level Cells

Power Aware simulation automatically treats a module as a gate-level cell if either of these conditions are satisfied:

- the module contains the ``celldefine` attribute
- the module contains the `specify` block

Corruption will only occur on the output ports of any detected gate-level cells. For RTL cells the standard processing of driver-based corruption still occurs.

In some cases, you may not want this flow to corrupt certain cells. For example, PAD cells of buffer cells which might be written with a ``celldefine` attribute in the design. There may be other cases where you don't want a cell to be treated as a gate-level cell, but would rather have driver-based corruption on the modules. For these instances, you can specify the `upf_is_leaf_cell` attribute along with `set_design_attributes` command for the specific module

Detection of Power Management Cells

Power Aware simulation automatically detects isolation, level-shifter and retention cells in the netlist and infers them for any corresponding UPF strategies. Power Aware simulation will then validate the cells, with respect to strategies in the UPF, and run power aware check on them.

Gate-level netlists may have some of the power management cells (isolation, level-shifter, or retention) corresponding to the UPF strategy already instantiated in the netlist. Some of these cells may already be specified as a value for the `-instance` argument of the `set_isolation`, `set_level_shifter`, or `set_retention` commands of the UPF strategy in UPF file.

For the rest of the cells that are not specified in the UPF file, Power Aware simulation automatically detects the right UPF strategy to which they belong and treats them in a similar way to cells of that strategy specified with an `-instance` argument.

Auto detection of power management cells leverages the following information for:

- Liberty attributes
 - `is_isolation_cell`
 - `is_level_shifter`
 - `retention_cell`
- `lib_cells` specified with the following UPF commands:
 - `map_isolation_cell`
 - `map_level_shifter cell`
- Arguments for the UPF `name_format` command

- -level_shift_prefix
- -level_shift_suffix
- -isolation_prefix
- -isolation_suffix
- Synopsys pragmas
 - isolation_upf
 - retention_upf

Reports

The cells detected as an instance of some strategy are reported in *report.upf.txt*, for example:

```
Power Domain: A, File: ./src/case1/test.upf(11).
  Creation Scope: /tb/dut
  ...
  Isolation Strategy: ISO1, File: ./src/case1/test.upf(22).
    Isolation Supplies:
      power : /tb/dut/VDD_0d99
      ground : /tb/dut/VSS_0d99
    Isolation Control (/tb/dut/restore), Isolation Sense (HIGH), Clamp
    Value (0), Location (fanout)
    Signals with -instance isolation cells:
      1. Signal : /tb/dut/instA/out, isolation cell :
      /tb/dut/iso_1_UPF_ISO
```

Messages

When Power Aware simulation is unable to detect the UPF strategy of an isolation/level-shifter cell, Power Aware simulation semantics are disabled and the cell is treated as always ON. The following message is displayed:

```
** Warning: (vopt-9768) Power aware simulation semantics disabled for
'/tb/dut/instA/ls_0_UPF_LS' as its power aware strategy could not be
identified.
```

For a cell identified as a retention cell, Power Aware simulation flags a warning if the cell is also identified as a level-shifter or isolation cell. Power Aware simulation then processes it as either a retention or an isolation/level-shifter cell, and the following message is displayed:

```
** Warning: UPF: (vopt-9823) Power aware cell '/tb/dut/iso_1' identified
as both 'isolation' cell and 'retention' cell. Assuming it to be a
'isolation' cell
```

Automatic Insertion of Power Management Cells

Power Aware simulation inserts power management cells (level-shifter, isolation, and retention) into the netlist based on UPF strategy commands. You can selectively disable each type of insertion with the following vopt arguments:

- `-pa_disable=insertiso` — to disable isolation cell insertion
- `-pa_disable=insertls` — to disable level_shifter cell insertion
- `-pa_disable=insertret` — to disable retention cell insertion

For pure gate-level designs in which all isolation, level shifting, and retention cells are already all present, and therefore none of these cells need to be inserted, the following vopt argument can be used as a shorthand:

- `-pa_gls` — equivalent to:
`-pa_disable=insertiso -pa_disable=insertls -pa_disable=insertret`

Automatic Corruption and Retention of UDPs

Power Aware simulation detects sequential UDPs in the design and applies corruption or retention behavior based on the corresponding UPF strategies.

This functionality is not supported in the no-optimization flow. In that flow, a warning will be issued if there is any sequential UDP in the design.

UDP Corruption and Retention Modes

This section defines different modes of operation of the retention and corruption logic for sequential UDPs in a gate-level or mixed RTL and gate-level simulation. These modes describe the functionality of different sequential UDPs introduced to mimic corruption and retention behavior.

- Save mode (single control) — Occurs when save is active.
- Save mode (dual control) — Occurs when save is active and restore is inactive.
- Restore mode (single control) — Occurs when save is inactive.
- Restore mode (dual control) — Occurs when save is inactive and restore is active.
- Error mode (dual control) — When the UPF **set_retention** parameter `SAV_RES_COR` is true the output of the register and the retained value of the register are both corrupted if the save and restore both are active and **save_condition** and **restore_condition** are true. This occurs only when neither the save signal nor the restore signal are edge sensitive.

- No change mode (dual control) —Occurs when save and restore are both inactive. In this case, the register is neither in save nor in restore mode. Therefore, the behavior would be as if it is a normal register with no retention facility present.

Note

User defined models specified using the **map_retention_cell** command will not be honored for sequential UDP retention. However the RTL portion of design will follow the user defined model simulation semantics given with **map_retention_cell** command.

Limitations

- Power Aware simulation is unable to determine whether retention logic is already present in a sequential UDP and therefore routinely inserts retention logic for such UDPs. Use the UPF `set_retention` command with the `-instance <inst_name>` option to avoid application of retention semantics to UDPs that already include retention logic.
- Power Aware simulation is unable to determine whether a UDP is power aware--that is, whether the UDP references supply ports and corrupts its outputs based on the values of those supply ports. Consequently Power Aware simulation assumes that all UDPs are not power aware and routinely applies corruption on UDP logic.
- Power Aware simulation is unable to process sequential UDPs that involve multiple clocks.

Chapter 4

Power Aware Reports

You can use the `vopt` and `pa report` commands in a two-step process to generate reports for a Power Aware simulation run, and then examine them to validate the application of the power intent on your design.

This flow generates power aware reports containing the following:

- The intent of the low-power defined in the power specification file.
- For UPF, either static or dynamic information on the current Power Aware simulation (including connections to supply and ground nets). See [UPF Reports](#).
- Any additions you may need to make to the specification file to achieve correct Power Aware functionality. For example, certain sections of the design that are non-synthesizable (such as a test bench) should not be inside a power domain. The report may suggest that you put this section of the design inside an always-on power domain.
- Behavioral constructs that must be in an always-on power domain.

Generating Reports for Power Aware

You can use different values with the `-pa_genrpt` argument of `vopt` to generate a variety of Power Aware data for the reports. [Table 4-1](#) lists the values for `-pa_genrpt` and their corresponding reports for UPF.

Power Aware reports are generated in two steps:

1. Use the `vopt` command with the following options:
 - `-pa_genrpt` — generates the necessary information.
 - `-pa_reportdir` — (optional) changes the default location of where the reports are written.
2. Use the `pa report` command from the `vsim` prompt to write out the reports.

How to Generate a Report with `vopt -pa_genrpt`

Syntax

```
vopt -pa_genrpt=[[nv | v][+{ud | us | u}][+b][+pa][+de]]
```

Note



For additional reference information on the `-pa_genrpt` argument, refer to the [vopt](#) command in the ModelSim Reference Manual.

Description

- The `-pa_genrpt` argument generates data for the report files listed in [Table 4-1](#).
- If you do not specify a value for the `-pa_genrpt` argument, all data is generated.
- To specify more than one reporting value for `-pa_genrpt`, use the plus sign (+) operator between values. For example:

```
vopt -pa_genrpt=nv+us+de
```

- When you execute the `pa report` command to write out the reports, the default location is the current working directory. To change the location where report files are saved, add the following argument to your `vopt` command:

```
vopt -pa_reportdir <pathname>
```

UPF Reports

If you are using a UPF file for your Power Aware analysis, you can generate the data for the following reports by specifying values for `vopt -pa_genrpt` described in [Table 4-1](#):

- [UPF Power Intent Report](#) (report.mpsa.txt)
- [UPF Static Report](#) (report.upf.txt)
- [Static Checking UPF Reports](#) (report.static.txt, report.nretsyncff.txt)
- [Dynamic UPF Report](#) (displayed in transcript window)
- [Architecture Report](#) (report.pa.txt)
- [Design Elements Report](#) (report.de.txt)

Table 4-1. Generating UPF Reports for Power Aware

Report Type	vopt -pa_genrpt=	Description	Report Output
Power Intent: Non-verbose (default)	nv	Displays a count of the Power Aware elements in the design with respect to power domains, along with additional information on power intent. This is the default if you do not specify nv or v.	report.mpsa.txt

Table 4-1. Generating UPF Reports for Power Aware (cont.)

Report Type	vopt -pa_genrpt=	Description	Report Output
Power Intent: Verbose	v	Displays the hierarchical path of individual Power Aware elements.	report.mpsa.txt
Dynamic	ud	Includes time and polarity of controls (such as control port of a power switch, retention save and restore signals, isolation enable signal), plus Power Domain Status in the form of <Power Domain name>, <Strategy name>, <Control signal names>, <Active Sense>, <Current polarity value>.	Transcript window (no file)
Static	us	Includes power domains and their supplies, power switches, retention, isolation, level-shifting strategies, and power state tables.	report.upf.txt
Dynamic and Static	u	Combined dynamic and static reporting.	Transcript window (dynamic) report.upf.txt (static)
Bitwise expanded port data	b	Writes bitwise expanded information for isolated and level shifted ports.	report.upf.txt
Architecture	pa	Lists all information related to the Power Aware architecture in the design.	report.pa.txt
Design Elements	de	Contains information on power design elements in the design.	report.de.txt
Isolation Cell Information	u+c	Dumps hierarchical path of isolation cells placed for a candidate port.	report.upf.txt

UPF Power Intent Report

Report Output: report.mpsa.txt

Command: vopt -pa_genrpt=nv | v

This report file contains information related to the power intent that has been applied on your design. File contents are structured according to power domains and provide hierarchical path names of the signals and instances that are affected by the power intent. This report also

contains the count of retention and non-retention cells. The *report.mspa.txt* file does not report on objects/hierarchy/instances that are excluded from Power Aware.

```
1 Total ( tb )
2   upf_retention_ret # 2
3     NPM_FF # 1
4     NPM_LA # 3
5     OUTPUT # 3
6 pd sub_total ( /tb/top_vh /tb/top_vl )
7
8 upf_retention_ret # 2
9   /tb/top_vh/q_regvh 1
10  /tb/top_vl/q_regvl 1
11
12 NPM_LA # 2
13   /tb/top_vh/q_latvh 1
14   /tb/top_vl/q_latvl 1
15
16 OUTPUT # 2
17   /tb/top_vh/q_combvh 1
18   /tb/top_vl/q_combvl 1
```

- Line 1 — The keyword Total followed by the top design module specified in parentheses.
- Lines 2 - 5 — A consolidated count of all the types of cells in the design, in the format:

```
<CELL_TYPE> # <Count>
```

where CELL_TYPE could be any of the following:

- NPM_FF — Represents non-power management flip-flops or corrupt flip-flops, which do not have retention associated with them. During simulation, they get corrupted only when power domain is switched off.
 - NPM_LA — Represents non-power management latches or corrupt latches, which do not have retention behavior associated with them. During simulation, they get corrupted only when power domain is switched off.
 - OUTPUT — Represents combinatorial logic that belong to the power domain. During simulation, they get corrupted only when power domain is switched off.
 - USER_DEFINED — For any user-defined retention models, the names of the model will be printed.
- Line 6 — After the consolidated counts, all the power domains are then listed in the following format:

```
<PD_NAME> sub_total (<Instance hierarchial path> ...)
```

The instance hierarchical paths indicate those instances that contain objects affected by the current power domain.

- Line 8 — The individual CELL_TYPES for that power domain.

- Lines 9-10 — The hierarchical paths of objects belonging to those CELL_TYPES are then listed with the individual count. These lines only appear in verbose mode (-pa_genrpt=v)

Example of UPF File and Power Intent Report

Example 4-1 shows an excerpt from a UPF specification file that defines power domains, supply ports, switching behavior, retention strategy, and isolation strategy.

Example 4-2 shows an excerpt of a power intent report.

Example 4-1. UPF File Excerpt

```
upf_version 1.0

set_scope tb
create_power_domain pd_aon -include_scope
create_supply_port vdd_port -domain pd_aon
create_supply_port gnd_port -domain pd_aon
create_supply_net vdd_net -domain pd_aon
create_supply_net gnd_net -domain pd_aon
connect_supply_net vdd_net -ports { vdd_port }
connect_supply_net gnd_net -ports { gnd_port }
set_domain_supply_net pd_aon -primary_power_net vdd_net -
primary_ground_net gnd_net

#####
#pd Power Domain
#####
create_power_domain pd -elements { top_vl top_vh }
create_supply_port V_pd_port -domain pd
create_supply_port G_pd_port -domain pd
create_supply_net V_pd_net -domain pd
create_supply_net G_pd_net -domain pd
create_supply_net pd_pwr -domain pd
#####
## connect supply ports to supply nets
#####
connect_supply_net V_pd_net -ports { V_pd_port }
connect_supply_net G_pd_net -ports { G_pd_port }
set_domain_supply_net pd -primary_power_net pd_pwr -
primary_ground_net G_pd_net
#####
# Header switch for pd
#####
create_power_switch pd_sw \
    -domain pd \
    -output_supply_port { out_sw_pd pd_pwr } \
    -input_supply_port { in_sw_pd V_pd_net } \
    -control_port { ctrl_sw_pd pwr } \
    -on_state { normal_working in_sw_pd { ctrl_sw_pd } } \
    -off_state { off_state {!ctrl_sw_pd} }
#####
# Retention Strategy for pd
#####
```

```
set_retention          pd_retention -domain pd -retention_power_net
V_pd_net
set_retention_control pd_retention -domain pd -save_signal { ret posedge }
-restore_signal { ret negedge }
map_retention_cell    pd_retention -domain pd -lib_model_name
upf_retention_ret     -lib_cell_type FF_CKHI

#####
# isolation Strategy for pd
#####
set_isolation          pd_isolation -domain pd -isolation_power_net
V_pd_net -clamp_value 1 -applies_to outputs
set_isolation_control pd_isolation -domain pd -isolation_signal iso
-isolation_sense high -location parent
```

Example 4-2. UPF Power Intent Report

```
-----
---- QuestaSim Power Aware Report File ----
-----

Total ( tb )

upf_retention_ret # 2

NPM_FF # 1

NPM_LA # 3

OUTPUT # 3
-----

pd sub_total ( /tb/top_vh /tb/top_vl )

upf_retention_ret # 2
  /tb/top_vh/q_regvh 1
  /tb/top_vl/q_regvl 1

NPM_LA # 2
  /tb/top_vh/q_latvh 1
  /tb/top_vl/q_latvl 1

OUTPUT # 2
  /tb/top_vh/q_combvh 1
  /tb/top_vl/q_combvl 1
-----

pd_aon sub_total ( /tb/top_aon )

NPM_FF # 1
  /tb/top_aon/q_regvl 1

NPM_LA # 1
  /tb/top_aon/q_latvl 1

OUTPUT # 1
  /tb/top_aon/q_combvl 1
```

```
-----  
-- NPM_FF => Denotes all Non Power Management Flip Flops of a Power  
Domain.  
-- NPM_LA => Denotes all Non Power Management Latches of a Power Domain.
```

UPF Static Report

Report Output: `report.upf.txt`

Command: `vopt -pa_genrpt={us | u}[+b]`

The UPF static report typically contains the information that was specified in the specification file and how Power Aware simulation has processed that information. You can use this report to statically validate the power intent (specified by UPF) and check whether it is properly applied.

This report file contains information regarding the power domain, power supplies, power switches and their characteristics, retention, isolation and level-shifter strategies and information regarding the power states and power state tables (PSTs).

Specifying a value of `b` writes bitwise expanded information for isolated and level-shifted ports into this report file.

Note



The content and structure of this report is same as that of *report.pa.txt* and includes all objects that are either added or modified by applying power intent to the design.

The following sections are described in the section about *report.pa.txt*:

- [Power Domain](#) (including supplies)
- [Power Switch](#) (including supplies)
- [Retention Strategy](#) (including supplies)
- [Isolation Strategy](#)(including supplies)
- [Level Shifter Strategy](#)
- [Power State Tables \(PSTs\)](#)

Example of UPF Static Report File

Example 4-3. Example File for Static UPF Reporting (report.upf.txt)

```
-----  
----- Questa Power Aware UPF Report File -----  
-----  
Power Domain: pd, File: ./src/supply1_prefix/test.upf(7).  
Creation Scope: /tb/top  
Primary Supplies:  
    power : /tb/top/pd_pwr
```

```
ground : /tb/top/G_pd_net
Power Switch: pd_sw, File: ./src/supply1_prefix/test.upf(35).
Output Supply port:
  out_sw_pd(/tb/top/pd_pwr)
Input Supply ports:
  1. in_sw_pd1(/tb/top/V_pd_net1)
  2. in_sw_pd2(/tb/top/V_pd_net2)
Control Ports:
  1. ctrl_sw_pd1(/tb/top/pwr1)
  2. ctrl_sw_pd2(/tb/top/pwr2)
Switch States:
  1. normal_working2(ON) : ( (ctrl_sw_pd2 && !ctrl_sw_pd1) )
  2. normal_working1(ON) : ( (ctrl_sw_pd1 && !ctrl_sw_pd2) )
  3. off_state(OFF) : ((!ctrl_sw_pd1 && !ctrl_sw_pd2))

Retention Strategy: pd_retention, File: ./src/supply1_prefix/test.upf(39).
Retention Supplies:
  power : /tb/top/V_pd_net
  ground : /tb/top/G_pd_net
Retention SAVE (/tb/top/ret), Retention Sense (posedge)
Retention RESTORE (/tb/top/ret), Retention Sense (negedge)

Isolation Strategy: pd_isolation, File: ./src/supply1_prefix/test.upf(45).
Isolation Supplies:
  power : /tb/top/V_pd_net
  ground : /tb/top/G_pd_net
Isolation Control (/tb/top/iso), Isolation Sense (HIGH), Clamp Value (1),
  Location (parent)
Isolated Signals:
  1. Signal : /tb/top/q_combvl
  2. Signal : /tb/top/q_latvl
  3. Signal : /tb/top/q_regvl
Level Shifter Strategy: pd_ls, File: ./src/supply1_prefix/test.upf(51).
Rule (both), Threshold (0), Applies_to (both), Location (automatic).
Level Shifted Candidate Ports:
  1. Signal : /tb/top/q_combvl
  2. Signal : /tb/top/q_latvl
  3. Signal : /tb/top/q_regvl
  4. Signal : /tb/top/set
  5. Signal : /tb/top/reset
  6. Signal : /tb/top/clk
  7. Signal : /tb/top/d
  8. Signal : /tb/top/iso
  9. Signal : /tb/top/ret
  10. Signal : /tb/top/pwr2
  11. Signal : /tb/top/pwr1
```

Static Checking UPF Reports

Report Output: report.static.txt

Command: vopt -pa_checks=s

In Power Aware, you can perform a static check of the design using the power intent specified in the UPF file.

- report.static.txt — The pa report command generates this text file if you used the -pa_checks argument to perform static checks.

For Static checking, Power Aware simulation verifies the design for the defined power intent without doing actual simulation. This static analysis uses the power intent definition to report any missing power-related information or inconsistencies. Because a simulation run is not

required, you only need to run `vopt` to perform static checking. Results from static checking are written to the `report.static.txt` report file.

Report Output: report.nretsyncff.txt**Command: `vopt -pa_checks=npu`**

This report file is generated only when you specify `vopt -pa_checks=npu`. It contains a list of hierarchical paths of those flip-flops in the design that do not have an asynchronous control signal.

Dynamic UPF Report

Report Output: Transcript window**Command: `vopt -pa_genrpt={ud | u}[+b]`**

Specifying a value of `ud` or `u` displays dynamic UPF information to the Transcript window.

The dynamic report display includes time and polarity of controls (such as control port of a power switch, retention save and restore signals, isolation enable signal), plus Power Domain Status in the form of:

```
<Power Domain name>, <Strategy name>, <Control signal names>, <Active Sense>,  
<Current polarity value>
```

Specifying a value of `b` for this argument (`vopt -pa_genrpt=ud+b`) includes bitwise expanded information for isolated and level-shifted ports.

Example of UPF Dynamic Report

```
# ** Note: (vsim-8916) MSPA_UPF_RET_CTRL_INFO: Time: 15 ns, Retention  
Strategy (PD_BOT_retention), Retention SAVE (/tb/ret_bot_reg), Retention  
Sense (posedge), switched to polarity (1).      Power Domain: PD_BOT  
  
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 20 ns, Power domain  
'PD_BOT' is powered down.  
  
# ** Note: (vsim-8913) MSPA_UPF_SWITCH_CTRL_INFO: Time: 20 ns, Power  
Switch (PD_BOT_sw), Control Signal (/tb/pg_bot), switched to polarity  
(0), Power Switch state (OFF).      Power Domain: PD_BOT  
  
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 35 ns, Power domain  
'PD_BOT' is powered up.  
  
# ** Note: (vsim-8913) MSPA_UPF_SWITCH_CTRL_INFO: Time: 35 ns, Power  
Switch (PD_BOT_sw), Control Signal (/tb/pg_bot), switched to polarity  
(1), Power Switch state (ON).      Power Domain: PD_BOT  
  
# ** Note: (vsim-8916) MSPA_UPF_RET_CTRL_INFO: Time: 40 ns, Retention  
Strategy (PD_BOT_retention), Retention RESTORE (/tb/ret_bot_reg),  
Retention Sense (negedge), switched to polarity (0).      Power Domain:  
PD_BOT
```

```
# ** Note: (vsim-8914) MSPA_UPF_ISO_CTRL_INFO: Time: 65 ns, Isolation
Strategy (PD_BOT_isolation), Isolation Control (/tb/iso_bot), Isolation
Sense (HIGH), switched to polarity (1).      Power Domain: PD_BOT
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 70 ns, Power domain
'PD_BOT' is powered down.

# ** Note: (vsim-8913) MSPA_UPF_SWITCH_CTRL_INFO: Time: 70 ns, Power
Switch (PD_BOT_sw), Control Signal (/tb/pg_bot), switched to polarity
(0), Power Switch state (OFF).      Power Domain: PD_BOT
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 85 ns, Power domain
'PD_BOT' is powered up.

# ** Note: (vsim-8913) MSPA_UPF_SWITCH_CTRL_INFO: Time: 85 ns, Power
Switch (PD_BOT_sw), Control Signal (/tb/pg_bot), switched to polarity
(1), Power Switch state (ON).      Power Domain: PD_BOT

# ** Note: (vsim-8914) MSPA_UPF_ISO_CTRL_INFO: Time: 90 ns, Isolation
Strategy (PD_BOT_isolation), Isolation Control (/tb/iso_bot), Isolation
Sense (HIGH), switched to polarity (0).      Power Domain: PD_BOT
```

Architecture Report

Report Output: report.pa.txt

Command: vopt -pa_genrpt=pa

This report contains information related to Power Aware architecture that results from the power intent defined in the UPF file, as applied to the design being simulated. The content and structure of this report is same as that of report.upf.txt and includes all objects that are either added or modified by applying power intent to the design.

This report file contains information regarding the following:

- [Power Domain](#) (including supplies)
- [Power Switch](#) (including supplies)
- [Retention Strategy](#) (including supplies)
- [Isolation Strategy](#)(including supplies)
- [Level Shifter Strategy](#)
- [Power State Tables \(PSTs\)](#)

Power Domain

The power domains are listed under the “Power Domain” section of the report. This section contains the name of the power domain and the reference to the UPF file that contains the UPF command that created the power domain. All the other characteristics are listed in the following lines shifted by a tab space and starting with the property name (such as Creation Scope, Primary Supplies, Retention Strategy).

UPF Command for Power Domain

```
create_power_domain pd -include_scope
```

Report File Sections and Subheadings

```
1 Power Domain: pd, File: ./src/supply1_prefix/test.upf(7)
2   Creation Scope: /tb/top
3   Primary Supplies:
4     power   : /tb/top/pd_pwr
5     ground  : /tb/top/G_pd_net
6   Supply Set handles:
7     1. dummy_ss_handle: < Anonymous >
8     2. primary: /tb/ss
9       Functions:
10      1. power : /tb/snet_pwr
11      2. ground : < Anonymous >
12   Supply Set handles:
13     1. primary: /tb/my_second_ss, Ref Gnd: /tb/gnd_net
14       Functions:
15      1. power : /tb/vdd_net
```

Where the sections contain the following information:

- Section 1 — Defines the power domain.
- Section 2 — Specifies the full hierarchical path of the creation scope.
- Section 3 — Specifies the primary supplies of the power domain. When a power domain is not connected to a primary supply, a warning message, similar to the following, is displayed and that domain is treated as always-on:

```
** Warning: test.upf(7): (vopt-9665) Power domain: 'pd' created in
scope '/top_v1' doesn't have a primary power/ground supply. Ignoring
it.
```

- Section 4 — Specifies supply set handles associated with the power domain. This provides the names of all handles and the absolute hierarchical path of the supply set associated with them.

The supply set functions are listed below the handle name as well as the individual functions and associated supply nets.

Note



If a handle is left unassociated with a supply set or supply net, the report will display <Anonymous> in the relevant field.

- Section 5 — Shows an example of the reference ground specified in the supply set.

Power Switch

The power switches associated with the power domain are listed in this section of the report. This listing contains the name of the power switch and the reference to the UPF source file

where that switch was created. All the output and input supply ports are listed, followed by control ports and the switch states mentioned in the UPF file.

UPF Command for Power Switch

```
create_power_switch pd_sw \  
-domain pd \  
-output_supply_port { out_sw_pd pd_pwr } \  
-input_supply_port { in_sw_pd1 V_pd_net1 } \  
-input_supply_port { in_sw_pd2 V_pd_net2 } \  
-control_port { ctrl_sw_pd1 pwr1 } \  
-control_port { ctrl_sw_pd2 pwr2 } \  
-on_state { normal_working1 in_sw_pd1 { (ctrl_sw_pd1 && !ctrl_sw_pd2) } } \  
-on_state { normal_working2 in_sw_pd2 { (ctrl_sw_pd2 && !ctrl_sw_pd1) } } \  
-off_state { off_state { (!ctrl_sw_pd1 && !ctrl_sw_pd2) } }
```

Report File Sections and Subheadings

```
1 Power Switch: pd_sw, File: test.upf(35).  
2   Output Supply port:  
   out_sw_pd(/tb/top/pd_pwr)  
3   Input Supply ports:  
   1. in_sw_pd1(/tb/top/V_pd_net1)  
   2. in_sw_pd2(/tb/top/V_pd_net2)  
4   Control Ports:  
   1. ctrl_sw_pd1(/tb/top/pwr1)  
   2. ctrl_sw_pd2(/tb/top/pwr2)  
5   Switch States:  
   1. normal_working2(ON) : ( (ctrl_sw_pd2 && !ctrl_sw_pd1) )  
   2. normal_working1(ON) : ( (ctrl_sw_pd1 && !ctrl_sw_pd2) )  
   3. off_state(OFF) : ((!ctrl_sw_pd1 && !ctrl_sw_pd2))  
6   Switch Instances:  
   1. /tb/dut/SW_INST_1  
   2. /tb/dut/SW_INST_2
```

Where the sections contain the following information:

- Section 1 — Port information, appears in the format of:

```
<formal_port_name> (<externally_connected_net>)
```
- Section 2 — Displays the output supply port of the switch
- Section 3 — Displays the input supply ports.
- Section 4 — Displays the control ports.
- Section 5 — Displays the states of the switch, listing each state in the following format:

```
<state_name> (<switch state>): <Boolean_expression>
```
- Section 6 — Displays the switch instances in the design associated with the UPF switch command. This information is dependent upon specification of the `-pa_enable=detectsw` argument.

Retention Strategy

Report information for the retention strategies is listed under this heading followed by the name of the strategy and the reference to the UPF source file where the strategy was created.

UPF Commands for Retention

```
set_retention pd_retention -domain pd -retention_power_net V_pd_net

set_retention_control pd_retention -domain pd -save_signal { ret posedge }
-restore_signal { ret negedge }
```

Report File Sections and Subheadings

```
1 Retention Strategy: pd_retention, File: test.upf(39).
2   Retention Supplies:
   power  : /tb/top/V_pd_net
   ground : /tb/top/G_pd_net
3   Retention SAVE (/tb/top/ret), Retention Sense (posedge)
   Retention RESTORE (/tb/top/ret), Retention Sense (negedge)
4   Retained Signals:
   1. Scope: /tb/top_vh, File: reg_vh.vhdl(12)
     Model: Default UPF Retention
       1. /tb/top_vh/q_regvh
       2. /tb/top_vh/q_latvh
   2. Scope: /tb/top_vl, File: reg_vl.v(1)
     Model: Default UPF Retention
       1. /tb/top_vl/q_regvl
       2. /tb/top_vl/q_latvl
```

Where the sections contain the following information:

- Section 1 — Defines the retention strategy
- Section 2 — Specifies the retention supplies, listing the full hierarchical path of the supply nets.
- Section 3 — Specifies the retention control signals and the retention sense.
- Section 4 — Specifies signals that have retention applied.

Isolation Strategy

Report information for the isolation strategy is listed under this heading followed by the name of the strategy and the reference to the UPF source file where the strategy was created.

UPF Commands for Isolation

```
set_isolation pd_isolation -domain pd -isolation_power_net V_pd_net
-clamp_value 1 -applies_to outputs

set_isolation_control pd_isolation -domain pd -isolation_signal iso
-isolation_sense high -location parent
```

Report File Sections and Subheadings

```
1 Isolation Strategy: pd_isolation, File: test.upf(45).
2   Isolation Supplies:
      power : /tb/top/V_pd_net
      ground : /tb/top/G_pd_net
3   Isolation Control (/tb/top/iso), Isolation Sense (HIGH),
      Clamp Value (1), Location (parent)
4   Isolated Signals:
      1. Signal : /tb/top/q_combvl
      2. Signal : /tb/top/q_latvl
      3. Signal : /tb/top/q_regvl
```

Where the sections contain the following information:

- Section 1 — Header information about the isolation strategy.
- Section 2 — Specifies the isolation supplies with the full hierarchical path name of the supply nets.
- Section 3 — Provides additional information related to the isolation strategy, specifically:
 - Isolation Control — The control signal triggering the isolation clamp value.
 - Isolation Sense — The sense of the control signal at which the clamp value is applied on the isolated port.
 - Clamp Value — The clamp value specified.
 - Location — The location specified in the isolation strategy.
- Section 4 — Specifies a list of candidate ports for isolation with full hierarchical path names of the port.

Level Shifter Strategy

The information regarding the level shifter strategy is specified under this heading followed by the strategy name and UPF file reference.

UPF Command for Level Shifting

```
set_level_shifter pd_ls -domain pd -applies_to both
```

Report File Sections and Subheadings

```
1 Level Shifter Strategy: pd_ls, File: test.upf(51).
2   Rule (both), Threshold (0), Applies_to (both), Location (automatic).
3   Level Shifted Candidate Ports:
      1. Signal : /tb/top/q_combvl
      2. Signal : /tb/top/q_latvl
      3. Signal : /tb/top/q_regvl
      4. Signal : /tb/top/set
      5. Signal : /tb/top/reset
      6. Signal : /tb/top/clk
```

7. Signal : /tb/top/d

Where the sections contain the following information:

- Section 1 — Header information about the level-shifter strategy.
- Section 2 — Information related to the level shifting strategy, including the following:
 - Rule — Shift Direction.
 - Threshold — Threshold value in volts.
 - Applies_to — The value of the -applies_to argument specified in the set_level_shifter command.
 - Location — The location specified with the set_level_shifter command.
- Section 3 — A list of candidate ports for level shifting with full hierarchical path names of the port.

Power State Tables (PSTs)

Report information for power state tables (PST) is listed under the heading Pst.

UPF Command for Power State Table

```
add_port_state PD_ALU_sw/out_sw_PD_ALU \
-state {s_state 2.2}\
-state {r_state 3.2}\
-state {o_state 4.2}\
-state {off_state off}

add_port_state PD_RAM_sw/out_sw_PD_RAM \
-state {ram_s 3.8 4.2 4.8 }\
-state {ram_r 5.2}\
-state {ram_o 6.2}\
-state {off_state off}

create_pst MyPowerStateTable -supplies {PD_ALU_primary_power
PD_RAM_sw/out_sw_PD_RAM}
add_pst_state Reboot -pst MyPowerStateTable -state {s_state ram_s}
add_pst_state Sleep -pst MyPowerStateTable -state {r_state ram_s}
add_pst_state Hibernate -pst MyPowerStateTable -state {r_state ram_r}
add_pst_state Complete_on -pst MyPowerStateTable -state {o_state ram_o}
```

Report File Sections and Subheadings

```
1 Pst MyPowerStateTable, File:prototype.upf(80) .
2 Scope => /tb
3 Header ==>
      PD_ALU_primary_power PD_RAM_sw/out_sw_PD_RAM
      Reboot prototype.upf(84): s_state ram_s
      Sleep prototype.upf(85): r_state ram_s
      Hibernate prototype.upf(86): r_state ram_r
      Complete_on prototype.upf(87): o_state ram_o
4 List of possible states on:
      PD_ALU_primary_power [ source supply port: out_sw_PD_ALU,
```

```
File:prototype.upf(12)]
1. o_state: 4.20
2. r_state: 3.20
3. s_state: 2.20
out_sw_PD_RAM [ source supply port: out_sw_PD_RAM,
File:prototype.upf(65)]
1. ram_s      : 3.80,4.20,4.80
2. ram_r      : 5.20
3. ram_o      : 6.20
4. off_state: OFF
```

Where the sections contain the following information:

- Section 1 — Header information about the level-shifter strategy.
- Section 2 — Specifies the hierarchical path of the scope where the PST is created.
- Section 3 — Specifies hierarchical paths of the supply nets/ports that form the columns of the PST. The hierarchical paths are relative to the Scope of the PST.

The lines that follow contain the rows of the PST as mentioned in the UPF and the corresponding states of the supply nets/ports mentioned in the PST header. At the end, a list of all possible states present on the objects mentioned in the header is shown. This includes detailed information of the states listed with the voltage information.

- Section 4 — Specifies a list of all possible states present on the objects mentioned in the header. The detailed information of the states is listed with the voltage information as well. This information is similar to what was specified in the corresponding `add_port_state` UPF command. The source supply port mentioned in brackets ([]) is the name of the supply port on which the `add_port_state` command is called or its the supply net/port which is directly connected to the port on which `add_port_state` command was called. The UPF file reference corresponds the file and line number of the `add_port_state` command.

Sample Power Architecture Report

```
-----
---- QuestaSim Power Aware Architecture Report File ----
-----
-- QuestaSim Version:      DEV-main 2294369 2011.01
-- Generated on           :      Tue Jan 25 09:37:28 2011
-----
-- This report file contains information about all
-- the Power Aware Architecture elements in the design.
-----

Power Domain: pd_aon, File: test.upf(4).
Creation Scope: /tb
Primary Supplies:
  power : /tb/vdd_net
  ground : /tb/gnd_net
-----

Power Domain: pd, File: test.upf(16).
Creation Scope: /tb
```

```
Primary Supplies:
  power : /tb/pd_pwr
  ground : /tb/G_pd_net
Power Switch: pd_sw, File: test.upf(37).
  Output Supply port:
    out_sw_pd(/tb/pd_pwr)
  Input Supply ports:
    1. in_sw_pd(/tb/V_pd_net)
  Control Ports:
    1. ctrl_sw_pd(/tb/pwr)
  Switch States:
    1. normal_working(ON) : ( ctrl_sw_pd )
    2. off_state(OFF) : (!ctrl_sw_pd)
Retention Strategy: pd_retention, File: test.upf(41).
  Retention Supplies:
    power : /tb/V_pd_net
    ground : /tb/G_pd_net
  Retention SAVE (/tb/ret), Retention Sense (posedge)
  Retention RESTORE (/tb/ret), Retention Sense (negedge)
  Retained Signals:
    1. Scope: /tb/top_vh, File: reg_vh.vhdl(12)
      Model: upf_retention_ret
        1. /tb/top_vh/q_regvh
    2. Scope: /tb/top_vl, File: reg_vl.v(1)
      Model: upf_retention_ret
        1. /tb/top_vl/q_regvl

Isolation Strategy: pd_isolation, File: test.upf(48).
  Isolation Supplies:
    power : /tb/V_pd_net
    ground : /tb/G_pd_net
  Isolation Control (/tb/iso), Isolation Sense (HIGH), Clamp Value (1),
  Location (parent)
  Isolated Signals:
    1. Signal : /tb/top_vh/q_combvh
    2. Signal : /tb/top_vh/q_latvh
    3. Signal : /tb/top_vh/q_regvh
    4. Signal : /tb/top_vl/q_combvl
    5. Signal : /tb/top_vl/q_latvl
    6. Signal : /tb/top_vl/q_regvl
```

Design Elements Report

Report Output: report.de.txt

Command: vopt -pa_genrpt=de

This report contains all the information related to elements present in user design and its corresponding Power Aware information. The contents of this report are similar to that provided in the *report.mspa.txt* file, but it is presented in a different format that makes it easier to extract information from the report. Report information is listed in single-line records that contain keywords indicating the specific Power Aware characteristics.

This report file contains information regarding the following:

- [Design Element Scopes and Power Domains](#)
- [Corrupted Signals](#)
- [State Elements](#)
- [Retention Signals](#)

Design Element Scopes and Power Domains

This information identifies the extent of power domains, such that all scopes in the design belonging to a particular power domain are listed in this report.

Format

The report format for listing domains, scopes, and elements is the following:

```
<power_domain>: {Path<N>} = scope <element> [SIM] [ <> ]
```

where:

- <power_domain> — identifies the name of the power domain. When <power_domain> is represented by two dashes (--) it indicates that the <element> was excluded using exclusion rules.
- Path<N> — identifies the hierarchical path of the scope in the rest of the report.
- scope — keyword that identifies that this line is a design element scope (to assist in using a grep search).
- <element> — identifies the full hierarchical path of the design element (its scope).
- SIM — (optional) keyword that identifies a power domain referring to simulation-only elements (such as processes, always blocks, or signals).
- <> — (optional) identifies a scope that is present at the power domain boundary.

Example: Excerpt of UPF Commands

```
set_scope tb  
create_power_domain pd_aon -include_scope  
create_power_domain pd -elements {top_vl top_vh}
```

Example: Design Elements Report Excerpt for Scopes and Power Domains

```
pd_aon: {Path1} = scope /tb/top_aon  
pd: {Path2} = scope /tb/top_vh <>  
pd: {Path3} = scope /tb/top_vl <>
```

Corrupted Signals

The signals inside the power domain corrupted when the supply of the power domain is switched off.

Format

The report format for listing corrupted signals is the following:

```
<power_domain>: {Path<N>}/<signal>
```

where

- `<power_domain>` — identifies the name of the power domain. When `<power_domain>` is represented by two dashes (--) it indicates that the `<signal>` was excluded using exclusion rules.
- `Path<N>` — identifies the hierarchical path of the scope in the rest of the report.
- `<signal>` — identifies the name of the signal.

Example: Design Elements Report Excerpt for Corrupted Signals

```
pd_aon: {Path1}/q_combv1  
pd: {Path2}/q_latvh NPM_LA  
pd: {Path3}/q_regv1 R
```

State Elements

The signals in the design acting as state elements are listed in the same way as [Corrupted Signals](#), along with a special keyword that identifies the kind of state element.

Format

The report format for listing corrupted signals is the following:

```
<power_domain>: {Path<N>}/<signal> <keyword>
```

where

- `<power_domain>` — identifies the name of the power domain. When `<power_domain>` is represented by two dashes (--) it indicates that the `<signal>` was excluded using exclusion rules.
- `Path<N>` — identifies the hierarchical path of the scope in the rest of the report.
- `<signal>` — identifies the name of the signal.
- `<keyword>` — one of the following values, indicating whether the signal is acting as a state element or a retention element:
 - `NPM_LA` — Non-retention Latch

- NPM_FF — Non-retention Flip-Flop
- MEM — Non-retention Memory
- UDP_LA — Non-retention UDP Latch
- UDP_FF — Non-retention UDP Flip-Flop
- R — Retention element
- <no keyword> — Combinatorial logic

Example: Design Elements Report Excerpt for State Elements

```
pd: {Path3}/q_combv1
pd: {Path3}/q_regv1 R
pd: {Path3}/q_latv1 NPM_LA
pd_aon: {Path1}/q_regv1 NPM_FF
```

Retention Signals

The signals in the design performing retention behavior are marked with a special keyword R after the signal name.

The signals in the design performing retention behavior are listed in the same way as [Corrupted Signals](#), along with the keyword R after the signal name.

Format

The report format for listing corrupted signals is the following:

```
<power_domain>: {Path<N>}/<signal> R
```

where

- <power_domain> — identifies the name of the power domain. When <power_domain> is represented by two dashes (--) it indicates that the <signal> was excluded using exclusion rules.
- Path<N> — identifies the hierarchical path of the scope in the rest of the report.
- <signal> — identifies the name of the signal.
- R — indicates a retention signal.

Example: Excerpt of UPF Commands

```
set_retention pd_retention -domain pd -retention_power_net V_pd_net

set_retention_control pd_retention -domain pd -save_signal { ret posedge }
-restore_signal { ret negedge}
```

Example: Design Elements Report Excerpt for Retention Signals

```
pd: {Path3}/q_regv1 R
pd: {Path2}/q_regvh R
```

Working With A Design Element Report

General Information

The name of the report and other information are presented as commented text at the top of the report. Comments are marked with double hyphens (--) at the beginning of a line. For example:

```
-----
----- Questa Sim Power Aware Design Element Report File -----
-----
-- QuestaSim Version: DEV-10.0 2238002 2010.11
-- Generated on : Mon Nov 22 13:44:03 2010
-----
```

Extracting Information from a Design Element Report

The structure and keywords of a Power Aware design element report allow you to extract the information related to power intent easily and effectively using any string matching utility, such as the `grep` command.

The following examples show how to use the `grep` command to extract specific details of your power intent.

List all instances in the design belonging to a power domain

Function: Search all scopes belonging to power domain `pd_aon`.

Command:

```
grep pd_aon report.de.txt | grep scope
```

Output:

```
pd_aon: {Path1} = scope /tb/top_aon
```

List all retention signals in the design

Command:

```
grep " R" report.de.txt | grep -v "^--"
```

Output:

```
pd: {Path2}/q_regvh R
pd: {Path3}/q_regv1 R
```

Expand the Path Id

Command:

```
grep "{Path2} =" report.de.txt
```

Output:

```
pd: {Path2} = scope /tb/top_vh <>
```

Search a particular signal for power intent

Command 1:

```
grep "/tb/top_vh\>" report.de.txt
```

Output 1:

```
pd: {Path2} = scope /tb/top_vh <>
```

Command 2:

```
grep "{Path2}/q_regvh" report.de.txt
```

Output 2:

```
pd: {Path2}/q_regvh R
```

(implies that /tb/top_vh/q_regvh is a retention register inside power domain pd)

Identify instances at power domain boundary

Command:

```
grep "<>" report.de.txt
```

Output:

```
pd: {Path2} = scope /tb/top_vh <>  
pd: {Path3} = scope /tb/top_vl <>
```

Sample Power Aware Design Element Report

```
-----  
----- QuestaSim Power Aware Design Element Report File -----  
-----  
-- QuestaSim Version:  
-- Generated on      :  
-----  
-- This report file contains PA information about all the elements in user  
-- design.  
  
pd_aon: {Path1} = scope /tb  
pd_aon: {Path2} = scope /tb/top_aon  
pd: {Path3} = scope /tb/top_vh <>  
--: {Path4} = scope /tb/top_vl <>  
  
pd_aon: {Path1}/tie_high_low  
pd_aon: {Path1}/q_combaon  
pd_aon: {Path1}/q_latchaon  
pd_aon: {Path1}/q_regaon
```

```
pd_aon: {Path1}/q_combvh
pd_aon: {Path1}/q_latchvh
pd_aon: {Path1}/q_regvh
pd_aon: {Path1}/q_combvl
pd_aon: {Path1}/q_latchvl
pd_aon: {Path1}/q_regvl
pd_aon: {Path1}/ret
pd_aon: {Path1}/iso
pd_aon: {Path1}/pwr
pd_aon: {Path1}/set
pd_aon: {Path1}/rst
pd_aon: {Path1}/clk
pd_aon: {Path1}/d
pd_aon: {Path2}/q_combvl
pd_aon: {Path2}/q_regvl NPM_FF
pd_aon: {Path2}/q_latvl NPM_LA
pd: {Path3}/q_combvh
pd: {Path3}/q_regvh R
pd: {Path3}/q_latvh NPM_LA
--: {Path4}/q_combvl
--: {Path4}/q_regvl R
--: {Path4}/q_latvl NPM_LA
```

Behavioral Element Reporting

Use the `-pa_behavlogfile` argument to generate a report on behavioral constructs in the design that need to be in an always-on power domain.

Usage

```
vopt -pa_behavlogfile=<filename>
```

Description

This argument creates a report, *filename*, of all the non-synthesizable constructs found in design. This report allows you to identify all the constructs that have to be put in an always-on power domain.

The format for the constructs of this argument is the following:

- I: hierarchical path of instance (for non-synthesizable instances)
- P: hierarchical path of block process/always/initial (for non-synthesizable process/always/initial blocks)
- S: hierarchical path of signals (for signals (writers) falling in non-synthesizable unnamed process/always/initial block)

Example Excerpt from a Behavioral Log File

```
S : /top/out
P : /top/l1
P : /top/l2
```

S : /top/out
I : /top/bot

Chapter 5

Automatic Checking

This chapter describes how to perform static and dynamic checking as part of Power Aware simulation. To implement either mode of checking, use the **-pa_checks** argument of the **vopt** command. This argument takes on a variety of values, which are listed and described in the following sections:

- [Static Checking in Power Aware](#)
- [Dynamic Checking in Power Aware](#)
- [Implementing Checking at Gate Level](#)

Note



You can enable all power aware checks (static and dynamic) by specifying the **-pa_checks=all** argument.
You can enable all dynamic power aware checks by specifying the **-pa_checks=d** argument or **-pa_checks** without any argument.
You can enable all static power aware checks by specifying the **-pa_checks=s** argument.

Static and Dynamic Checking Overview

This section provides an overview of static and dynamic checking for level shifters and isolation.

Level Shifter Checking

A level shifter cell is present in the design description if a **set_level_shifter -instance** command in your UPF file identifies that cell in the design description as a level shifter cell.

A level shifter cell is implied by a level shifting strategy if the **set_level_shifter** command exists with the appropriate options for the appropriate power domain.

- Level shifting is **statically** required for a power domain crossing if the source and sink domains *can be* both powered on at the same time and the difference between the maximum voltages exceeds a certain threshold.
- Level shifting is **dynamically** required for a power domain crossing if the source and sink domains *are* both powered on at the same time and the difference between the maximum voltages exceeds a certain threshold.

The direction of a level shifter must be:

- **high_to_low** — if the maximum voltage powering the source domain is *higher* than the maximum voltage powering the sink domain.
- **low_to_high** — if the maximum voltage powering the source domain is *lower* than the maximum voltage powering the sink domain.

Isolation Checking

An isolation cell is present in the design description if a **set_isolation -instance** command in your UPF file identifies that cell in the design description as an isolation cell.

An isolation cell is implied by an isolation strategy if a **set_isolation** command exists with the appropriate options for the appropriate power domain.

- Isolation is **statically** required for a power domain crossing if the source domain *can be* off when the sink domain is on.
- Isolation is **dynamically** required for a power domain crossing if the source domain *is* off when the sink domain is on.

Static Checking in Power Aware

The **-pa_checks** argument enables static checking to validate the behavior of power intent and specification in the designs. The values you specify for the **-pa_checks** argument activate static checking of signals for the power conditions: [Static Isolation Checks](#) and [Static Level Shifter Checks](#).

The static checking functionality does the following:

- Displays a vopt summary related to all the checks performed and the result.
- Generates a detailed report file (*report.static.txt*) containing the result of static checks in the following formats:
 - Domain-wise dumping — all the ports are identified and collected with respect to connectivity between power domains.
 - Level-shifter-strategy-wise dumping — the same information displayed with respect to individual level shifting strategy. It helps to correlate easily with the list of candidate ports dumped in the UPF report file, *report.upf.txt*.

For all static (isolation or level-shifter) checks, Power Aware simulation analyzes any specified [Power State Tables](#) (PSTs) and power states added on power domains and supply sets in the UPF file. The purpose of this is to detect the power domain relative OFF/ON condition and relative operating voltages.

If state dependencies between two connected power domains are not present in PST/add_power_state, then Power Aware simulation cannot determine power domain relative

ON/OFF states statically. In this case, static isolation checking will report isolation strategies as “Not Analyzed.”

Table 5-7 shows a quick-reference comparison of static and dynamic checks.

Usage Notes for Static Checking

The **-pa_checks** argument of the **vopt** command has numerous possible values that you can assign to enable either dynamic or static checking for Power Aware simulation.

- For a quick listing of all values, refer to [Table A-1](#).
- For additional reference information, refer to the **vopt** command in the ModelSim Reference Manual.
- For a listing of static and dynamic checks according to elements defined for a given power domain, refer to [Table 5-7](#).
- To specify more than one value, use the plus (+) operator between values, for example:

```
vopt -pa_checks=sni+sdi
```

Related Topics

[Voltage Level-Shifting \(Multi-Voltage Analysis\)](#)

[Modeling Isolation](#)

Debugging Static Checks

You can perform debugging on static checking for isolation or level shifters by using the **vopt -pa_dbgstatic** command. The available values determine the debugging behavior performed, as described below.

- **-pa_dbgstatic=rsn**

Power Aware simulation appends information at the end of a report following the phrase “Possible reason” to help determine the cause of the check. Report file example (*report.static.txt*):

```
Inferred type: 'Not inserted' [ count: 1 ],  
Source port : /tb/TOP/bot4/bot5/out1_bot [ connected mask: "1" ]  
[ LowConn ] { Domain: pd_bot } -> Sink port: /tb/TOP/bot4/out1_bot  
[ connected mask: "0001" ] [ HighConn ] { Domain: pd_top }  
Possible reason: 'Level shifter is specified as -no_shift'
```

- **-pa_dbgstatic=msk**

Captures the connection mask of the signal taken in a particular path. For example, if there is a vector taking a different path, the phrase “connected mask” provides

information about which bits are used in connecting the particular source and sink.
 Report File example (*report.static.txt*):

```
Inferred type: 'Incorrect' [ count: 0 ],
Source port : /tb/TOP/bot4/out1_bot [ connected mask: "0100" ]
[ LowConn ] { Domain: pd_aon } -> Sink port: /tb/TOP/bot4/out1_bot
[ connected mask: "0100" ] [ HighConn ] { Domain: pd_top }
```

Static Isolation Checks

Table 5-1 lists a summary of the various static isolation checks you can apply by specifying different values for the **vopt -pa_checks** command. All static isolation check results are written to the report file, *report.static.txt*.

Table 5-1. Static Isolation Checks

Check	Usage Syntax	Description Example Message
Static Missing	vopt -pa_checks=smi	<p>For domain crossings where the source domain is relatively OFF with respect to sink domain.</p> <p>Reports missing isolation cells for any isolation strategies not specified for such crossings.</p> <p>Warning: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 1 Missing isolation cells.</p>
Static Redundant	vopt -pa_checks=sri	<p>If power domain crossings occur where isolation strategies are specified, but isolation is not required for power domain crossing, then these strategies will be reported as redundant.</p> <p>Reports redundant isolation cells when isolation is not required from driving power domain to sink power domain and isolation cell is placed for the power domain crossing.</p> <p>Warning: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 3 Redundant isolation cells.</p>
Static Incorrect	vopt -pa_checks=sii	<p>Reports incorrect isolation cells when isolation is required for power domain crossing but isolation strategy is specified with set_isolation -no_isolation.</p> <p>Warning: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 3 Incorrect isolation cells.</p>

Table 5-1. Static Isolation Checks (cont.)

Check	Usage Syntax	Description Example Message
Static Valid	vopt -pa_checks=svi	Isolation is required for a power domain crossing and valid isolation strategies are specified for domain crossing. Note: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 1 Valid isolation cells.
Static Not Analyzed	vopt -pa_checks=sni	Reports isolation strategies as “Not analyzed” if power state table (PST) information is not sufficient to analyze whether isolation is required or not for input and output power domains of an isolation strategy. Warning: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 1 Not analyzed isolation cells.
Static Not Inserted	vopt -pa_checks=sdi	Reports isolation strategies as “Not inserted” when isolation is not required for a power domain crossing and isolation strategies specified with set_isolation -no_isolation . Warning: (vopt-9750) [UPF_ISO_STATIC_CHK] Found Total 1 Not inserted isolation cells.
All Static Isolation Checks	vopt -pa_checks=si	Perform all static isolation checks (smi, sri, sii, svi, sni, sdi)

Isolation Cell Instance Checking

When you use the [set_isolation](#) -instance command in a UPF file to instantiate RTL isolation cells, Power Aware simulation detects those cells and performs the following isolation checks on them, reporting the information in the file *report.static.txt*:

- **Incorrect Isolation Check**

```
Total 1 Incorrect isolation cells.
  Source power domain : PD_mid1 -> Sink power domain: PD_mid2
[ Total count: 1 ]
  1. ISO( count: 1 ): Candidate Port: /tb/TOP/mid1/out1_bot
[ connected mask: "0001" ], count:1, Isolation cell :
/tb/TOP/isoinst1_0(ISO_AND), Domain: PD_mid1,
  Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0001" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot
[ connected mask: "0001" ], count:1 [ LowConn ]
```

- **Valid Isolation**

```
PD_mid2 [ Total count: 1 ]
```

```

1. ISO( count: 1 ): Candidate Port: /tb/TOP/mid1/out1_bot
[ connected mask: "0010" ], count:1, Isolation cell :
/tb/TOP/isoinst1_1(ISO_AND), Domain: PD_mid1,
    Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0010" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot
[ connected mask: "0010" ], count:1 [ LowConn ]

```

- **Redundant Isolation**

```

Total 1 Redundant isolation cells.
    Source power domain : PD_mid1 -> Sink power domain: PD_mid2
[ Total count: 1 ]
    1. ISO( count: 1 ): Candidate Port: /tb/TOP/mid2/in1_bot
[ connected mask: "0100" ], count:1, Isolation cell :
/tb/TOP/isoinst2_0(ISO_AND), Domain: PD_mid2,
    Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0100" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot [
connected mask: "0100" ], count:1 [ LowConn ]

```

- **Not Analyzed Isolation**

```

Source power domain : PD_mid1 -> Sink power domain: PD_mid2
[ Total count: 1 ]
    1. ISO( count: 1 ): Candidate Port: /tb/TOP/mid2/in1_bot
[ connected mask: "1000" ], count:1, Isolation cell :
/tb/TOP/isoinst2_1(ISO_AND), Domain: PD_mid2,
    Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"1000" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot
[ connected mask: "1000" ], count:1 [ LowConn ]

```

Static Level Shifter Checks

You can apply various static checking functions for level shifters by using the **vopt -pa_checks** command. The value you specify determines the checking function performed, as shown in [Table 5-2](#).

Table 5-2. Static Level Shifter Checks

Check	Usage Syntax	Description Example Message
Missing Level Shifter	vopt -pa_checks=sml	Checks that if level shifter is required at domain crossing and level shifter strategy is not specified. ** Warning: (vopt-9693) [UPF_LS_STATIC_CHK] Found Total 1 Missing level shifters.

Table 5-2. Static Level Shifter Checks (cont.)

Check	Usage Syntax	Description Example Message
Incorrect Level Shifter	vopt -pa_checks=sil	Checks that if the direction of level shifters specified for domain crossing does not match with the direction as per voltage difference of domain crossing. For example: domain crossing requires low_to_high level shifter and level shifter specified is high_to_low, then it will be reported as incorrect. ** Warning: (vopt-9693) [UPF_LS_STATIC_CHK] Found Total 1 Incorrect level shifters.
Redundant Level Shifter	vopt -pa_checks=srl	Checks for any level shifter whose source and sink power domains have no voltage difference, flagging this as a redundant level shifter. ** Warning: (vopt-9693) [UPF_LS_STATIC_CHK] Found Total 1 Redundant level shifters.
Unanalyzed Level Shifter	vopt -pa_checks=snl	Reports level shifter strategies that are not analyzed because of insufficient information, such as when a power state table is not specified or is incomplete. ** Warning: (vopt-9693) [UPF_LS_STATIC_CHK] Found Total 1 Not analyzed level shifters.
Valid Level Shifter	vopt -pa_checks=svl	Indicates the valid level shifters that have matching voltage information and proper direction of shift. (Also, see Reporting for a Valid Level Shifter.) ** Note: (vopt-9693) [UPF_LS_STATIC_CHK] Found Total 2 Valid level shifters
Uninserted Level Shifter	vopt -pa_checks=sdl	Reports level shifter strategies that are defined with the set_level_shifter -no_shift command in UPF file, but are not inserted. Inferred type: 'Not inserted'[count: 1], Source port : /tb/TOP/bot4/bot5/out1_bot [LowConn] { Domain: pd_bot } -> Sink port: /tb/TOP/bot4/out1_bot [HighConn] { Domain: pd_top }
All Static Level Shifter Checks	vopt -pa_checks=sl	Enables all static level shifter checks. Writes Static Checks report to <i>report.static.txt</i> file.
All Static Checks	vopt -pa_checks=s	Enables all static level shifter and static isolation checks.

Level Shifter Cell Instance Checking

If you have used the `set_level_shifter` -instance command in a UPF file to instantiate RTL level shifter cells, Power Aware simulation will detect those cells and perform the following level shifting checks on them (instance checking results are written to report.static.txt):

- **Incorrect Level Shifter Check**

```
Total 1 Incorrect level shifters
  Source power domain : PD_mid1 -> Sink power domain: PD_mid2 [
Total count: 1 ]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/mid1/out1_bot [
connected mask: "0001" ], count:1, level shifter cell :
/tb/TOP/lsinst1_0(ls_buf), Domain: PD_mid1,
  Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0001" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot [
connected mask: "0001" ], count:1 [ LowConn ]
```

- **Valid Level Shifter**

```
Total 1 Valid level shifters
  Source power domain : PD_mid1 -> Sink power domain: PD_mid2 [
Total count: 1 ]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/mid1/out1_bot [
connected mask: "0010" ], count:1, : level shifter cell :
/tb/TOP/lsinst1_1(ls_buf), Domain: PD_mid1,
  Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0010" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot [
connected mask: "0010" ], count:1 [ LowConn ]
```

- **Redundant Level Shifter**

```
Total 1 Redundant level shifter.
  Source power domain : PD_mid1 -> Sink power domain: PD_mid2 [
Total count: 1 ]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/mid2/in1_bot [
connected mask: "0100" ], count:1, level shifter cell :
/tb/TOP/lsinst2_0(ls_buf), Domain: PD_mid2,
  Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"0100" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot [
connected mask: "0100" ], count:1 [ LowConn ]
```

- **Not Analyzed Level Shifter**

```
Total 1 Not Analyzed level shifters.
  Source power domain : PD_mid1 -> Sink power domain: PD_mid2 [
Total count: 1 ]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/mid2/in1_bot [
connected mask: "1000" ], count:1, level shifter cell :
/tb/TOP/lsinst2_1(ls_buf), Domain: PD_mid2,
  Source port : /tb/TOP/mid1/out1_bot [ connected mask:
"1000" ], count:1 [ LowConn ] -> Sink port: /tb/TOP/mid2/in1_bot [
connected mask: "1000" ], count:1 [ LowConn ]
```

Reporting for a Valid Level Shifter

- -pa_checks=svl

Indicates the valid level shifters that have matching voltage information and proper direction of shift. Message from vopt:

```
** Note: (vopt-9693) [ UPF_LS_STATIC_CHK ] Found Total 2 Valid level shifters
```

Report File dump (for the power domain): (report.static.txt)

```
Total 2 Valid level shifters.
Source power domain: pd_bot -> Sink power domain: pd_aon
[Total count: 1]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/bot1/out1_bot,
count:1, level shifting strategy : my_ls [PD: pd_bot ],
Source port : /tb/TOP/bot1/out1_bot, count:1 [ LowConn ] ->
Sink port: /tb/TOP/bot2/in2_bot, count:1 [ LowConn ]

  2. LS( count: 0 ): Candidate Port: /tb/TOP/bot1/out1_bot,
count:1, level shifting strategy : my_ls [PD: pd_bot ],
Source port : /tb/TOP/bot1/out1_bot, count:1 [ LowConn ] ->
Sink port: /tb/TOP/bot3/in2_bot, count:1 [ LowConn ]

  3. LS( count: 0 ): Candidate Port: /tb/TOP/bot1/out1_bot,
count:1, level shifting strategy : my_ls [PD: pd_bot ],
Source port : /tb/TOP/bot1/out1_bot, count:1 [ LowConn ] ->
Sink port: /tb/TOP/bot4/in2_bot, count:1 [ LowConn ]

Source power domain: pd_aon -> Sink power domain: pd_top
[Total count: 1]
  1. LS( count: 1 ): Candidate Port: /tb/TOP/bot2/out1_bot,
count:1, level shifting strategy : my_ls [PD: pd_aon ],
Source port : /tb/TOP/bot2/out1_bot, count:1 [ LowConn ] ->
Sink port: /tb/TOP/bot2/out1_bot, count:1 [ HighConn ]
```

Report File dump (for the level shifter strategy):

```
Total 2 Valid level shifters
Level shifting strategy: my_ls, Power domain: pd_aon
  1. Candidate port: /tb/TOP/bot2/out1_bot [ count: 1 ]
Inferred type:'Valid'[ count: 1 ], Source port :
/tb/TOP/bot2/out1_bot [ LowConn ] { Domain: pd_aon } ->
Sink port: /tb/TO
P/bot2/out1_bot [ HighConn ] { Domain: pd_top }

Level shifting strategy: my_ls, Power domain: pd_bot
  3. Candidate port: /tb/TOP/bot1/out1_bot [ count: 1 ]
Inferred type:'Valid'[ count: 0 ], Source port :
/tb/TOP/bot1/out1_bot [ LowConn ] { Domain: pd_bot } ->
Sink port: /tb/TO
P/bot4/in2_bot [ LowConn ] { Domain: pd_aon }
Inferred type:'Valid'[ count: 0 ], Source port :
/tb/TOP/bot1/out1_bot [ LowConn ] { Domain: pd_bot } ->
Sink port: /tb/TO
P/bot3/in2_bot [ LowConn ] { Domain: pd_aon }
```

```
Inferred type: 'Valid' [ count: 1 ], Source port :  
  /tb/TOP/bot1/out1_bot [ LowConn ] { Domain: pd_bot } ->  
  Sink port: /tb/TO  
  P/bot2/in2_bot [ LowConn ] { Domain: pd_aon }
```

Note



Sometimes, you may find the level shifter count for a particular source-to-sink path may be 0. This happens when there is only one level shifter identified for insertion at that boundary (even though there are multiple fanouts possible), but the value for `set_level_shifter -location` is not specified as fanout. In that case, the level shifter insertion is counted for only one of the paths, while the other paths will have level shifter counts reported as 0.

Dynamic Checking in Power Aware

Use the `-pa_checks` argument of the `vopt` command to perform more comprehensive dynamic rule checking to validate the behavior of power intent and specification in the designs. The values you specify activate dynamic checking of signals for various power conditions:

- [Dynamic Retention Checking](#)
- [Dynamic Isolation Checking](#)
- [Dynamic Level Shifter Checking](#)
- [Miscellaneous Dynamic Checking](#)
 - Toggle
 - Control Signal Corruption Checking
 - Always-On Power Domain Checking
 - Power Domain Status

[Table 5-7](#) shows a quick-reference comparison of static and dynamic checks.

Usage Notes

The `-pa_checks` argument of the `vopt` command has numerous possible values that you can assign to enable either dynamic or static checking for Power Aware simulation.

- For a list of all values, refer to [Table A-1](#).
- For additional reference information, refer to the `vopt` command in the Reference Manual.
- For a list of static and dynamic checks according to elements defined for a given power domain, refer to [Table 5-7](#).

- To specify more than one checking, use the plus (+) operator between values. For example:

```
vopt -pa_checks=rop+cp+a
```

- To enable all dynamic checks, specify the -pa_checks=d argument.

Dynamic Retention Checking

You can specify any of the values in [Table 5-3](#) to provide retention checking:

Table 5-3. Dynamic Retention Checks

Check	Usage Syntax	Description Example Message
Power off	vopt -pa_checks=rop	<p>Reports that there were retention signals mapped to the power domain. However, these signals were not asserted when the power was switched off.</p> <p>Error: (vsim-8903) MSPA_RET_OFF_PSO: Time: 35 ns, Retention control (x) for the following retention elements of power domain 'PD' is not asserted during power shut down: #/tb/top/mid1/bot_latch1. # File: ./src/pa_all_checks/, Line: 2, Power Domain:PD</p>
Power on	vopt -pa_checks=rpo	<p>Reports that there was an error in the sequence of triggering of the retention and power signal. For retention to succeed, the power should be high. However, this check is triggered when that is not the case.</p> <p>Error: (vsim-8904) MSPA_RET_PD_OFF: Time: 85 ns, Power for domain: 'PD' is not ON (0) when retention is enabled for retention elements: # /tb/top/mid1/bot_latch1. # File: ./src/pa_all_checks/, Line: 2, Power Domain:PD</p>

Table 5-3. Dynamic Retention Checks (cont.)

Check	Usage Syntax	Description Example Message
Clock/Latch enable	vopt -pa_checks=rcs	<p>Certain Power Aware models require that the clock/latch enable must be at a certain value when retention takes place.</p> <p>Reports an error when this condition is not satisfied. If latch is enabled and can change its value, triggering retention can potentially cause race conditions in the stored value. This is also a check against such conditions.</p> <p>Error: (vsim-8905) MSPA_RET_CLK_STATE: Time: 85 ns, LatchEn is not at proper level: 'LOW' (1) for the retention element(s) of type: AHRLA of power domain: PD. # /tb/top/mid1/bot_latch1. # File: Line: 2, Power Domain:PD</p> <p>Error: (vsim-8905) MSPA_RET_CLK_STATE: Time: 207 ns, Clock is not at proper level: 'LOW' (1) for the retention element(s) of type: CLRFF of power domain: PD. # /tb/top/mid1/bot_ff. # File: Line: 3, Power Domain:PD</p>
Clock toggle	vopt -pa_checks=rsa	<p>Some Power Aware models require that the clock not toggle when the power is down. This check helps in monitoring this condition.</p> <p>Error: (vsim-PA-8906) MSPA_RET_SEQ_ACT: Time: 208 ns, clock toggled during retention period for retention element(s): # /tb/top/mid1/bot_ff</p>
All	vopt -pa_checks=r	

Dynamic Isolation Checking

Isolation checks trigger messages when a particular isolation strategy has failed or a noise has been detected in isolating a particular hierarchy.

Table 5-4. Dynamic Isolation Checks

Check	Usage Syntax	Description Example Message
Isolation Clamp Value Check	vopt -pa_checks=icp	Ensures isolation cell is clamping to correct clamp value specified in UPF file. Catches any functional issue in isolation cell inserted in RTL or applied isolation. Performed during active isolation period. Error: (vsim-8930) MSPA_ISO_CLAMP_CHK: Time: 40 ns, Isolated port for isolation cell (strategy: iso_PD_mid2_2) on port '/tb/TOP/mid2/in2_bot' having value (x) is different from clamp value (1) during isolation period. # File: test.upf, Line: 125, Power Domain:PD_mid2
Isolation Disable Protocol Check	vopt -pa_checks=idp	Flags violation if isolation control of isolation cell/strategy is disabled during power shut OFF of source power domain (power domain of driving ports) and sink power domain ON. Error: (vsim-8919) MSPA_ISO_DIS_PG: Time: 250 ns, Isolation control is disabled during power shut OFF (0) for the following: # Port: /tb/TOP/mid3/in2_bot[3:2]. # File: test.upf, Line: 113, Power Domain:PD_mid3
Isolation Enable Protocol Check	vopt -pa_checks=iep	Flags violation if isolation control of isolation cell/strategy is not enabled when source power domain (power domain of driving ports) is switched OFF and sink power domain ON. Error: (vsim-8918) MSPA_ISO_EN_PSO: Time: 358 ns, Isolation control (0) is not enabled when power is switched OFF for the following: # Port: /tb_25/FA4_inst/ FA_inst2/d.

Table 5-4. Dynamic Isolation Checks (cont.)

Check	Usage Syntax	Description Example Message
Isolation Disable Protocol Check for COA States	vopt - pa_checks=idpcoa	<p>Flags violation if isolation control is disabled when the driving power domain is in the following CORRUPT simstates, and receiving logic does not have a CORRUPT simstate: CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_STATE_ON_CHANGE, and CORRUPT_STATE_ON_ACTIVITY.</p> <p>Error: (vsim-8919) MSPA_ISO_DIS_PG: Time: 60 ns, Isolation control is disabled during driving supply CORRUPT_ON_ACTIVITY simstate (1) for the following: # Port: /tb/TOP/mid1/out1_bot, # File: test.upf, Line: 62, Power Domain:PD_mid1</p>
Isolation Enable Protocol Check for COA States	vopt - pa_checks=iepcoa	<p>Flags violation if isolation control is not enabled when a driving power domain goes into any of the following states and receiving logic does not have CORRUPT simstate: CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_STATE_ON_CHANGE, and CORRUPT_STATE_ON_ACTIVITY.</p> <p>Error: (vsim-8918) MSPA_ISO_EN_PSO: Time: 20 ns, Isolation control (x) is not enabled when driving supply is in CORRUPT_ON_ACTIVITY simstate for the following: # Port: /tb/TOP/mid1/out1_bot, # File: test.upf, Line: 62, Power Domain:PD_mid1</p>
Isolation Functionality Check	vopt -pa_checks=ifc	<p>Ensures that when isolation is not applied, the value at isolation cell's output is same as that at its input. This is to catch any functional error because of applied isolation cell. Performed during in-active isolation period. During inactive isolation period, it will catch occurrences where value of isolated port is different than the ports' value.</p> <p>Error: (vsim-8931) MSPA_ISO_FUNC_CHK: Time: 100 ns, Isolated port for isolation cell (strategy: iso_PD_mid2) on port '/tb/TOP/mid2/in2_bot' having value (x) is different from port value (1) during non-isolation period. # File: test.upf, Line: 111, Power Domain:PD_mid2</p>

Table 5-4. Dynamic Isolation Checks (cont.)

Check	Usage Syntax	Description Example Message
Isolation Redundant Activity Check	vopt -pa_checks=ira	<p>Flags violation if there was no requirement of isolation (such as driving logic is not in an OFF or CORRUPT state when receiving logic is in an ON state) and there is activation of an isolation control signal. In general, this ensures that if activation of isolation control occurs that it is required by power management behavior.</p> <p>Error: (vsim-8934) MSPA_ISO_REDUNDANT_ACT: Time: 430 ns, Redundant activity on isolation control signal, for crossing {(PD_mid1) => (PD_mid2)}. # Port: /tb/TOP/mid1/out1_bot, # File: test.upf, Line: 44, Power Domain:PD_mid1</p>
Isolation Race Check	vopt -pa_checks=irc	<p>The value on isolated ports should not change when isolation is enabled (at the time of assertion of isolation control) and when isolation is disabled (at the de-assertion of isolation control). This check flags any toggling of isolated port's value at assertion/de-assertion of isolation control signal.</p> <p>Error: (vsim-8910) MSPA_ISO_PORT_TOGGLE: Time: 50 ns, Isolated port for isolation cell (strategy: iso_PD_mid1) on port '/tb/TOP/mid1/out2_bot' toggled when its control signal is activated. # File: test.upf, Line: 70, Power Domain:PD_mid1</p>
Isolation Toggle Check	vopt -pa_checks=it	<p>Catches any change in isolated ports' value during isolation period (such as in between the period when isolation is enabled till isolation is disabled). Performed during active isolation period.</p> <p>Error: (vsim-PA-8908) MSPA_ISO_ON_ACT : Time: 60 ns, /tb/TOP/mid1/out1_bot toggled when isolation signal was active</p>

Table 5-4. Dynamic Isolation Checks (cont.)

Check	Usage Syntax	Description Example Message
Missing Isolation Cell	vopt -pa_checks=umi	<p>Catches failure occurring whenever source domain is OFF and sink domain is ON for a power domain crossing and isolation strategy is not specified for that domain crossing.</p> <p>Error: (vsim-8929) MSPA_UPF_MISSING_ISO_CHK: Time: 130 ns, Missing isolation cell for domain boundary, PD_mid1 => PD_wrapper2 for following: # Source port : /tb/TOP/mid1/out1_bot [connected mask: "1"] [LowConn] -> Sink port: /tb/TOP/mid2/wrapper1/wrapper2/in1 [connected mask: "1"] [LowConn], # File: test.upf, Line: 25, Power Domain:--</p>
Power on	vopt -pa_checks=upc	<p>Catches switching off of retention/isolation supplies during active retention/isolation period.</p> <p>Error: (vsim-8920) MSPA_UPF_PG_CHK: Time: 0 ns, Power for Isolation strategy: 'ISO_FA4_1_3' of power domain: 'PD_FA4_1' is switched OFF during isolation. # File: test.upf, Line: 121, Power Domain:PD_FA4_1</p>
All	vopt -pa_checks=i	<p>Enable isolation checking performed by all arguments, this value enables only iep, idp, irc, icp, ifc, and it.</p>

Dynamic Level Shifter Checking

You enable dynamic checking functions for level shifters by using the **vopt -pa_checks** command. The value you specify determines the checking function performed, as shown in [Table 5-5](#).

Operating Voltage for Dynamic Checking

It is recommended to change the operating voltages of those domains during simulation. In some cases, the dynamic checks report the operating voltage of one of the domains as 0—this happens when you have not changed the voltage on the primary power and ground pin of the domain and you are operating with default unknown voltage levels. You can change the

operating voltages by using the **supply_on** or **supply_off** commands defined in the UPF SystemVerilog package.

Table 5-5. Dynamic Level Shifter Checks

Check	Usage Syntax	Description Example Message
Missing Level Shifter	vopt -pa_checks=uml	<p>During simulation, checks if a level shifter is required between domain crossing and reports if a level shifter strategy is not specified.</p> <p>Also, strategies specified with set_level_shift -no_shift in the UPF file are checked for any missing level shifters.</p> <pre># ** Error: MPSA_UPF_MISSING_LS_CHK: Missing level shifters for domain boundary, pd_aon (Operating Voltage: 2.000000 V) => pd_top (Opera ting Voltage: 1.000000 V) for the following: # Source port : /tb/TOP/bot4/out1_bot [LowConn] -> Sink port: /tb/TOP/bot4/out1_bot [HighConn] # Time: 90 ns Scope: mspa_top.mspa_upf_top.mspa0_pd_tb.mspa3_pd_top. mspa5_pd_aon. MPSA_UPF_MISSING_LS_CHK_0 File: src/inc_ls_dyn_check1/top.upf Line: 4</pre>
Incorrect Level Shifter	vopt -pa_checks=uil	<p>Checks if the direction of level shifters specified for domain crossing does not match the direction of the voltage difference at domain crossing. For example, domain crossing requires low_to_high level shifter but level shifter specified is high_to_low, then it will be reported as incorrect.</p> <pre># ** Error: MPSA_UPF_INCORRECT_LS_CHK: Shift mismatch for level shifter(my_ls_bot3, rule: low_to_high, Domain: pd_aon) at domain boundary, pd_aon (Operating Voltage: 2.000000 V) => pd_top (Operating Voltage: 1.000000 V) for following: # Source port : /tb/TOP/bot3/out1_bot [LowConn] -> Sink port: /tb/TOP/bot3/out1_bot [HighConn] # Time: 90 ns Scope: mspa_top.mspa_upf_top.msp a0_pd_tb.mspa3_pd_top.mspa5_pd_aon. my_ls_bot3. MPSA_UPF_INCORRECT_LS_CHK_1 File: src/inc_ls_dyn_check1/top.upf Line:4</pre>
All	vopt -pa_checks=ul	Enables all dynamic level shifter checks.

Miscellaneous Dynamic Checking

Table 5-6 lists values you can specify for **vopt -pa_checks** that perform the following checking operations:

Table 5-6. Miscellaneous Dynamic Checks

Check	Usage Syntax	Description Error Messages
Toggle	vopt -pa_checks=t	<p>Enables the simulator to catch a condition where inputs to the power domain toggle even when the power domain is turned off.</p> <p># ** Error: (vsim-PA-8908) MSPA_PD_OFF_ACT: Time: 36 ns, /tb/top/clk toggled during power down of power domain: PD</p>
Control Signal Corruption	vopt -pa_checks=cp	<p>During simulation, this check helps catch conditions when the power signal to any power domain gets corrupted.</p> <p>This check will not flag a violation when both the source and sink power domain supplies are off.</p> <p>For UPF, this check is applicable to control ports of a switch, isolation enable signal of an isolation strategy, and retention save and restore signals of a retention strategy.</p> <p># ** Error: (vsim-8901) MSPA_CTRL_SIG_CRPT: Time: 240 ns, Control Signal '/tb/pg_array[1]' is corrupted. (Current Value: x) #File: test.upf, Line: 29, Power Domain:PD_TOP</p>
Power Domain Status	vopt -pa_checks=p	<p>This check identifies when each power domain is switched on or off.</p> <p>MSPA_PD_STATUS_INFO: Power domain 'PD' is powered down at time 20.</p> <p>MSPA_PD_STATUS_INFO: Power domain 'PD' is BIASED down at time 20. (CORRUPT_ON_CHANGE)</p> <p>MSPA_PD_STATUS_INFO: Power domain 'PD' is BIASED up at time 24. (CORRUPT_ALL_ON_ACT)</p>

Table 5-6. Miscellaneous Dynamic Checks (cont.)

Check	Usage Syntax	Description Error Messages
Illegal or Undefined State	vopt -pa_checks=pis	<p>This check flags a violation if there is an undefined or illegal state on a PST or supply port.</p> <p>A state is undefined or illegal if the state of nets or ports is not a valid combination as related to the power state table in the UPF file.</p> <p>A state on a supply port is undefined if it is not defined with add_port_state for the supply port.</p> <pre># ** Error: (vsim-8933) MSPA_UPF_ILLEGAL_STATE_REACHED: Time: 80 ns, PST 'eth_pci_pst' reached an undefined state. # {tb_pow = (tb_nom : 4.800000 V), mid1_MAIN_NET = (undefined : 4.000000 V), mid2_MAIN_NET = (NOM2 : 5.300000 V)} # File: ./src/illegal_pst_2/test.upf, Line: 79, PST state:UNDEFINED # ** Error: (vsim-8933) MSPA_UPF_ILLEGAL_STATE_REACHED: Time: 90 ns, Port 'TB_PRI' reached an undefined state. # {TB_PRI = (undefined : 4.099999 V)}</pre>
Glitch Detection	vopt -pa_checks=ugc	<p>Catches any spurious spikes (glitches) on control lines so that it does not cause false switching of control ports of various control logic (such as isolation/power switch and retention).</p> <p>You can use pa msg -glitch_window command to specify time window for glitch checking.</p>
Non-Retention Register Reset	vopt -pa_checks=npu	<p>Checks that non-retention registers are reset when the power domain containing them is powered up.</p> <pre>** Error: (vsim-8912) MSPA_NRET_ASYNCFF: Time: 12 ns, Asynchronous(set/reset) control for the following flop(s) of power domain 'PD1' is not asserted at power up: # /tb/top_inst/out1. # File: test.upf, Line: 4, Power Domain:PD1</pre> <p>Also generates the report file <i>report.nretsyncff.txt</i>.</p>

Implementing Checking at Gate Level

For Power Aware simulation on gate-level designs (PA-GLS), you can generate various level shifter and isolation checks.

Related Topics

[Voltage Level-Shifting \(Multi-Voltage Analysis\)](#)

[Modeling Isolation](#)

Level Shifting for Gate-Level Checking

You can create a gate-level instance of a model that will be recognized as a level-shifter instance by either of the methods given below. This type of level-shifter instance will be recognized for static and dynamic Level Shifter Checks.

Method 1

Specify the Liberty **is_level_shifter** attribute as part of the module definition to identify the level shifter cell. For example:

```
(* is_level_shifter = 1 *)
module ls_buf(
    (*pg_type = "primary_power"*) input logic pwr_rail,
    (*pg_type = "primary_ground"*)input logic gnd_rail,
    (* level_shifter_data_pin = 1 *)input data,
    output logic out);
assign out = (data);
endmodule
```

Method 2

Assign a level-shifting prefix or suffix string to the instance, as specified by the UPF **name_format** command. For example:

```
LVLHLD1BWP lsinst2_UPF_LS(.I(w2), .Z(w4));
```

Isolation for Gate-Level Checking

A gate-level instance of a model that will be recognized as an isolation instance can occur by any of the methods given below.

Method 1

Create this instance by specifying the Liberty **is_isolation_cell** attribute as part of the module definition to identify the isolation cell. Also, set the **isolation_cell_enable_pin = "TRUE"** on the instance enable pin. For example:

```
(* is_isolation_cell = 1 *)
```

```

module ISO_AND(
    (*pg_type = "primary_power"*) input logic pwr_rail,
    (*pg_type = "primary_ground"*) input logic gnd_rail,
    (*isolation_cell_enable_pin = "TRUE"*) input logic en,
    (*isolation_cell_data_pin = "TRUE"*) input data,
    output logic out);
assign out = (data & ~en);
endmodule

```

Method 2

Pragma information to the instantiation is generated by the synthesis application. For example:

```

ISOLOD1BWP isoinst1(.I(o1), .Z(w1), .ISO(ctrl)); //synopsys isolation_upf
iso_PD_mid1+PD_mid1

```

where **iso_PD_mid1** is the UPF strategy for this isolation cell instance, and **PD_mid1** is the Power domain for which strategy has been specified.

Method 3

Create this instance by assigning an isolation prefix or suffix string to it, as specified by the UPF `name_format` command. For example:

```

ISO isoinst2_UPF_ISO(.I(o1), .Z(w1), .ISO(ctrl));

```

Quick-Reference Comparison of Static and Dynamic Check Arguments

Table 5-7. Static and Dynamic Checks Comparison

Check	Static Syntax	Dynamic Syntax
Missing Level Shifter	-pa_checks=sml	-pa_checks=uml
Redundant Level Shifter	-pa_checks=srl	
Incorrect Level Shifter	-pa_checks=sil	-pa_checks=uil
Unanalyzed level shifter	-pa_checks=snl	
Valid level shifter	-pa_checks=svl	
Uninserted level shifter	-pa_checks=sdl	
Missing Isolation Cell	-pa_checks=smi	-pa_checks=umi
Redundant Isolation Cell	-pa_checks=sri	
Incorrect Isolation Cell	-pa_checks=sii	

Table 5-7. Static and Dynamic Checks Comparison

Check	Static Syntax	Dynamic Syntax
Valid Isolation	-pa_checks=svi	
Isolation not analyzed	-pa_checks=sni	
Isolation not inserted	-pa_checks=sdi	
Isolation Enable Protocol		-pa_checks=iep
Isolation Disable Protocol		-pa_checks=idp
Isolation Race Check		-pa_checks=irc
Isolation Enable Protocol Check for COA States		-pa_checks=iepcoa
Isolation Disable Protocol Check for COA States		-pa_checks=idpcoa
Isolation Functionality Check		-pa_checks=ifc
Isolation Clamp Value Check		-pa_checks=icp
Isolation Redundant Activity Check		-pa_checks=ira
Isolation Toggle Check		-pa_checks=it
Retention Enable Protocol		-pa_checks=rop
Retention Enable/Disable Protocol		-pa_checks=rpo
Latch Enable/Clock Level Protocol		-pa_checks=rcs
Latch Enable/Clock Toggle Protocol		-pa_checks=rsa
Primary Supply		-pa_checks=cp
Isolation and Retention Supply		-pa_checks=upc
Non-Retention Register Reset		-pa_checks=npu
Glitch Detection		-pa_checks=ugc
Toggle		-pa_checks=t
Always-on Power Domain		-pa_checks=a
Power Domain Status		-pa_checks=p
Illegal or Undefined State		-pa_checks=pis

Chapter 6

Power Aware Coverage

Power aware coverage data helps you ensure that your regression suites are adequately testing power aware elements of your design, specifically:

- Dynamic power aware checks, as specified with the **-pa_checks** switch to the **vopt** command.
- Power aware states and transitions, as created through the UPF file with the **add_power_state**, **add_port_state** or **add_pst_state** commands.

This chapter describes the following topics:

- [Power Aware Coverage Collection](#) — provides procedures for collecting coverage data.
- [Power Aware Coverage Analysis](#) — provides procedures for analyzing your collected coverage data.
- [Power Aware Coverage Report Reference](#) — provides reference information about the power aware coverage reports.

Power Aware Coverage Collection

This section describes how you can alter your existing scripts to collect coverage data for your design and test suite.

- [Collecting Power Aware Coverage Information on Dynamic Checks](#)
- [Collecting Power Aware Coverage Information on States and Transitions](#)

You can view the coverage information during a live simulation or analyze it later by saving it to a Unified Coverage Data Base (UCDB) file. Refer to [Saving Code Coverage in the UCDB](#) in the User's Manual for more information.

Collecting Power Aware Coverage Information on Dynamic Checks

You can collect coverage information on dynamic checks and analyze the results within the ModelSim environment. Refer to the section "[Dynamic Checking in Power Aware](#)" for more information.

Prerequisites

- Enable dynamic checks with the **-pa_checks** switch to **vopt**.

Procedure

1. Compile:

You do not need to alter your existing commands.

2. Optimize and Simulate:

- Enable the default coverage behavior. Specifically, a power aware dynamic check is marked as “covered” when the check has been attempted and never failed (fail count = 0).
 - **vopt**: Add the **-pa_coverage=checks** switch
 - **vsim**: No changes needed
- Enable advanced coverage behavior. Specifically, a power aware dynamic check is marked as “covered” when the check has never failed and has passed at least once (fail count = 0 and pass count > 0).
 - **vopt**: Add the **-pa_coverage=checks** switch
 - **vsim**: Add the **-assertcover** switch
- View any unattempted power aware check assertions. Specifically, the coverage data includes information about any unattempted power aware checks. You can use this in combination with the other two options.
 - **vopt**: Add the **-pa_coverage=checks** switch
 - **vsim**: Add the **-unattempteddimmed** switch

3. Generate UCDB — refer to the section “[Generating a UCDB File for Power Aware](#)”.

4. Generate reports — refer to the section “[Generating Power Aware Coverage Reports](#)”.

5. Analyze data after simulation — refer to the section “[Accessing Coverage Data for Post-Simulation Analysis](#)”.

Concepts

- **Dynamic check coverage** — A particular dynamic check will be performed when test vector will provide stimuli so that check is hit in simulation. For example, when you want to check for missing isolation cell for all power domain crossings, you can enable this check with the **-pa_check=umi** switch to **vopt**. To trigger this check during simulation, for a particular power domain crossing (such as, PD_SRC => PD_SINK), your test vector should create a stimuli such that power domain PD_SRC is OFF and PD_SINK is ON. If test vectors are not creating this scenario then this particular

dynamic check will not be hit during simulation. In this case see that there are no missing isolation cells in the design, but in fact this check is not even hit by test vectors.

- **PA Check Exclusions** — Any power aware checks for gate level cell instance scopes will not be taken into account for coverage.

In this case, they will be excluded for coverage calculation. You can find the list of excluded assertion information in the report *pachecks.excluded.txt*.

Collecting Power Aware Coverage Information on States and Transitions

You can collect coverage information on power aware states and transitions, as created in your UPF file, and analyze the results within the ModelSim environment. Refer to the section “[Power State and Transition Concepts](#)” for more information.

Prerequisites

- Include power aware state information in your UPF file.

Procedure

1. Compile
You do not need to alter your existing commands.
2. Optimize
You do not need to alter your existing command.
3. Simulate
Add the **-coverage** option to your existing command.
4. Generate UCDB — refer to the section “[Generating a UCDB File for Power Aware](#)”.
5. Generate reports — refer to the section “[Generating Power Aware Coverage Reports](#)”
6. Analyze data after simulation — refer to the section “[Accessing Coverage Data for Post-Simulation Analysis](#)”.

Concepts

- **Undefined Power State** — During simulation, when none of the named states (including predefined states) of a state object is active the tool-created “Undefined” state becomes active. This tool-inserted undefined state is representative of all the power states that are missing in the UPF while adding states on a particular UPF objects, such as power domain, Supply set, Supply port, PST

- **pa_coverageinfo** — The simulator generates a power aware coverage scope, which represents all of the power aware state and transition coverage data, inside the related design instance named **pa_coverageinfo**. The coverage scope contains information related to supply sets, power domains, ports, nets and power state tables.

Name conflicts of **pa_coverageinfo** are resolved by adding a counter to the end of the name (**pa_coverageinfo_<n>**) for the second and any further occurrences.

Power Aware Coverage Analysis

- [Generating a UCDB File for Power Aware](#)
- [Generating Power Aware Coverage Reports](#)
- [Accessing Coverage Data for Post-Simulation Analysis](#)

Generating a UCDB File for Power Aware

When you save your coverage statistics to a UCDB file you are able to accumulate and analyze the results during post-simulation analysis.

Refer to the Chapter “[Coverage and Verification Management in the UCDB](#)” in the User’s Manual for information on using UCDB files.

Prerequisites

- Complete a simulation where you have enabled coverage collection, as described in the section “[Power Aware Coverage Collection](#)”.

Procedure

Generate the UCDB containing information about all coverage for the whole design:

```
coverage save <name>.ucdb
```

Alternatively, if you want to generate a UCDB containing coverage data with only your power aware information, use the following command (adding -pa):

```
coverage save -pa <name>.ucdb
```

Generating Power Aware Coverage Reports

You can generate several formats of report to analyze the results of your power aware coverage. This section describes how you generate each type of report from either the command line or through the GUI during a live simulation.

If you want to generate reports and do not want to open a simulation, you should substitute **coverage report** with **vcover report** and add the UCDB file as an argument, for example:

```
vcover report pa.ucdb -assert -pa
```

Power Aware Checks Coverage Summary

This produces a simple report including a summary of the checks that never failed.

- Command line:

```
coverage report -assert -pa
```

- GUI:

- Menu: **Tools > Coverage Report > Text**
- Coverage Text Report dialog box:
 - **Verbosity:** Default
 - **Coverage Type:** Assertions and Power Aware

Refer to the section “[Power Aware Checks Coverage Summary](#)” for a detailed description of this report.

Power Aware Checks Coverage Detailed Report

This produces a detailed report showing each type of check with pass and failure information.

- Command line:

```
coverage report -assert -pa -details
```

- GUI:

- Menu: **Tools > Coverage Report > Text**
- Coverage Text Report dialog box:
 - **Verbosity:** Details
 - **Coverage Type:** Assertions and Power Aware

Refer to the section “[Power Aware Checks Coverage Detailed Report](#)” for a detailed description of this report.

Power Aware Checks Coverage Verbose Report

This produces a expansive report showing each type of check with detailed information.

- Command line:

```
coverage report -assert -pa -verbose -xml
```

- GUI:
 - Menu: **Tools > Coverage Report > Text**
 - Coverage Text Report dialog box:
 - **Verbosity:** Details
 - **Coverage Type:** Assertions and Power Aware
 - **Output Mode:** XML Format

It is suggested that you output to XML format for usability. You can then open the .xml file with a spreadsheet program or other XML parser.

- **Report Pathname:** Specify a filename with the .xml suffix.

Refer to the section “[Power Aware Coverage Verbose Report](#)” for a detailed description of this report.

Power Aware State Coverage Summary

This produces a summary report showing coverage information organized by state machine instance.

- Command line:

```
coverage report -pa
```
- GUI:
 - Menu: **Tools > Coverage Report > Text**
 - Coverage Text Report dialog box:
 - **Verbosity:** Default
 - **Coverage Type:** Power Aware

Refer to the section “[Power Aware Coverage Summary](#)” for a detailed description of this report.

Power Aware State Coverage Detailed Report

This produces a detailed report showing coverage information organized by machine instance.

- Command line:

```
coverage report -pa -details
```
- GUI:
 - Menu: **Tools > Coverage Report > Text**
 - Coverage Text Report dialog box:
 - **Verbosity:** Details

- **Coverage Type:** Power Aware

Refer to the section “[Power Aware Coverage Detailed Report](#)” for a detailed description of this report.

Generating HTML Reports

As an alternative to the text-based reports, you can create a navigation-ready HTML report of your power aware coverage data.

- Command line:

```
coverage report -pa -html
```
- GUI:
 - Menu: **Tools > Coverage Report > HTML**
 - Coverage HTML Report dialog box:
 - **Other Options:** Power Aware

Accessing Coverage Data for Post-Simulation Analysis

Prerequisites

- Enable coverage collection during your power aware flow. Refer to the section “[Power Aware Coverage Collection](#)”.
- Create a UCDB based on a the power aware simulation. Refer to the section “[Generating a UCDB File for Power Aware](#)”.

Procedure

1. Load UCDB

```
vsim -viewcov <name>.ucdb
```

This loads the UCDB into the GUI for your analysis.

2. Generate reports

```
coverage report -pa [-details]
```

Refer to the Chapter “[Coverage and Verification Management in the UCDB](#)” in the User’s Manual for information on using UCDB files.

Power Aware Coverage Report Reference

This section contains an overview of the various coverage reports you can generate, as described in the section “[Generating Power Aware Coverage Reports](#)”.

Power Aware Coverage Summary

This report provides an overview of information about all power aware coverage metrics, including state and transition coverage and power aware check coverage.

Generation

```
coverage report -pa
```

Example

```
Coverage Report Summary Data by PA state machine instance
Total Coverage By Instance (filtered view): 37.8%

PA State Machine Instance: /tb/CPU1/pa_coverageinfo/PD_RAM_sw/out_sw_PD_RAM
UPF Object PORT: /tb/CPU1/PD_RAM_sw/out_sw_PD_RAM
Enabled Coverage      Active      Hits      Misses % Covered
-----
States                4          0         4        0.0
Transitions          16         0        16        0.0

PA State Machine Instance: /tb/pa_coverageinfo/out_sw_PD_ALU
UPF Object PORT: /tb/out_sw_PD_ALU
Enabled Coverage      Active      Hits      Misses % Covered
-----
States                4          0         4        0.0
Transitions          16         0        16        0.0

PA State Machine Instance: /tb/pa_coverageinfo/PD_ALU_sw/out_sw_PD_ALU
UPF Object PORT: /tb/PD_ALU_sw/out_sw_PD_ALU
Enabled Coverage      Active      Hits      Misses % Covered
-----
States                4          2         2        50.0
Transitions          16         2        14        12.5

PA State Machine Instance: /tb/pa_coverageinfo/MyPowerStateTable
UPF Object PST: /tb/MyPowerStateTable
Enabled Coverage      Active      Hits      Misses % Covered
-----
States                5          0         5        0.0
Transitions          25         0        25        0.0

TOTAL POWER STATE COVERAGE: 43.7% POWER STATE COVERAGE TYPES: 12

POWER AWARE CHECKS COVERAGE SUMMARY
NEVER FAILED: 75.0% ASSERTIONS: 24
```

Power Aware Coverage Detailed Report

This report provides detailed information about the state and transition coverage metrics followed by details about the power aware check coverage metrics.

Generation

```
coverage report -pa -details
```

Example (Excerpt)

```
PA State Machine Instance: /tb/pa_coverageinfo/MyPowerStateTable
UPF Object PST: /tb/MyPowerStateTable
PA State Coverage:
  Enabled Coverage      Active      Hits      Misses % Covered
  -----
  States                5          0          5       0.0
  Transitions          25         0          25       0.0
=====PA State Machine Details=====
PA State Coverage for PA State Machine instance /tb/pa_coverageinfo/MyPowerStateTable --
PSM_ID: MyPowerStateTable_pst_state_info.state_var
Current State Object : MyPowerStateTable_pst_state_info.state_var
-----
State Value MapInfo :
-----
Line      State Name      Value
-----
118      Complete_on     5
118      Hibernate       4
118      Sleep           3
118      Reboot_alias    2
118      Reboot          1
Uncovered States :
-----
State
-----
Complete_on
Hibernate
Sleep
Reboot_alias
Reboot
Uncovered Transitions :
-----
Line      Trans_ID      Transition
-----
124       7             Complete_on -> Complete_on
123       8             Complete_on -> Hibernate
122       9             Complete_on -> Sleep
121       10            Complete_on -> Reboot_alias
120       11            Complete_on -> Reboot
124       13            Hibernate -> Complete_on
123       14            Hibernate -> Hibernate

...

123       32            Reboot -> Hibernate
122       33            Reboot -> Sleep
121       34            Reboot -> Reboot_alias
120       35            Reboot -> Reboot
Excluded States :
-----
State      Exclusion
-----
undefined  E-hit
Excluded Transitions :
-----
Line      Trans_ID      Transition      Exclusion
-----
118       0             undefined -> undefined  E-hit

Summary
-----
Active      Hits      Misses % Covered
-----
States      5          0          5       0.0
Transitions 25         0          25       0.0

POWER STATE COVERAGE:
-----
UPF OBJECT                                     Metric      Goal/ Status
                                                At Least
```

Power Aware Coverage Power Aware Coverage Report Reference

```

-----
TYPE : SUPPLY SET /tb/CPU1/PD_RAM_SS              75.0%      100 Uncovered
SUPPLY SET coverage instance \tb/CPU1/pa_coverageinfo/PD_RAM_SS/PS_PD_RAM_SS
      75.0%      100 Uncovered
Power State PD_RAM_SS_ON1                        100.0%      100 Covered
  bin ACTIVE                                     1           1 Covered
Power State PD_RAM_SS_ON2                        100.0%      100 Covered
  bin ACTIVE                                     1           1 Covered
Power State PD_RAM_SS_ON3                        100.0%      100 Covered
  bin ACTIVE                                     1           1 Covered
...
Power State DEFAULT_CORRUPT                      100.0%      100 Covered
  bin ACTIVE                                     4           1 Covered
TYPE : SUPPLY SET /tb/CPU1/PD_RAM_SS              14.2%      100 Uncovered
SUPPLY SET coverage instance \tb/CPU1/pa_coverageinfo/PD_RAM_SS/PS_TRANS_PD_RAM_SS
      14.2%      100 Uncovered
Power State Transitions                          14.2%      100 Uncovered
  bin PD_RAM_SS_ON1 -> PD_RAM_SS_ON2             0           1 ZERO
  bin PD_RAM_SS_ON1 -> PD_RAM_SS_ON3             0           1 ZERO
  bin PD_RAM_SS_ON1 -> PD_RAM_SS_ON4             0           1 ZERO
  bin PD_RAM_SS_ON1 -> PD_RAM_SS_ON5             0           1 ZERO
  bin PD_RAM_SS_ON1 -> PD_RAM_SS_ON6             0           1 ZERO
  bin PD_RAM_SS_ON1 -> DEFAULT_NORMAL            0           1 ZERO
  bin PD_RAM_SS_ON1 -> DEFAULT_CORRUPT           1           1 Covered
  bin PD_RAM_SS_ON2 -> PD_RAM_SS_ON1             0           1 ZERO
  bin PD_RAM_SS_ON2 -> PD_RAM_SS_ON3             0           1 ZERO
...
  bin DEFAULT_NORMAL -> PD_RAM_SS_ON5            0           1 ZERO
  bin DEFAULT_NORMAL -> PD_RAM_SS_ON6            0           1 ZERO
  bin DEFAULT_NORMAL -> DEFAULT_CORRUPT           0           1 ZERO
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON1            1           1 Covered
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON2            1           1 Covered
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON3            1           1 Covered
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON4            0           1 ZERO
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON5            0           1 ZERO
  bin DEFAULT_CORRUPT -> PD_RAM_SS_ON6            1           1 Covered
  bin DEFAULT_CORRUPT -> DEFAULT_NORMAL           0           1 ZERO
...
TOTAL POWER STATE COVERAGE: 43.7% POWER STATE COVERAGE TYPES: 12

```

POWER AWARE CHECKS COVERAGE RESULTS:

```

-----
Check          Coverage          Failure Pass
Type   Object
-----
Isolation Functionality Check
  /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_func_chk
                                0     2
Isolation Clamp Value Check
  /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_clamp_chk
                                3     2
Isolation Race Check
  /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/iso_port_toggle_pos/iso_toggle_posedge
                                0     3
Isolation Race Check
  /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/iso_port_toggle_neg/iso_toggle_negedge
                                0     2
Toggle Check
  /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_on_act
                                0     0
Missing Level Shifter
  /tb/CPU1/pa_missing_ls_check_1/mspa_upf_missing_ls_chk
                                0     4
Missing Isolation
  /tb/CPU1/pa_missing_iso_check_2/mspa_upf_missing_iso_chk
                                0     4
...

```

Power Aware Coverage Verbose Report

This report provides a view of all coverage information collected about the dynamic checks analyzed during the simulation. It is suggested that you generate this file in XML format to be loaded in a spreadsheet program for ease of use.

Generation

```
coverage report -pa -verbose -xml -file pa_checks.xml
```

Power Aware Checks Coverage Summary

This report provides a basic overview of the coverage statistics, including information collected from state and transition counts and dynamic check counts.

Generation

```
coverage report -assert -pa
```

Example

```
Total Coverage By Instance (filtered view): 62.5%
POWER AWARE CHECKS COVERAGE SUMMARY
NEVER FAILED: 75.0% ASSERTIONS: 24
```

Power Aware Checks Coverage Detailed Report

This report provides a detailed view of all the dynamic checks analyzed during the simulation. The information includes: the type of check, a hierarchical path to the check, and the failure and pass counts of the check.

Generation

```
coverage report -assert -pa -details
```

Example

```
Total Coverage By Instance (filtered view): 62.5%
POWER AWARE CHECKS COVERAGE RESULTS:
-----
Check   Coverage                               Failure Pass
Type   Object                                Count  Count
-----
Isolation Functionality Check
      /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_func_chk
                                           0      2
Isolation Clamp Value Check
      /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_clamp_chk
                                           3      2
Isolation Race Check
      /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/iso_port_toggle_pos/iso_toggle_posedge
                                           0      3
Isolation Race Check
      /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/iso_port_toggle_neg/iso_toggle_negedge
                                           0      2
Toggle Check
      /tb/CPU1/out_alu_UPF_ISO/pa_iso_cell_checks_vec/mspa_iso_on_act
                                           0      0
Missing Level Shifter
```

Power Aware Coverage

Power Aware Coverage Report Reference

```
    /tb/CPU1/pa_missing_ls_check_1/mspa_upf_missing_ls_chk
Missing Isolation
    /tb/CPU1/pa_missing_iso_check_2/mspa_upf_missing_iso_chk
Control Signal Corruption Check
    /tb/CPU1/pa_upf_control_check_1/mspa_ctrl_crpt_chk
Control Signal Corruption Check
    /tb/CPU1/RAM1/pa_rcs_check_vec_6/rcs_checks[2]/ctrl_chk
```

Chapter 7

Visualization of Power Aware Operations

You can use ModelSim GUI windows to visualize aspects of your Power Aware design and simulation. The following sections describe the ModelSim windows provided for Power Aware visualization:

- [UPF Object Display](#) — The Objects, Structure, and Wave windows provide visibility into UPF objects.
- [Power Aware Schematic Display](#) — The Schematic window provides color coding of design elements in different power domains.
- [Power Aware Waveform Display](#) — The Wave window provides highlighting of bias, corruption, and isolation.
- [Power State and Transition Display](#) — The FSM List and FSM Viewer windows provide finite state machine information on power state tables and multi-state transitions.

UPF Object Display

You can view objects created through the UPF file in the Structure, Objects and Wave windows by adding the `-pa_enable=supplynetworkdebug` option to your `vopt` command.

A special icon is used to identify UPF objects in these windows:

Figure 7-1. UPF Object Icon



Visualizing UPF Objects in the GUI

This task outlines how you can enable the visualization of UPF objects in the Structure, Objects and Wave windows

Prerequisite

- Have an existing power aware simulation flow.

Procedure

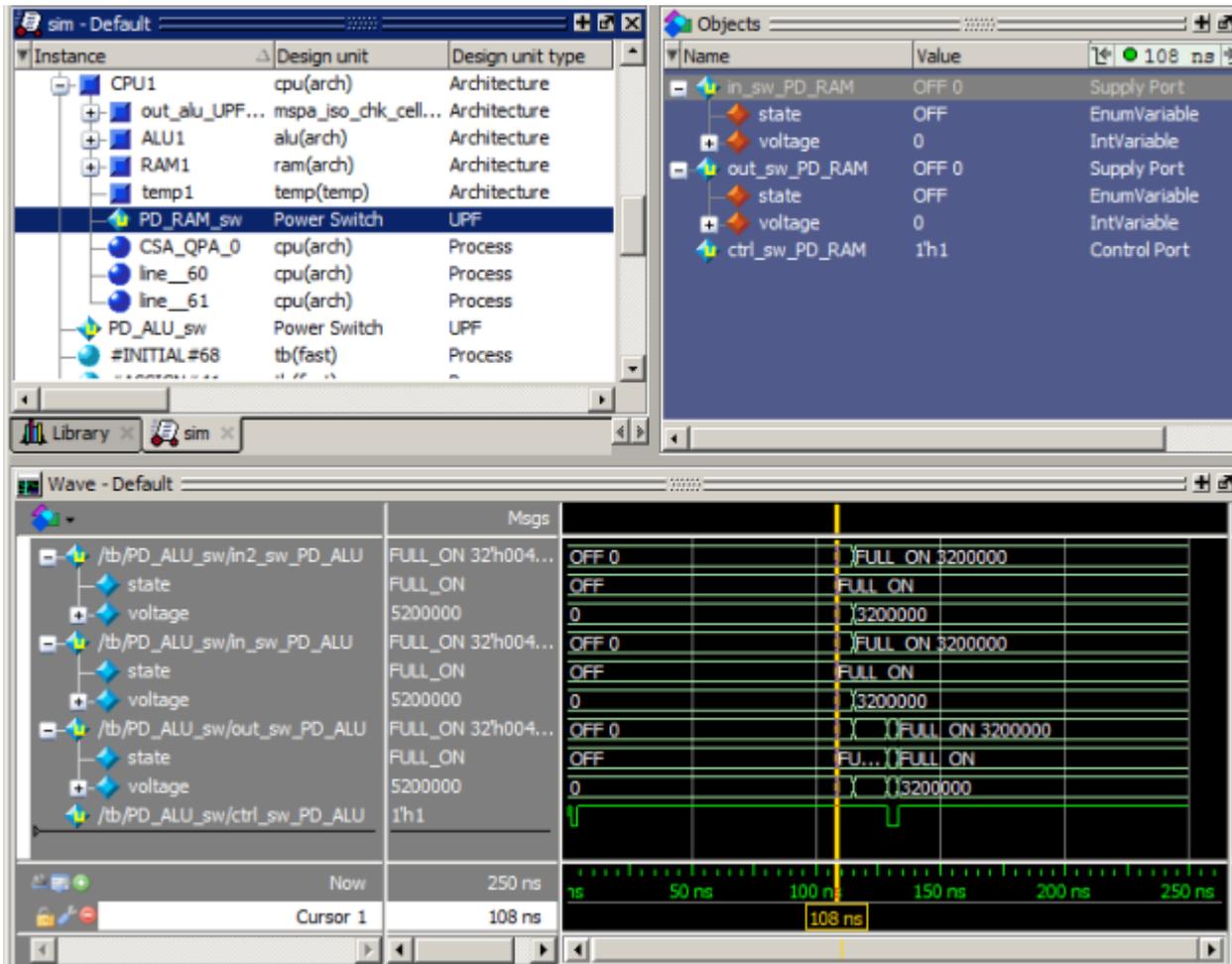
1. Open the ModelSim GUI.
2. Add the **-pa_enable=supplynetworkdebug** option to your **vopt** command.
3. Run your power aware simulation flow as you normally would.
4. Ensure the Objects, Structure, and Wave windows are open:
 - view structure
 - view objects
 - view wave
5. Locate and select a UPF object in the Structure Window (refer to [Figure 7-2](#)).
 - a. Click the UPF object to view additional port and net information in the Objects window.
 - b. Right-click the UPF object and select Add Wave to add all of the related ports and nets in the Wave window.

The default value of supply ports and nets in these windows are:

- Voltage (Decimal)
- State (Enum, UNDETERMINED, FULL_ON, OFF, PARTIAL_ON).

Results

Figure 7-2. Visualization of UPF Objects in the GUI



Power Aware Schematic Display

You can perform debugging at the same time you run a Power Aware simulation by adding the **-debugdb** argument to both of the **vopt** and **vsim** commands. The results of both the Power Aware analysis and the debug operation are provided as Power Aware schematic in the Schematic window. You can also view a correlation between the UPF power intent and the design display in the Schematic window.

Refer to the “[Schematic Window](#)” Chapter of the User’s Manual for more information.

Note



Debugging in Power Aware is supported for RTL usage flow only—it is not available for gate-level simulation (GLS).

Top-Down Debugging (From the Test Bench)

You can run Power Aware analysis and debugging from the top of the design (test bench) with or without specifying an optimized design unit. These methods resemble the conventional two-step and three-step optimization flows.

- No optimized design unit — This flow resembles the conventional two-step flow, where you use the **vopt** command to specify a Power Aware simulation and debugging only; no optimization is performed. When you run the **vsim** command, it performs debugging and runs **vopt** internally to perform optimization (see [Using the Delayed Optimization Flow](#)). For example:

```
vlog design.v
vcom design.vhdl
vopt -pa_upf <config_file> -debugdb tb
vsim tb -pa -debugdb [-vopt]
```

- Optimized design unit — This flow resembles conventional three-step flow, where you use the **vopt** command to specify a Power Aware simulation, the name of the design unit to be optimized, and debugging,. When you run the **vsim** command, it begins simulation on the optimized design unit and runs debugging ([General Steps for Running Power Aware](#)). For example:

```
vlog design.v
vcom design.vhdl
vopt -o optdu -pa_upf <config_file> -debugdb tb
vsim optdu -pa -debugdb
```

Bottom-Up Debugging (From the Design Under Test)

You can run Power Aware analysis from the DUT hierarchy and debugging on the complete design. Running **vopt -pa_top** captures the DUT hierarchy for Power Aware analysis

- No optimized design unit — This flow resembles conventional two-step flow, where you use the **vopt** command to specify a Power Aware simulation and debugging without optimization. Use the **-pa_top** argument to capture the DUT hierarchy for Power Aware analysis. When you run the **vsim** command, it performs debugging and runs **vopt** internally to perform optimization (see [Using the Delayed Optimization Flow](#)). For example:

```
vlog design.v
vcom design.vhdl
vopt -pa_upf <config_file> tb -debugdb -pa_top /tb/dut
vsim tb -pa -debugdb [-vopt]
```

- Optimized design unit — This flow resembles the conventional three-step flow, where you use the **vopt** command to specify a Power Aware simulation, the name of the design unit to be optimized, and debugging. Use the **-pa_top** argument to capture the DUT hierarchy for Power Aware analysis. For example:

```
vlog design.v
vcom design.vhdl
vopt -o optdu -pa_upf <config_file> tb -debugdb -pa_top /tb/dut
vsim optdu -pa -debugdb
```

This DUT-based flow with an optimized design unit not only does common analysis for Power Aware and debugging, but also provides flexibility to enable Power Aware analysis from specific hierarchy and do code generation in one step.

Usage Notes

- The **-pa_top** argument is used to specify hierarchy of UPF root scope. This supports Power Aware analysis of UPF from hierarchy other than the vopt TOP hierarchy. If **vopt** is run from test bench (tb) and UPF scope is starting from DUT (which is instantiated in test bench as dut_inst), then you need to specify vopt **-pa_top /tb/<dut_inst>**.
- If **-pa_top** is specifying the hierarchy other than UPF root scope then an Error message will be displayed:

```
** Error: ./src/ss_error_1/test.upf(2): UPF: (vopt-9782) PA Top
'/tb/top/dut_inst/top_inst' is specifying incorrect hierarchy for
UPF scope 'DUT'.
```

- Do not use the vopt **-pa_prefix** and **-pa_replacetop** arguments with **-pa_top**. If you do, the **-pa_prefix** and **-pa_replacetop** arguments are ignored and a Warning message is displayed:

```
** Warning: UPF: (vopt-9780) Option "-pa_prefix/-pa_replacetop" is
not applicable with option "-pa_top". Ignoring option "-pa_prefix/-
pa_replacetop".
```

Schematic Window Visualization for Debugging

You can open the Schematic window to view debugging results from a Power Aware analysis. In particular, these results are shown as follows:

- Power Domain — All design elements are colored and highlighted according to their respective power domains (see [Figure 7-3](#)).
 - All design elements, such as mux, flip-flops, and gates are colored according to the power domain specification.
 - The granularity of power domain visualization is at the instance level.
 - Any simulation-only power domains that are specified on a process or signal are not highlighted.
 - Colorization supports up to 16 different power domains.
 - If the power domain is in the OFF state, the schematic will be filled with a grey color (see [Figure 7-4](#)).

- Isolation cells will be filled in with a green color.
- Level-shifter cells will be filled in with a purple color.
- **Excluded Domains** — shown as default schematic color. You can define a power domain to be excluded.
 - Power Aware exclude file support (vopt **-pa_excludefile**)
 - Currently, PG-type connected instances are excluded. There is no analysis information to decide whether they inherit the parent power domain, create their own, or have multiple power domains (like memories)

Both are supported with instance level granularity, signal and process level exclusion will not be visualized.

- **UPF Source Viewing** — You can display the source text of the UPF file for power domain specifications. Power domain information can be viewed for a design element in either of the following ways:
 - Right-click and select **View Selection > Power Domain** — Displays UPF source code (see [Figure 7-5](#)).
 - Hover the mouse cursor — Displays a Tool Tip that concatenates the HDL source file with the appropriate line number in the UPF source file (see [Figure 7-6](#)).

Example 7-1. UPF File to Demonstrate Schematic Visualization for Debugging

```
set_design_top top

create_power_domain P1 -elements {dut1}
#-----
create_supply_port VDD      -domain P1
create_supply_net  VDD_NET -domain P1
connect_supply_net VDD_NET -ports { VDD }
#-----
create_supply_port GND      -domain P1
create_supply_net  GND_NET -domain P1
connect_supply_net GND_NET -ports { GND }
#-----
create_supply_net  VDD_PRI -domain P1
set_domain_supply_net P1 -primary_power_net VDD_PRI -primary_ground_net
GND_NET
#----- #

create_power_switch P1_SW \
-domain P1 \
-output_supply_port {VDD_SW VDD_PRI} \
-input_supply_port  {VDD_SW_In1 VDD_NET} \
-control_port {ctrl1 dut_sleep} \
-on_state {full_s1 VDD_SW_In1 {!ctrl1}} \
-off_state {off_s0 {ctrl1}}
```

```

create_power_domain P2 -elements {dut2}
#-----
create_supply_port VDD_P2      -domain P2
create_supply_net  VDD_N  -domain P2
connect_supply_net VDD_N -ports { VDD_P2 }
#-----
create_supply_port GND_P2      -domain P2
create_supply_net  GND_N  -domain P2
connect_supply_net GND_N -ports { GND_P2 }
#-----
create_supply_net  VDD_P -domain P2
set_domain_supply_net P2 -primary_power_net VDD_P -primary_ground_net
GND_NET

```

```

create_power_switch P1_SW2 \
-domain P2 \
-output_supply_port {VDD_SW VDD_PRI} \
-input_supply_port  {VDD_SW_In1 VDD_NET} \
-control_port {ctrl1 dut_sleep} \
-on_state  {full_s1 VDD_SW_In1 {!ctrl1}} \
-off_state {off_s0 {ctrl1}}

```

```

create_power_domain P3 -elements {dut3}
#-----
create_supply_port VDD_P3      -domain P3
create_supply_net  VDD_NE -domain P3
connect_supply_net VDD_NE -ports { VDD_P3 }
#-----
create_supply_port GND_P3      -domain P3
create_supply_net  GND_NE -domain P3
connect_supply_net GND_NE -ports { GND_P3 }
#-----
create_supply_net  VDD_PR -domain P3
set_domain_supply_net P3 -primary_power_net VDD_PR -primary_ground_net
GND_NE
create_power_switch P1_SW3 \
-domain P3 \
-output_supply_port {VDD_SW VDD_PRI} \
-input_supply_port  {VDD_SW_In1 VDD_NET} \
-control_port {ctrl1 dut_sleep} \
-on_state  {full_s1 VDD_SW_In1 {!ctrl1}} \
-off_state {off_s0 {ctrl1}}

```

Example 7-2. ModelSim Commands to Run Power Aware Debugging

```

vlog -sv mid.v top.v
vopt -pa_upf test.upf top -debugdb
vsim -debugdb -pa -L mtiPA -vopt -do test.do top

```


Figure 7-5. UPF Source File: Right-Click and Choose Power Domain

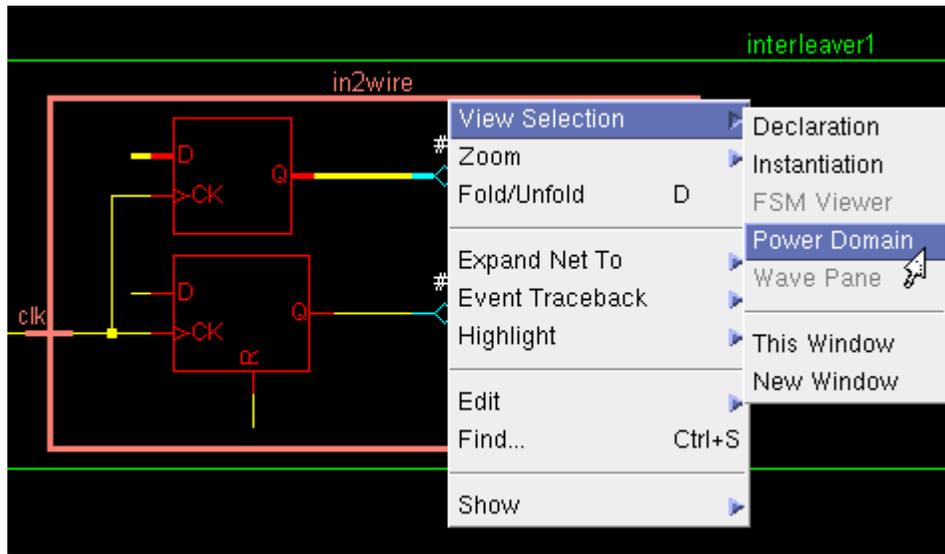
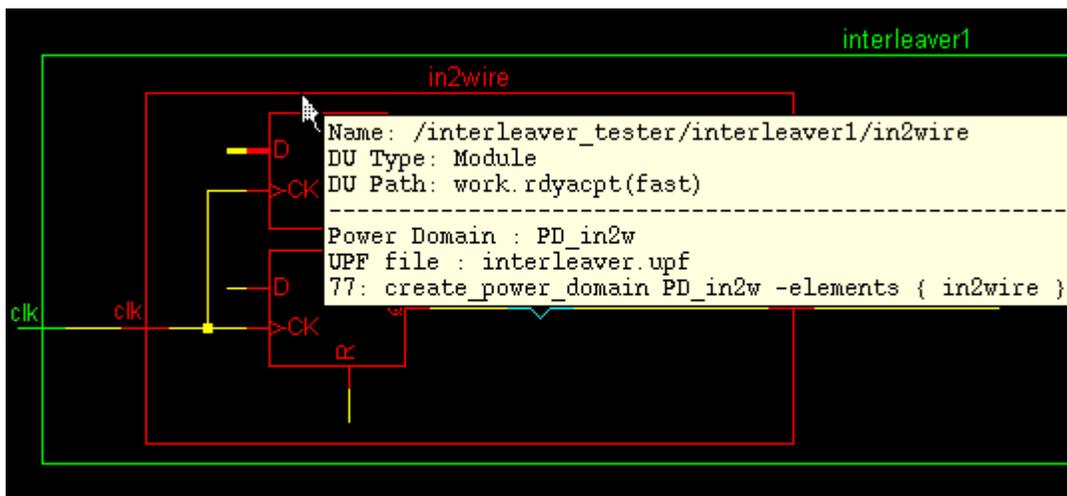


Figure 7-6. UPF Source File: Hover the Mouse and View Tool Tip



Power Aware Waveform Display

When using the conventional Wave window display, it can be difficult to see the effects of Power Aware simulation. For example, a zero on a signal may represent normal simulation behavior, it may be the result of an isolated port clamped to zero, or it could be corruption on a bit type.

In particular, isolation has been difficult to confirm through simulation that the intent has been met. Typically, you would want results to show isolation buffer placement and clamping, identify the corrupted and clamp values associated with that buffer, and confirm that isolation happens at the proper time.

To do this, you can activate Power Aware highlighting in the Wave window, which provides visual indicators for the isolation, corruption, and biasing behavior in your simulation results. These waveform indicators provide valuable information by visually distinguishing values caused by Power Aware activity. This should help quickly determine if their power intent is correctly applied.

The visual indicators provided by Power Aware highlighting show the power state of signals viewed in the Wave window. Highlighting on waveforms appears during the interval when they are corrupted, isolated, or biased.

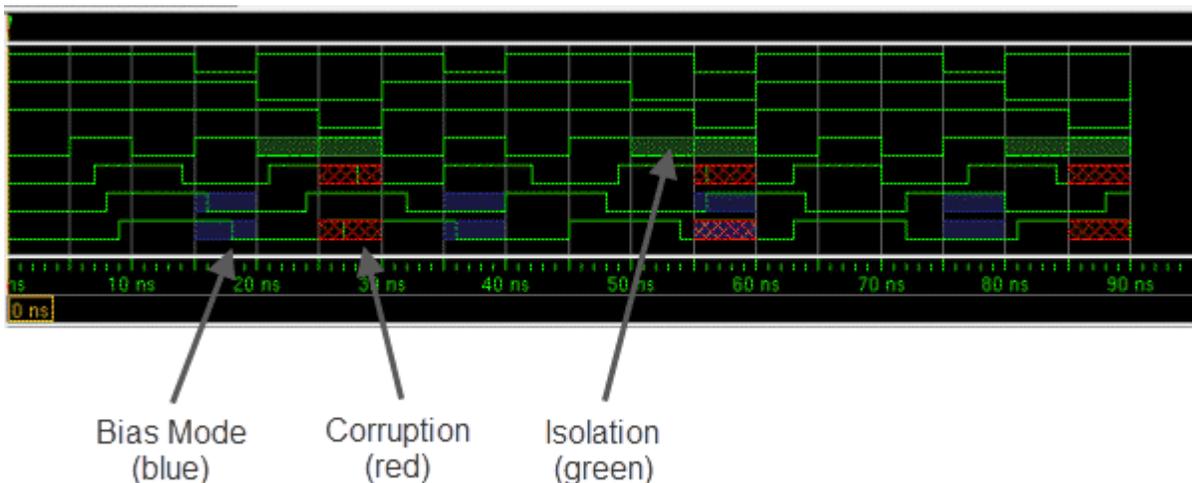
Figure 7-7 show an example of waveform highlighting, where:

- Bias mode is indicated by blue highlighting
- Corruption is indicated by red cross-hatch highlighting
- Isolation is indicated by green highlighting

i **Tip:** When you hover the mouse cursor over an isolation highlight region, a balloon popup appears. This indicates the clamp value and location, along with the actual signal value.

Also, when you click to expand an isolation highlighted signal (on the + to the left of the signal name), associated signals are displayed that provide more information about the isolation.

Figure 7-7. Power Aware Highlighting in the Wave Window



Using Power Aware Highlighting

To enable Power Aware highlighting, use the following flow:

1. Enable highlighting during elaboration by adding the **-pa_enable=highlight** option to your **vopt** command line.
2. Enable highlighting during simulation by adding the **-pa_highlight** option to the **vsim** command.

This argument enables the generation of the WLF data used by the Wave window to display the highlighting.

3. Enable highlighting in the GUI by selecting:

Wave > Tools > Wave Preferences... > [Display tab] > PA waveform highlighting

You only need to perform this action upon your first invocation, or if your GUI settings are reset.

4. View Power Aware activity in post-simulation debug by loading the WLF file containing the information as follows

```
vsim -view vsim.wlf
```

Power State and Transition Display

Because power domains are not limited to two states (ON or OFF) and multi-voltage capability allows designs to assign different voltage levels to different states, tracking combinations of states in different power domains has become increasingly difficult.

Power State and Transition Concepts

In UPF, you can add states on the following objects and model a Power Aware state machine (PASM) corresponding to each:

- PST — resulting from the **add_pst_state** UPF command
- Supply port — resulting from the **add_port_state** UPF command.
- Supply net — resulting when a PST contains a supply net entry.
- Power domain — resulting from the **add_power_state** UPF command
- Supply sets — resulting from the **add_power_state** UPF command

A power state table (PST) defines the allowable combinations of power states of supply ports and nets—those combinations of states that can exist at the same time during simulation of the design. As a result, changing the power state supply port/nets changes the state of PST.

The UPF command **add_power_state** adds power states on power domains and supply sets. The power states associated with supply sets determine the simulation semantics of connected design elements (power domain).

Power domain elements can be in one of the following simstates based on which power states of the related supply set are true at a given simulation time: **NORMAL**, **CORRUPT**, **CORRUPT_ON_ACTIVITY**, **CORRUPT_STATE_ON_CHANGE** or **CORRUPT_STATE_ON_ACTIVITY**.

Typically power states are composed hierarchically while creating power intent, for example the power state of a top-level power domain depends upon power states of any contributing power domains. These contributing power states (leaf level power domains) then depend upon power states of their supply sets.

Power states of a supply set depend upon power states of supply nets associated with functions of the supply set. Therefore, as design complexity increases, such as with IP-level hierarchical power intent, these power states become drivers for the whole power aware verification and it becomes important to provide debugging facilities for power states.

Note



You can find an example of a UPF file with a PST at:

`<install_dir>/examples/pa_sim/example_one/rtl_top.upf`

Differences Between a Conventional RTL FSM and a PASM

Some conceptual differences between conventional RTL Finite State Machines (FSM) and Power Aware state machines are:

- Power Aware State machines are asynchronous in nature—there use no concept of clock.
- Power Aware state machines can reach multiple states at a time (Nondeterministic Situation).
- Power Aware state machines are modeled by Power Aware simulation.
- There are a few interdependent Power Aware state machines. For example, a PST state machine runs in accordance to supply port/net Power Aware state machines.

Visualizing Power Aware State Machines

Dynamic visualization with debugging capabilities of power states is useful for verification of power intent.

Prerequisites

- Your UPF file contains power state information (**add_power_state**, **create_pst**, **add_port_state**, and/or **add_pst_state** UPF commands).
- vsim command includes -coverage option

Process

1. Begin your simulation.
2. Analyze state machines from your design in the [Power Aware State Machine List Window](#).
3. View bubble diagrams of individual state machines in the [Power Aware State Machine Viewer Window](#).

Power Aware State Machine List Window

Use this window to view a list of power aware state machines after beginning a Power Aware simulation.

This window lists all of the Power Aware State Machines in the design with the hierarchical path of their creation.

For example, if a PST has been created in **dut/top**, then this PST will be visible in scope **dut/top/pa_coverageinfo**.

This window is very similar to the [FSM List Window](#).

Accessing

Access the window using either of the following

- **View > PA State Machine List**
- Command: view powerstatelist

Figure 7-8. Power Aware State Machine List Example

Instance	States	Transitions	UPF TYPE
VDD_0d99_port_state_info.state_var	2	4	PORT
VDD_0d81_port_state_info.state_var	2	4	PORT
vout_p_port_state_info.state_var	3	9	PORT
VSS_port_state_info.state_var	2	4	PORT
top_pst_pst_state_info.state_var	3	9	PST
VDD_0d99_port_state_info.state_var	2	4	PORT
VDD_0d81_port_state_info.state_var	2	4	PORT
VSS_port_state_info.state_var	2	4	PORT
vout_p_port_state_info.state_var	3	9	PORT

GUI Elements of the Power Aware State Machine List Window

This section describes GUI elements specific to this window.

Column Descriptions

Table 7-1. Power Aware State Machine List Window Columns

Column	Description
Instance	Instance name of the state machine. The hierarchical display shows state dependency, specifically those instances where states of a particular power state machine depend upon states of other power state machines.
States	Number of states.
Transitions	Number of transitions between the states.
UPF Type	One of the following: PORT, NET, PST, POWER DOMAIN, SUPPLY SET

Popup Menu

Table 7-2. Power Aware State Machine List Window Popup Menu

Popup Menu Item	Description
View State Machine	Displays a graphical representation of the power aware state transitions in the Power Aware State Machine Viewer Window
View UPF	Opens the UPF file and displays the line creating the power aware state machine.
Add to	Adds the selected (or all) state machines to the Wave window, List window, or to the log (add log).
Properties	Displays the Power Aware State Machine Properties dialog box, which contains detailed information about the selection.

PA State Machines Menu

Table 7-3. PA State Machines Menu

Menu Item	Description
View State Machine	Displays a graphical representation of the power aware state transitions in the Power Aware State Machine Viewer Window .
View UPF	Opens the UPF file and displays the line creating the power aware state machine.
Add to	Adds the selected (or all) state machines to the Wave window, List window, or to the log (add log).
Options	Opens the State Machine Display Options dialog box that allows you to control: <ul style="list-style-type: none"> • Specific instructions when adding state machines to the Wave window • Limit the number of path elements in the Instance column.

Power Aware State Machine Viewer Window

Use this window to view a state diagram of any of your power aware state machines.

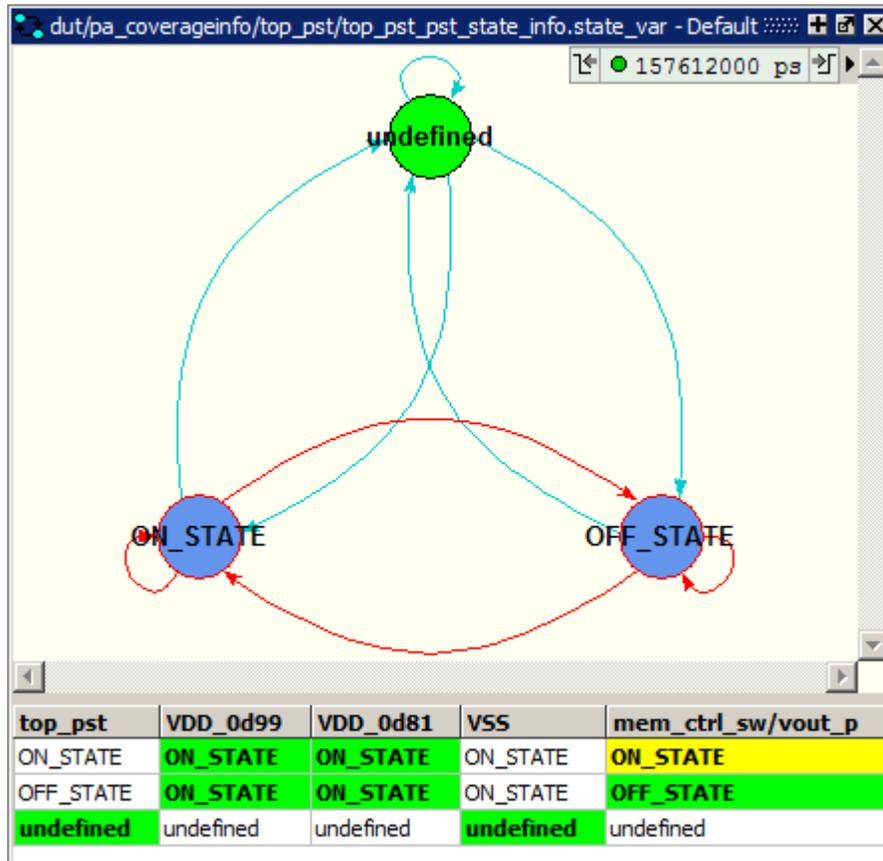
This window is very similar to the [FSM Viewer Window](#).

Accessing

Access the window as follows:

- From the Power Aware State Machine List Window, double click on the power aware state machine you want to analyze

Figure 7-9. Power Aware State Machine Viewer Window Example



GUI Elements of the Power Aware State Machine Viewer Window

This section describes GUI elements specific to this window.

Window Element Descriptions

- Title bar — The UPF type and hierarchical path.
- Tab label — Leaf name of the state machine.
- State Diagram — All states of a state machine and their transitions.
 - Bubbles — Information associated with logic expression supply expression simstate legality. Information appears in properties box associated with each state bubble.

- Color — Green represents the current state and yellow represents the previous state. A thick red border identifies an illegal state (as allowed with **add_power_state**).
- Label — State name
- Transitions — Directional arrows showing transitions between the states.
- Time Mode (upper right-hand corner) — The time of the current state transition, which decides yellow and green. Also affects States table.
- Goto Previous and Next State buttons — Allows synchronization between other state machine windows and the wave window and the objects window.
- Synchronization — By default, this window is synchronized with the Wave window, Objects window, and other Power Aware State Machine View windows.

State Machine to State Machine — State machines opened will remain synchronized with each other, related to the simulation time stamp. This enables you to work with multiple power aware state machines.

State Machine to Wave — The Wave window and Power Aware State Machine Viewer windows are synchronized to facilitate your debugging. When you move a wave window time cursor to observe various values of signals in wave window, the Power Aware State Machine Viewer window changes to match the same simulation time stamp, and the reverse is true.

- States table — (Only appears for supply sets, power domains, and PSTs) Shows current and previous values of objects that decide the power state of the supply set or power domain. This table helps you perform root cause analysis of your power aware states. When you encounter an unexpected power state, this feature allows you to understand the reason behind it, specifically when looking for unexpected, illegal, or undefined power states.
 - Column headers — Shows objects (power domains, supply sets, supply nets, logic variables) that contribute to the power states. If a header is associated with a power domain or supply set, you can double click on it to view the state transitions of that state machine. For PSTs, the column headers show ports and nets from the creation of the PST with the **create_pst** UPF command and you can double click on them to view the state transitions of that state machine.
 - Rows — Shows values for the time selected in the Time Mode widget, where green indicates the current value and yellow indicates the previous value.

Popup Menu

Table 7-4. Power Aware State Machine Viewer Window Popup Menu

Popup Menu Item	Description
Transition > View Full Text	Displays the full text of all condition expressions.
View UPF	Opens the UPF file and displays the line creating the power aware state machine.
Show States Table	Toggles the display of the states table for supply sets and power domains.
Zoom Full	Fits the bubble diagram into the visible space.
Set Context	Sets the context (env command) of the session based on your selection.
Add to	Adds the selected (or all) state machines to the Wave window, List window, or to the log (add log).
Properties	Displays the Power Aware State Machine Properties dialog box, which contains detailed information about the selection.

FSM View Menu - For Power Aware State Machines

Table 7-5. FSM View Menu, Specific to Power Aware State Machines

FSM View Menu Item	Description
Show States Table	Toggles the display of the states table for supply sets and power domains
Show States Counts	Displays the coverage counts for each state in the state bubble.
Show Transition Counts	Displays the coverage counts for each transition.
Show Transition Conditions	Displays the condition for each transition. The condition format is based on the GUI_expression_format Operators .
Enable Info Mode Popups	Displays popup information when you hover over a state or transition.
Track Wave Cursor	Displays current and previous state information based on the cursor location in the Wave window.

Table 7-5. FSM View Menu, Specific to Power Aware State Machines (cont.)

FSM View Menu Item	Description
Transitions to “Reset”	Controls the display of transitions to a reset state: <ul style="list-style-type: none"> • Show All • Show None — will also add a “hide all” note to the lower-right hand corner. • Hide Asynchronous Only • Combine Common Transitions — (default) creates a single transition for any transitions to reset that use the same condition. The transition is shown from a gray diamond that acts as a placeholder.
Options	Displays the FSM Display Options dialog box, which allows you to control: <ul style="list-style-type: none"> • how FSM information is added to the Wave Window. • how much information is shown in the Instance Column

Appendix A

Power Aware Commands and Options

Note



The functionality described in this chapter requires an additional license feature for ModelSim SE. Refer to the section “[License Feature Names](#)” in the Installation and Licensing Guide for more information or contact your Mentor Graphics sales representative.

This appendix provides reference information on the following:

- [ModelSim Commands Used for Power Aware](#) — a summary of the arguments for the vopt and vsim commands that you use to implement a Power Aware simulation.
- [Additional Commands You Can Use with Power Aware](#) — a summary of Tcl commands that you can use to control a Power Aware simulation.
- [Power Aware Messages](#) — basic information on message handling for Power Aware simulation.
- [Excluding Design Elements from Power Aware](#) — a description of how to use vopt -pa_excludefile to skip Power Aware processing for any module, its instances in a particular hierarchical path in the design, or signals in the design.
- [Voltage Level-Shifting \(Multi-Voltage Analysis\)](#)
- [Restricting Isolation and Level Shifting on a Port](#)
- [Simulating Designs Containing Macromodels](#)
- [Creating Feedthrough For RTL Conversion Functions](#)

ModelSim Commands Used for Power Aware

Tables A-1 and A-2 list the arguments for the `vopt` and `vsim` commands that you use to run a Power Aware simulation. Refer to the *ModelSim Reference Manual* for a more comprehensive description of these commands and their arguments.

Table A-1. Power Aware Arguments for vopt

vopt Argument	Argument Value(s)
-pa_all	<i>none</i>
-pa_behavlogfile	<filename>
-pa_ce	o os osw sc scb
-pa_cfg	<filename>
-pa_checks=	Quick-Reference Comparison of Static and Dynamic Check Arguments
-pa_checkseq=	“<t1> [<unit>] [...<tn>]”
-pa_connectpgpin	i a e
-pa_corrupt= -pa_nocorrupt=	real integer
-pa_dbgstatic	msh rsn
-pa_defertop	<i>none</i>
-pa_dbgstatic	msh rsn
-pa_disable=	Using -pa_enable and -pa_disable
-pa_dumplibertydb=	<database_path>
-pa_dumpupf	<filename>
-pa_enable=	Using -pa_enable and -pa_disable
-pa_excludedelayedbuffer	<i>none</i>
-pa_excludefile	<filename>
-pa_genrpt=	[{nv v}] [{us ud u}] [b] [pa] [de]
-pa_gls	<i>none</i> (required for purely gate-level designs)
-pa_hiersep	<alphanum_character>
-pa_intcrptval0	<i>none</i>
-pa_lib	<library_pathname>
-pa_libertyrefresh=	<database_path>
-pa_libertyupdate	

Table A-1. Power Aware Arguments for vopt (cont.)

vopt Argument	Argument Value(s)
-pa_lsthreshold	<real>
-pa_loadlibertydb=	<database_path>
-pa_modeltype=	2
-pa_nopcfctrlcheck	<i>none</i>
-pa_out	<i>none</i>
-pa_prefix	<hier_path>
-pa_replacetop	<string>
-pa_reportdir	<pathname>
-pa_reportfile	<filename>
-pa_rtlinfo	<i>none</i>
-pa_tclfile	<filename>
-pa_top	<pathname>
-pa_upf	<filename>
-pa_upfextensions	Supported UPF Extensions
-pa_upfsyntaxchecks	<i>none</i>
-pa_upfversion=	1.0 2.0(default)

Table A-2. Power Aware Arguments for vsim

vsim Argument	Argument Value
-pa	<i>none</i>
-pa_allowtimezeroevent=	all
-pa_debugdir	<directory>
-pa_disabletimezeroevent	<i>none</i>
-pa_gls	<i>none</i> (previously used for purely gate-level designs. You can now use -pa option instead.)
-pa_highlight	<i>none</i>
-pa_lib	<pathname>
-pa_togglelimit=	<integer>
-pa_top	<pathname>

Table A-2. Power Aware Arguments for vsim (cont.)

vsim Argument	Argument Value
-pa_zcorrupt	<i>none</i>

Using -pa_enable and -pa_disable

The values for the -pa_enable and -pa_disable arguments to the `vopt` command listed in [Table A-3](#) allow you to enable or disable certain actions that performed during Power Aware simulation.

Syntax

```
vopt -pa_enable=<value>[+<value>...]
```

```
vopt -pa_disable=<value>[+<value>...]
```

Description

- Each argument uses the same set of values, which means using a value with either argument toggles that action from its default state or from a previously specified state.
- [Table A-3](#) lists each value, a brief description of the Power Aware action performed, and which argument used for that value by default.
- You can specify one or more values for either argument—there is no order dependency when specifying multiple values. To specify more than one value for either argument, use the + operator between values. For example:

```
vopt -pa_enable=lowerboundary+insertiso
vopt -pa_disable=sourcesink+relatedsupplies
```

Table A-3. Power Aware Actions for vopt -pa_enable and -pa_disable

Value	Action	Default
ackportbehavior	For UPF 2.0, power switches with the -ack_port switch require a supply set. <ul style="list-style-type: none"> • pa_enable — Issues an error if this condition is not met. • pa_disable — allows violation of this condition and uses an always on supply set. Essentially reverting to UPF 1.0 functionality 	-pa_enable
anonupfobjects	Create anonymous supply sets or nets for future replacement by associated objects.	-pa_disable

Table A-3. Power Aware Actions for vopt -pa_enable and -pa_disable (cont.)

Value	Action	Default
defaultoff	Changes the default state of supply nets and ports to either OFF or FULL_ON <ul style="list-style-type: none"> • -pa_disable=defaultoff sets default to FULL_ON • -pa_enable=defaultoff sets default to OFF 	-pa_enable
detectiso	Detect isolation cells present in the design.	-pa_enable
detectret	Detect retention cells present in the design.	-pa_enable
detectls	Detect level-shifter cells present in the design.	-pa_enable
detectsw	Detect switches instantiated in the design and their association with a UPF create_power_switch command.	-pa_disable
highlight	Enables highlighting in the Wave window	-pa_disable
ignorepecialdrivers ignoregroundsupplyconn ignorehangingoutput ignorepowersupplyconn ignoretiehigh ignoretielow ignoreundriveninput	If you enable any of these options, the tool disables the automatic insertion of level shifter and isolation cells based on the following scenarios. It also disables any static level shifter, isolation checking or dynamic checks (missing level-shifter or missing isolation): <ul style="list-style-type: none"> • All of the following scenarios • Ports connected to ground supplies • Unconnected output ports • Ports connected to power supplies • Ports tied to 0 • Ports tied to 1 • Undriven input ports 	-pa_disable
insertiso	Insert isolation cells.	-pa_enable
insertret	Controls retention cell insertion. <ul style="list-style-type: none"> • -pa_enable=insertret enables retention cell insertion in the gate-level flow (-pa_gls). • -pa_disable=insertret disables retention cell insertion in the RTL or mixed RTL-gate-level flows. 	-pa_enable
insertls	Insert level-shifter cells.	-pa_enable

Table A-3. Power Aware Actions for vopt -pa_enable and -pa_disable (cont.)

Value	Action	Default
lowerboundary	Perform isolation on the ports present in lower boundary of power domain.	-pa_enable
modsrns	Enables or disables support of the non-LRM UPF command set_related_supply_net .	-pa_enable
nonoptimizedflow	Must be enabled if you use -novopt on the vopt command line	-pa_disable
relatedsupplies	Define related supplies attributes on the boundary ports (accessible pins of hard macros).	-pa_enable
sourcesink	Apply pathwise analysis of isolation or level-shifting.	-pa_enable
supplynetworkdebug	Enables or disables the appearance of UPF objects created in the Structure, Object, and Wave windows.	-pa_disable
udpnoret	Enables or disables the retention of sequential UDPs in a gate-level or mixed RTL-gate-level power aware simulation. This does not impact the default behavior of UDP corruption. Refer to the section “UDP Corruption and Retention Modes” for more information.	-pa_disable
undeterminedstate	Controls behavior related to power aware switches or supply nets/ports going into an UNDETERMINED state as defined by UPF 2.0. Refer to the section “Detailed Support for supply_net_type” for more information <ul style="list-style-type: none"> • pa_enable — enables UPF 2.0 behavior • pa_disable — reverts to UPF 1.0 behavior 	-pa_enable
verbosereporting	Toggles the addition of a full hierarchical path of the power domain for static and dynamic reporting.	-pa_disable

Additional Commands You Can Use with Power Aware

Power Aware simulation supports the following additional Tcl commands, which are not UPF commands, that you specify in either a UPF file (vopt -pa_upf) or a Tcl file (vopt -pa_tclfile).

- `set_corruption_extent`
- `set_feedthrough_object`
- `set_related_supply_net`

set_corruption_extent

Syntax

```
set_corruption_extent -domains {<domain_name> ...} -ce {o | os | osw | sc | scb}
```

Arguments

- `domain_name` — the name of any power domain in the current scope, You can specify multiple domain names in a space-separated list, enclosed in braces [{}].
- `-ce` — sets the corruption extent, which takes one of the following values:
 - `o` — outputs only.
 - `os` — outputs and sequential elements.
 - `osw` — outputs and sequential and non-sequential wires.
 - `sc` — sequential and combination logic (based on UPF corruption semantics honoring all sequential and combination logic for corruption—excluding any buffers in the path for corruption).
 - `scb` — sequential, combination, and buffer logic.

Description

Specify in either a UPF file (`vopt -pa_upf`) or a Tcl file (`vopt -pa_tclfile`).

Changes the corruption extent of the power domains created by the UPF file.

Example

Change the corruption semantics of domains P1 and P2 created in the scope `tb` to outputs only.

```
set_scope tb
set_corruption_extent -domains {P1 P2} -ce o
```

set_feedthrough_object

Syntax

```
set_feedthrough_object -function <function_list> [-package <package_name> ]
```

Arguments

- -function <function_list> — a required list of one or more function names (you must specify at least one function name).
- -package <package_name> — detects only functions from the specified package, package_name. Optional.

Description

Allows conversion functions to be treated as feedthroughs for UPF-based corruption. The objective is to detect conversion functions in the PA-RTL and treat them as feedthrough paths.

Specify in either a UPF file (vopt -pa_upf) or a Tcl file (vopt -pa_tclfile).

set_related_supply_net

Syntax

```
set_related_supply_net -object_list <objects> -reset -power <power_net_name>  
-ground <ground_net_name>
```

Arguments

- `-object_list <objects>` — List of ports or pins that is to have a related power or ground supply defined. Pins or ports are referenced relative to the active scope.
- `-power <power_net_name>` — The related supply power net, referenced relative to the active scope. You can specify this switch by itself or in combination with `-ground`.
- `-ground <ground_net_name>` — The related supply ground net, referenced relative to the active scope. You can specify this switch by itself or in combination with `-power`.
- `-reset` — not currently supported.

Description

Allows associating an instance signal pin or a hierarchical port with specific supply nets. Thus, you can create a supply net based on supply pins so you can specify your related supplies of cells.

Specify with `vopt -pa_upfextensions` (see [Using -pa_upfextensions](#)).

Power Aware simulation internally maps this command to the `set_port_attributes` command that you specify in the UPF file, specifically the following arguments:

```
-related_power_pin <power_net_name>  
-related_ground_pin <ground_net_name>
```

Note that v2.0 of the UPF standard interprets the **related_power_net** and **related_ground_net** attributes as defining the driver supply set of an output port or the receiver supply set of an input port. The standard also declares that it is an error in the following cases:

- if the actual driving logic is present and its supply is not the same as the driver supply.
- if the actual receiving logic is present and its supply is not the same as the receiver supply.

As a result, use of **set_related_supply_net** for any purpose other than specifying the driver supply set of a macro model output or the receiver supply set of a macro model input may generate errors.

Specifically, Power Aware simulation gives a `vopt` error message (`vopt-9814`). You can use the `-warning` argument of `vopt` to change the severity of this message to a warning so that simulation may continue:

```
vopt -warning 9814
```

i Refer to [Power Aware Messages](#) for more information on changing the level of message severity.

Default Behavior

The functionality in the section is enabled by default.

For top-level ports (in the absence of boundary information), it is left up to the simulator to decide how to handle the primary input and output ports.

You can define the supply related to the primary input and output ports, and instruct the simulator as to what voltage the input ports can be driven and what voltage the output ports can drive.

If you use the `-source`, `-sink`, or `-diff_supply_only` options for setting isolation strategies, the related supply tells the simulator what supply is powering the cell on the other side of the domain boundary, which can then be used to determine whether or not isolation is necessary based upon your constraints. This related supply information is useful for static analysis.

This command relates the specified power and/or ground net to the port or pin specified.

When you use `set_related_supply_net` on a primary port:

- Input port assumes the supply net specified is the driver of the input port.
- Output port assumes the supply net specified is the receiver of the output port.

That is, the command specifies the external supplies. In other words a buffer insertion takes place at the location parent.

- For input port — Buffer --> Input port
- For output port — Output Port --> Buffer

Static analysis honors this information of related supplies.

As buffers are now inserted in design just like isolation, when supply of buffer goes off, then buffer output gets corrupted.

For this case, when isolation is also applied on the port at which `srns` is defined:

- For input ports — buffer is applied at location parent (followed by isolation cells).
Buffer -- ISO -- Input port
- For output ports — buffer is applied at location parent (preceded by isolation cells)
Output Port -- ISO -- Buffer

When buffer (`set_related_supply_net supply`) goes off:

- Input Port — Corruption is seen on the input port.
- Output Port — Corruption is seen on the logic (or port) driven by output port.

Power Aware Messages

This section describes the ways in which you can control messaging within the ModelSim power aware functionality.

Dynamic Power Aware Check Message Control

Use the `pa msg` command to finely control the appearance of messages resulting from power aware checks during simulation.

Note



When you disable a message for a specific power aware check, you also disable the underlying assertion as well. This means that the any messages are disabled as well as any resulting coverage collection.

The syntax for the command is:

```
pa msg [-enable | -disable] [-severity {note | warning | error | fatal}] [-stopafter <number>]
[message_number ...] [-pa_checks[=<spec>]] [-scope <scope_name>] [-domain <pd_name>]
[-strategy <strategy_name>] [-ports <ports_list>] [-glitch_window <duration>] [-all]
```

Concepts of `pa msg`

- **Command scope (-scope)** — When you do not specify the `-scope` argument, the scope of the command is set to the design top, specifically it applies to the entire design (from design top scope downwards). Use of this option changes the scope of command to the specified scope.

For example if `tb` is the design top (the scope from which `vsim` was invoked) then:

```
pa msg
```

with no arguments, applies to all objects from the `/tb` scope. While:

```
pa msg -scope /tb/dut
```

applies to all objects from the `/tb/dut` scope.

The `pa msg` command also applies to all objects from the scope of the command. It also applies to all domains and all relevant objects from scope downwards, specifically:

- If you specify `-domain`, the command is restricted to apply only to all instances within that domain.
- If you specify `-strategy` or `-ports`, the command applies only to those objects with in the scope (or domain).
- Any names specified with `-domain` or `-ports` should be relative to the scope of the command.

Other examples show you how to apply the command to all objects of the power domain `/tb/dut/mid1/PD_mid1`.

```
pa msg -scope /tb/dut/mid1 -domain PD_mid1
```

Also, when you do not specify `-scope`, names associated with `-ports` or `-domain` are relative to design top.

```
pa msg -domain dut/mid1PD_dut ... -ports {dut/mid1/P1 dut/mid1/P2}
```

2. **Use of `-all`** — When you specify `-all`, the command applies to all dynamic power aware check messages.

```
pa msg <options> -all
```

Otherwise, when you do not specify `-all`, it applies to all messages except the ones with severity NOTE.

```
pa msg <options>
```

- **Disabling checks also disables assertions** — When you disable a particular power aware check, the underlying assertion for that check is also disabled, resulting the following:
 - disabling the messages of selected checks.
 - disabling the associated coverage collection of that assertion
- Power aware checks are disabled at time 0 (zero) — By default, all dynamic power aware checks are disabled at time 0, resulting in no messages at this time. You can change this behavior with `pa msg` before running your simulation, for example, if you want to enable Power Status Info (`-pa_checks=p`) at time 0, do the following:

```
vsim > pa msg -enable -pa_checks=p
```

```
vsim > run 1ns
```

Use Model for `pa msg`

You can use this command at any time during simulation. It will have impact in simulation after the time stamp of its usage until the time any new `pa msg` command overrides the previous one, for example

In the following example, the power aware check **umi** (missing iso) will be disabled after 100ns and it will be re-enabled at 150ns. Therefore, in the time window from 100ns to 150ns this pa check will be disabled.

```
run 100ns
```

```
pa msg -pa_checks=umi -disable
```

```
run 50ns
```

```
pa msg -pa_checks=umi -enable
```

Option Descriptions of `pa msg`

The options to the `pa msg` command are organized, broadly, into three categories:

- Selecting power aware checks.
- Taking finer control over selected power aware checks.
- Performing actions on selected power aware checks.

Power Aware Check Selection

Note



If you do not specify any of these options, the command selects all dynamic power aware checks, except the ones with severity NOTE.

- **message numbers** — These numbers are found in the violation message of the power aware check. The numbers you specify select the associated power aware checks. For example:

```
pa msg 8920 8931
```

selects the power aware checks associated with message numbers 8920 and 8931.

- **-pa_checks[=<spec>]** — This option is the same as that specified with the **vopt** command. For example,

```
pa msg -pa_checks=irc+upc
```

selects the “isolation race” and “switching off of isolation and retention supply” checks.

- **-all** — This option selects all dynamic PA Checks.

Fine Control of Check Selection

- **-scope <scope_name>** — This option defines the scope of the command, where the scope is restricted to apply from this scope downward. By default, the scope is the design top scope.

Note



This option is meaningful when you specify it with `-domain`, `-strategy`, or `-ports`.

The scope is dependent on how you specify **scope_name**:

- If you specify an absolute path name, starting the argument with a slash (/), then the scope is becomes your explicit argument.

```
pa msg ... -scope /tb/TOP/mid1
```

- If you do not specify an absolute path name, not starting with a slash (/) then scope becomes: <design_scope>/<scope_name>. For example, assuming the design top scope is **tb**:

```
pa msg ... -scope TOP/mid1
```

the scope of the command will be **/tb/TOP/mid1**.

- **-strategy <strategy_name>** — This option selects a power aware check belonging to the specified strategy. If you use this option with **-domain**, then checks belonging to **domain.strategy** will be selected.

For example, the following command will select the “Isolation functionality” check (associated with message 8931) for isolation strategy **iso_top**:

```
pa msg 8931 -strategy iso_top
```

The following command will select all isolation checks which belong to strategy **iso_mid1** of power domain **<Scope Of Command>/PD_mid1**:

```
pa msg -pa_checks=i -domain PD_mid1 -strategy iso_mid1
```

This option will be ignored if you use it to finely control checks that do not belong to any strategy. For example, the following checks do not have any UPF strategy

- Power domain status check "-pa_checks=p", which refers to message 8902
- Power domain toggle check "-pa_checks=t", which refers to message 8908
- Missing level shifter check "-pa_checks=uml", which refers to message 8915
- Missing isolation check "-pa_checks=umi", which refers to message 8929
- Non-retention register reset check "-pa_checks=npu", which refers to message 8912

Note



It the specified strategy name does not exist tool will flag a NOTE message as shown below and will ignore this option.

```
# ** Warning: (vsim-4086) Strategy 'iso_PD_mid3' specified with '-strategy' option not found
```

- **-ports <list of ports>** — This option restricts your selection power aware checks for specified ports only, where the names of ports should be relative to Scope of Command.

For example, to select Isolation clamp value check for Ports **/tb/TOP/mid1/P1** and **/tb/TOP/mid1/P2**, which belong to strategy **iso_mid1** of power domain **PD1**, use the command:

```
pa msg -pa_checks=icp -domain PD1 -strategy iso_mid1 -scope /tb/TOP/mid1 -ports {P1 P2}
```

This option is applicable only to the following checks:

- Isolation race check "-pa_checks=irc" which refers to message 8910
- Isolation functionality check "-pa_checks=ifc" which refers to message 8931
- Isolation clamp value check "-pa_checks=icp" which refers to message 8930
- Isolation toggle check "-pa_checks=it" which refers to message 8908

If you specify this option with checks other than these, you'll receive a NOTE message and the option will be ignored. For example, the command:

```
pa msg 8918 -domain PD1 -strategy iso_mid1 -scope /tb/TOP/mid1 -ports {P1 P2}
```

Issues the error:

```
** Note: (vsim-4087) The command 'pa msg' is ignored for message
number 8918, as option '-ports' is not applicable for this message
number.
```

because **-ports** is not applicable with message 8918.

Actions for Power Aware Checks

Use the following options to specify the action to be taken on the selected power aware checks.

- **-enable or -disable** — Use these options to enable or disable a particular power aware check which was selected using the selection and fine control options.
- **-severity <note | warning | error | fatal >** — Use this option to change default severity of a selected check.
- **-stopafter <number>** — Use this option to specify a maximum count. If a selected check exceeds this specified count, then simulation will stop.
- **-glitch_window <duration>** — Use this option to specify the maximum allowed time window of a glitch. It is applicable for glitch checks (-pa_checks=ugc) only. If specified with any other checks it will be ignored.

Examples

The following commands show several usage examples of the pa msg command.

- Disable all isolation functionality and clamp value checks for isolation strategy **iso_mid** belonging to power domain **/tb/TOP/mid1/PD_mid**:

```
pa msg -disable -pa_checks=ifc+icp -scope /tb/TOP/mid1 -domain PD_mid
-strategy iso_mid
```

In this command, if action is to be taken only for Port **/tb/TOP/mid1/P3** then you would specify:

```
pa msg -disable -pa_checks=ifc+icp -scope /tb/TOP/mid1 -domain PD_mid
-strategy iso_mid -ports {P3}
```

- Disable all power aware checks belonging to strategy **iso_mid2**:

```
pa msg -all -strategy iso_mid2
```

It applies to all power aware checks which belong to the **iso_mid2** strategy only. If you want to control **iso_mid2** of **/tb/TOP/PD_top** only then you should specify:

```
pa msg -all -domain TOP/PD_top -strategy iso_mid2
```

or

```
pa msg -all -scope /tb/TOP -domain PD_top -strategy iso_mid2
```

- Enable all isolation checks at time 0.

Before running simulation specify:

```
pa msg -enable -pa_checks=i
```

After this command, and when simulation has run, all failing isolation checks, even those at time 0, will be flagged.

Controlling Power Aware Message Severity During vopt Stage

You can use the following **vopt** arguments to suppress or control the severity of messages that occur while running Power Aware:

- **-suppress <msg_num>** — Suppresses a particular message by its ID number (**msg_num**).
Messages are not displayed and processing continues.
- **-warning <msg_num>** — Changes the severity of a particular message to Warning.
Messages are displayed and processing continues.
- **-error <msg_num>** — Changes the severity of a particular message to Error.
Messages are displayed and processing continues.
- **-note <msg_num>** — Changes the severity of a particular message to Note.
Messages are displayed and processing continues.
- **-fatal <msg_num>** — Changes the severity of a particular message to Fatal.
Messages are displayed and processing stops.

Applying different severity levels allows you to prevent stoppage of **vopt** operation or inconsistent behavior in processing and resolve phases when the error is encountered. An

example of this is when vopt stops for errors in the processing phase, while continuing when the same errors occur in the resolve phase.



When you suppress or lower the severity of an error message, you may not see the desired result, since the corresponding UPF command behavior gets bypassed or ignored.

Examples

- Use vopt to suppress or change the severity of a single message:

```
vopt -pa_upf top.upf -o t tb -suppress <msg_num>
```

- Use vopt to suppress or change the severity of multiple messages:

```
vopt -pa_upf top.upf -o t tb -warning <msg_num1>,<msg_num2>,  
<msg_num3>
```

Excluding Design Elements from Power Aware

You can exclude Power Aware processing for any module, instance, or signal within a particular hierarchical path in a design.

Usage

```
vopt -pa_excludefile <filename>
```

Description

The `-pa_excludefile` option instructs Power Aware processing to skip any module listed in the specified exclude file.

- `<filename>` — The name of a text file that specifies modules, instances, or signals you want to exclude from Power Aware verification.

Entries for modules must be of the following form:

```
<module_name> [-a] <hier_path>
```

`module_name` — any regular expression (enclosed in quotation marks).

`-a` — an optional switch that enables recursive exclusion of `module_name`.

`hier_path` — the full pathname to the instance of the module you want to exclude. When a module instance is skipped, Power Aware simulation displays the following message:

```
** Note: (vopt-9691) Excluding power aware module '<module_name>'  
in path '<hier_path>'.
```

When used with `-a`, specifying `hier_path` limits the recursive exclusion to a particular scope.

Entries for instances or signals must be of the following form:

```
<module_name> [<instance_pathname>] [[-s | -r] <signal_name> ...]
```

-s — specifies signal exclusion

-r — specifies recursion (applied to s, where all occurrences inside only that module scope will be excluded).

Examples—Excluding a Module

The following examples show entries in the exclude file.

1. Direct Power Aware processing to skip all instantiation of **bot_mod** found within the top hierarchy.

- Entry in the exclude file:

```
bot_mod
```

- Log messages for the vopt command:

```
** Note: (vopt-9691) Excluding power aware module 'bot_mod' in path
'/top/t1'.
```

```
** Note: (vopt-9691) Excluding power aware module 'bot_mod' in path
'/top/t1/t2'.
```

2. Direct Power Aware processing to skip all instantiation of **bot_mod2** and **bot_mod3** found within the top hierarchy.

- Entry in the exclude file:

```
"bot_mod[2-3]" /top/mid
```

- Log messages for the vopt command:

```
** Note: (vopt-9691) Excluding power aware module 'bot_mod2' in path
'/top/inst1'.
```

```
** Note: (vopt-9691) Excluding power aware module 'bot_mod3' in path
'/top/inst2'.
```

Examples—Excluding Signals/Nets

The following examples show entries in the exclude file.

1. Direct Power Aware processing to skip signal/net **sig**, present in all instantiations of **bot_mod** found within the instance **/top/t1**.

- Entry in the exclude file:

```
bot_mod /top/t1 -s sig
```

- Log messages for the vopt command:

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/sig'.
```

A warning message will be reported if there was no match found in exclude file, for signal **sig** in module **bot_mod** in instance **/top/t1**:

```
** Warning: exclude.txt(1): (vopt-9014) No match found in exclude file, for signal 'sig' in module 'bot_mod' in instance path '/top/t1'.
```

2. Direct Power Aware processing to skip signals or nets **sig2**, **sig3**, **sig4**, present in all instantiation of **bot_mod** found within the instance **/top/t1**.

- Entry in the exclude file:

```
bot_mod /top/t1 -s "sig[2-4]"
```

- Log messages for the vopt command:

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig3' in power aware module 'bot_mod' in path '/top/t1/sig3'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig4' in power aware module 'bot_mod' in path '/top/t1/sig4'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig2' in power aware module 'bot_mod' in path '/top/t1/sig2'.
```

3. Direct Power Aware processing to skip signal or net, **sig**, present in the scope of all instantiations of **bot_mod** found within the instance **/top/t1**.

- Entry in the exclude file:

```
bot_mod /top/t1 -sr sig
```

- Log messages for the vopt command:

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/fg__5/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/fg__4/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/fg__3/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/fg__2/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/fg__1/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/blk/sig'.
```

```
** Note: exclude.txt(1): (vopt-9013) Excluding signal 'sig' in power aware module 'bot_mod' in path '/top/t1/sig'.
```

Voltage Level-Shifting (Multi-Voltage Analysis)

This section describes the voltage level-shifting capability of Power Aware, which is primarily implemented as Unified Power Format (UPF) commands and Power Aware arguments to the ModelSim vopt command.

The supply network state provides information about the possible power states of the network. Power Aware simulation uses that information to detect level shifters wherever a signal crosses from a power domain operating at a voltage level that may be different than the voltage level of another power domain to which it connects (also known as multi-voltage analysis).

Power State Tables

Power Aware simulation uses information from a Power State Table (PST) in Power Aware analysis. PSTs are also parsed and dumped to the UPF report file (report.upf.txt). It is assumed that the PST is complete; any domains that are not mentioned in PST will not be used for analysis.

The traversal does not skip any power switches encountered in the supply network path. The traversal goes only behind direct connectivity of supply ports and supply nets that are created in UPF. It does not go behind a supply net present in the design or the UPF supply net/port that is directly connected to an HDL supply net or port.

Example

```
Pst top_pst, File:../UPF/rtl_top.upf(127).
Header ==>                : VDD_0d99 VDD_0d81 VSS
ON  ../UPF/rtl_top.upf(133): ON      ON      ON
OFF ../UPF/rtl_top.upf(134): ON      ON      ON

List of possible states on:
VDD_0d99 [ source supply port: VDD_0d99, File:../UPF/rtl_top.upf(21)]
1. ON: 0.99,1.10,1.21

VDD_0d81 [ source supply port: VDD_0d81, File:../UPF/rtl_top.upf(22)]
1. ON: 0.81,0.90,0.99

VSS [ source supply port: VSS, File:../UPF/rtl_top.upf(23)]
1. ON: 0.00,0.00,0.00
```

Level Shifter Specification

Reporting

Power Aware simulation parses the **set_level_shifter** command in the UPF file and selects a list of candidate ports for level shifter insertion. These ports are also dumped into the UPF report file (report.upf.txt) as in the following example:

```
Level Shifter Strategy: my_ls, File: ./src/simple_mv7/test.upf(63).
Rule (high_to_low), Threshold (0), Applies_to (outputs).
Level Shifted Candidate Ports:
  1. Signal : /tb/TOP/bot2/out1_bot

Level Shifter Strategy: my_ls_bot3, File: ./src/simple_mv7/test.upf(69).
Rule (low_to_high), Threshold (0), Applies_to (outputs).
Level Shifted Candidate Ports:
  1. Signal : /tb/TOP/bot3/out1_bot
```

Threshold Control for Level Shifters

You can set a global threshold level for a Power Aware analysis containing multiple voltage levels using the following command:

```
vopt -pa_lsthreshold <real>
```

where <real> is any numerical value that specifies a voltage threshold.

Use this argument with the vopt command when you know that level shifting is not required for particular range of voltage differences. You can then specify a global threshold— otherwise Power Aware simulation will flag missing level shifter errors even if the potential difference between two domains is within an acceptable range.

Level Shifter Instances

If you have used the **set_level_shifter** -instance command in a UPF file to instantiate level shifters, Power Aware simulation will detect those instances and perform level shifting checks on them (see [Static Checking in Power Aware](#)).

An instance is recognized as a level shifter instance in any of the following cases:

- The level shifter instance is specified with **-instance** argument of **set_level_shifter** command.
- For a gate-level design, any of the following:
 - The **is_level_shifter** attribute is specified for the module. Example:

```
(* is_level_shifter = 1 *)
module ls_buf(
    (*pg_type = "primary_power"*) input logic pwr_rail,
```

```

(*pg_type = "primary_ground"*)input logic gnd_rail,
(* level_shifter_data_pin = 1 *)input data,
output logic out);
assign out = (data);
endmodule

```

- o Instantiation has prefix or suffix string for level shifter specified with **name_format** UPF command. Example:

```
LVLHLD1BWP lsinst2_UPF_LS(.I(w2), .Z(w4));
```

Limitations on Level Shifting

- Support for VHDL and Verilog synthesizable data types. Restricted support for SystemVerilog (array, struct).
- Level shifters not specified at the power-domain boundary are not considered for multi-voltage checks.

Restricting Isolation and Level Shifting on a Port

The [set_isolation](#) and [set_level_shifter](#) UPF commands each have -source and -sink arguments, which you can use to apply isolation or level shifting only to certain paths of a specific port in your design. When you specify either or both of these arguments, Power Aware simulation identifies all the paths through the given port and applies isolation or level shifting to only those paths whose driver and receiver supplies match the specified source and sink supplies.

Isolation and Level Shifting Behavior

Using the -source and -sink arguments affects Isolation and Level Shifter insertion behavior in the following ways:

- Determine all the paths passing through a given port.
To determine the path, all the buffers, isolation cells, and level shifter cells are treated as feedthroughs and actual drivers and receivers are determined.
- Insert isolation or level shifter cells after matching source or sink supplies, if specified.
To match equivalent supplies, the driver nets of primary power and ground nets are matched.
- Determine, via the -location argument (for both commands), the placement of an isolation or level shifter cell.

You can specify any of the following values for **set_isolation -location** or **set_level_shifter -location**:

- Fanout — isolation or level shifter cell is placed at all fanout locations (sinks) of the port.
- Fanin — isolation or level shifter cell is placed at all fanin locations (sources) of the port.
- Faninout — isolation or level shifter cell is placed at all fanout locations (sinks) for each output port, or at all fanin locations (sources) for each input port.
- Parent — isolation or level shifter cell is placed in the parent of the domain whose interface port is being isolated or shifted.
- Automatic — same as Parent.
- Self — isolation or level shifter cell is placed inside the domain whose interface port is being isolated or shifted.
- Sibling — same as Self.

Note



Level shifter cells are not currently inserted in RTL — their effect will not be present in simulation. Only Power Aware checking (vopt -pa_checks) will validate these cells.

How to Apply the -source and -sink Arguments

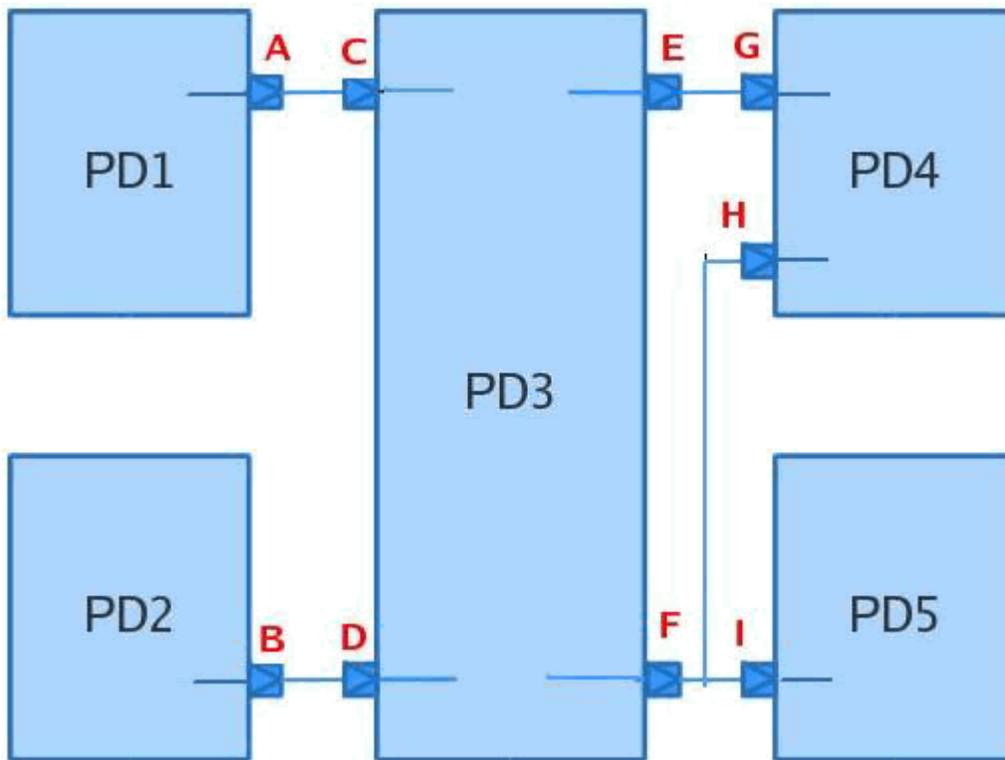
Figure A-1 shows a block diagram of power domains, ports, and paths for use in the examples that follow.

Example A-1. UPF Commands That Define Power Domains

The following list shows fragments of UPF commands used to define the diagram of Figure A-1:

```
create_supply_set PD1_SS ...
create_power_domain PD1 ...
associate_supply_set PD1_SS -handle PD1.primary
create_supply_set PD2_SS ...
create_power_domain PD2 ...
associate_supply_set PD2_SS -handle PD2.primary
create_power_domain PD3 ...
```

Figure A-1. Supply Paths to Power Domains



Source Examples

- Place isolation cell at Port C and isolate Path A-C.

```
set_isolation iso1 -domain PD3 -source PD1_SS -location parent ...
```
- Place isolation cell at Port B and isolate Path B-D.

```
set_isolation iso2 -domain PD3 -source PD2_SS -location fanin ...
```

Sink Examples

- Place isolation cells at Port G and Port H and isolate Path E-G and F-H.

```
set_isolation iso1 -domain PD3 -sink PD4_SS -location fanout ...
```
- Place isolation cell at Port H and isolate Path F-H.

```
set_isolation iso2 -domain PD3 -elements {F} -sink PD4_ss  
-location fanout ...
```
- Place isolation cell at Port F and isolate Path F-H.

```
set_isolation iso3 -domain PD3 -elements {F} -sink PD4_ss  
-location parent ...
```

Differential Supply Examples

You can prevent the application of isolation into a path from driver to receiver for isolation strategy by using the `-diff_supply_only` argument to the `The set_isolation` command.

For these examples, assume that **PD2**, **PD3** and **PD4** all have same supply sets:

```
create_supply_set PD2_SS ...
create_power_domain PD2 ...
associate_supply_set PD2_SS -handle PD2.primary
associate_supply_set PD2_SS -handle PD3.primary
associate_supply_set PD2_SS -handle PD4.primary
```

- Place isolation cells at Ports C and F and isolate Paths A-C and F-I. Do not isolate path F-H.

```
set_isolation iso1 -domain PD3 -applies_to both
-diff_supply_only TRUE -location parent ...
```

- Place isolation cells at Ports A and I and isolate Paths A-C and F-I.

```
set_isolation iso2 -domain PD3 -applies_to both
-diff_supply_only TRUE -location faninout ...
```

- Place isolation cell at Port A and isolate Path A-C.

```
set_isolation iso3 -domain PD3 -applies_to both -source PD1_SS
-diff_supply_only TRUE -location faninout ...
```

Multiple Strategies Example

- Isolate the same port F with different Sinks. Here, Port H will be isolated with `iso1` and port I with `iso2`.

```
set_isolation iso1 -domain PD3 -elements {F} -sink PD4_SS
-location fanout -clamp 1 ...
```

```
set_isolation iso2 -domain PD3 -elements {F} -sink PD5_SS
-location fanout -clamp 0 ...
```

Multiple Isolation Cells Examples

Refer to [Figure A-2](#) for the following examples on using multiple isolation cells.

- Using `-source` and `-location fanout` — Isolates all output port paths and places isolation cells at Ports G, H and I. Places two isolation cells at Port H and I for the same port F.

```
set_isolation iso1 -domain PD3 -source PD3_SS -location fanout ...
```

- Using `-source` and `-location parent` — Places isolation cells at Ports E and F. Places only one Isolation Cell at Port F, which isolates both Paths F-H and F-I.

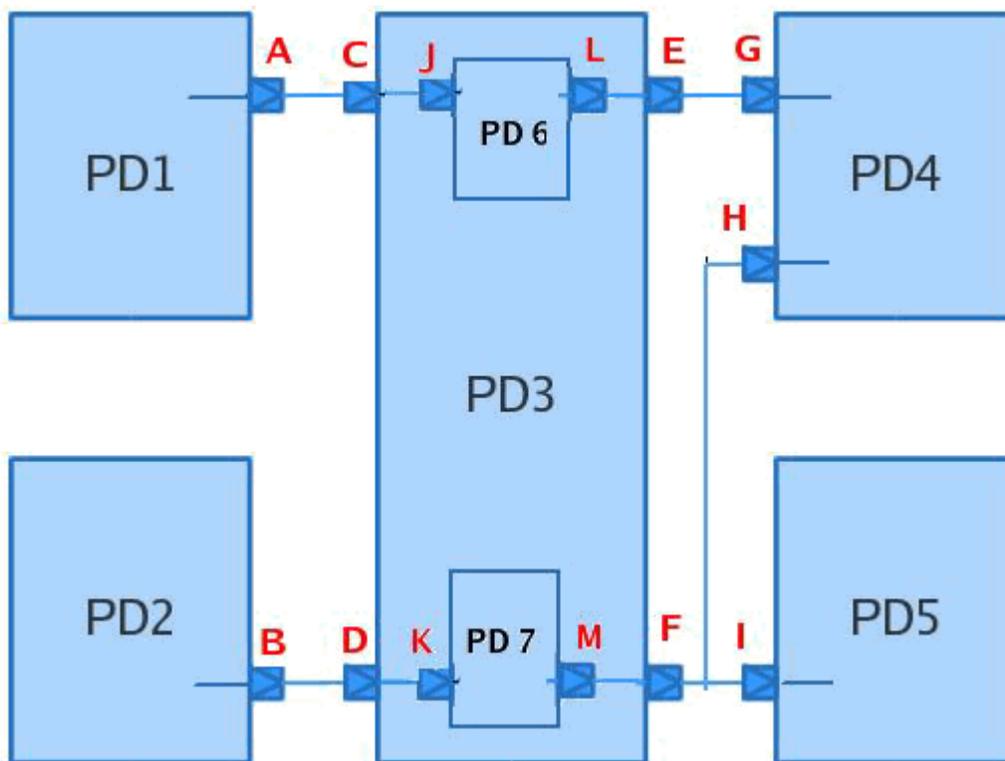
```
set_isolation iso2 -domain PD3 -source PD3_SS -location parent ...
```

- Places isolation at Port G. Relative ordering is maintained with `iso3` cell in front of `iso4` cell, means `Iso3 -> iso4 -> port G`.

```
set_isolation iso3 -domain PD3 -elements {E} -location fanout
-clamp 1 ...

set_isolation iso4 -domain PD4 -elements {G} -location parent
-clamp 0 ...
```

Figure A-2. Multiple Isolation Cells



Multiple Strategies in a Path Examples

Refer to [Figure A-2](#) for the following examples on using multiple strategies in a path.

- Places isolation cell at Port H and isolates path M-F-H.

```
set_isolation iso1 -domain PD7 -sink PD4 -location fanout ...
```

- Places isolation cell at Port M and isolates only path M-F-H. This means the clamp value will be seen at Port H. All other ports M, F and I will not have clamp value during power down.

```
set_isolation iso2 -domain PD7 -sink PD4 -location parent ...
```

- Places three isolation cells at Port H with relative ordering: iso3 > iso4 > iso5 > port H.

```
set_isolation iso3 -domain PD7 -sink PD4 -location fanout ...  
  
set_isolation iso4 -domain PD3 -elements {F} -sink PD4  
-location fanout ...  
  
set_isolation iso5 -domain PD4 -elements {H} -location parent ..
```

- Places two isolation cells at Port F and isolates only port H with relative ordering: Port F > iso6 > iso7. This means the clamp value (o/p of iso7) will be seen at Port H. All other ports (M, F and I) will not have clamp value during power down.

```
set_isolation iso6 -domain PD7 -sink PD4 -location fanout ...  
  
set_isolation iso7 -domain PD3 -elements {F} -sink PD4  
-location parent ...
```

Simulating Designs Containing Macromodels

A macromodel is a block-level model in a design that has been optimized for power, area, or timing and has been silicon-tested. Defining the power intent for a macromodel depends on whether you have access to its internal structure (logic and topology). If internal access is not available, you can specify power intent only on its external pins—this is referred to as a “hard macro.”

To specify power intent for a hard macro, you define related-supplies attributes on these accessible pins as boundary ports. Similarly, you can isolate a hard macro from the rest of the design by applying the isolation on its boundary ports.

Specifying power intent for a hard macro is available by any of the following methods:

- In a UPF file with the commands:
 - [set_port_attributes](#)
 - [set_pin_related_supply](#)
- Related-supplies attributes in RTL
- Using a Liberty file in GLS

Using UPF Commands

To specify hard macro power intent, use the UPF commands and their arguments described below.

Command

[set_port_attributes](#)

Arguments

- ***-receiver_supply***

This argument specifies the supply of the logic reading the port. If the receiving logic is not within the logic design starting at the design root, it is presumed the receiver supply is the supply for the receiving logic.

- ***-driver_supply***

This argument specifies the supply of the logic driving the port. If the driving logic is not within the logic design starting at the design root, it is presumed the driver supply is the supply for the driver logic and the port is corrupted when the driver supply is in a simstate other than NORMAL.

- ***-related_power_port***
-related_ground_port
-related_bias_ports

Any of these arguments create an implicit supply set containing the supply nets connected to the ports. The behavior differs, depending on the mode of the port being attributed:

in mode — the implicitly created supply set is treated as the *-receiver_supply* set.

out mode — the implicitly created supply set is treated as the *-driver_supply* set.

inout mode — the implicitly created supply set is treated as both the *-receiver_supply* and *-driver_supply* set.

- ***-receiver_supply***

If you use this argument and if the receiving logic is within the logic design starting at the design root, it shall be an error if its supply is not the receiver supply.

- ***-driver_supply***

If you use this argument and if the driver logic is within the logic design starting at the design root, it shall be an error if its supply is not the driver supply.

Command

[set_pin_related_supply](#)

This command defines the related power and ground pins for signal pins on a library cell. It conveys information similar to **related_power_pin** and **related_ground_pin** in Liberty, but it *may* override them. This command is restricted to only leaf-library cells and non-synthesizable hierarchical modules.

Attributes in RTL

You can define the following attributes in RTL to specify the power intent of hard macros:

- UPF_related_power_pin

- UPF_related_ground_pin

System Verilog Example

```
(* UPF_related_power_pin = "my_Vdd" *) output my_Logic_Port;
```

VHDL Example

```
attribute UPF_related_power_pin of my_Logic_Port : signal is "my_Vdd";  
(* UPF_related_power_pin = "my_Vdd" *) output my_Logic_Port;
```

Liberty File

You can define the following attributes at the pins in a Liberty file:

- related_power_pin
- related_ground_pin

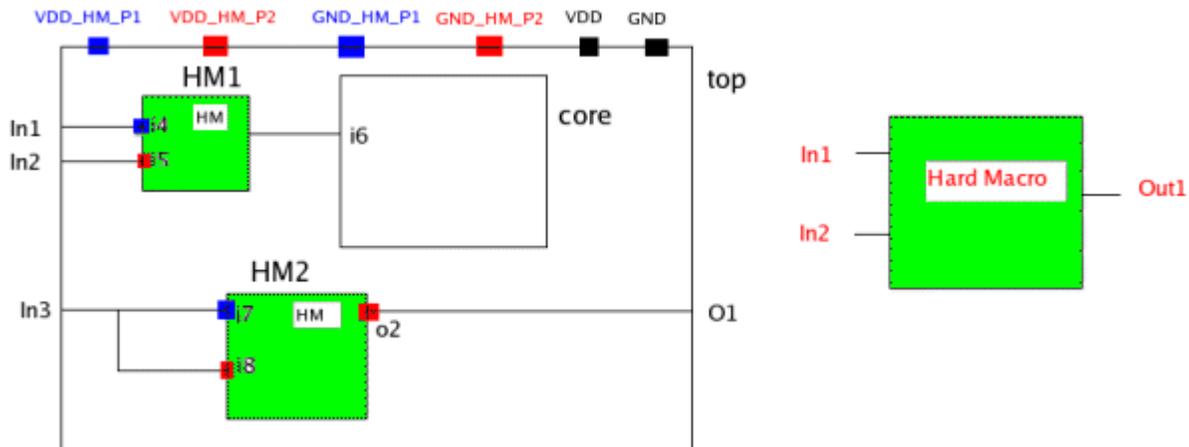
The related_power_pin and related_ground_pin attributes are defined at the pin level for output, input, and inout pins. These attributes associate a predefined power and ground pin with the corresponding signal pins under which they are defined. A default related_power_pin and related_ground_pin will always exist in any cell.

Example of Power Intent on a Hard Macro

Figure A-3 shows an example of a block diagram of hard macro power domains, using the following top-level UPF definition:

```
set_scope top  
  
create_power_domain PD_top -include_scope  
  
create_power_domain PD_HM1 -elements { HM1 }  
  
set_domain_supply_net PD_top -primary_power_net VDD  
-primary_ground_net GND  
  
set_domain_supply_net PD_HM1 -primary_power_net VDD_HM_P1  
-primary_ground_net GND_HM_P1
```

Figure A-3. Design Consisting of Hard Macros



The following sections show how to define the hard macro power intent for each method.

UPF Commands

Constraints are specified on pins.

```
set_pin_related_supply HM1 -pins { i4 } -related_power_pin VDD_HM_P1
    -related_ground_pin GND_HM_P1

set_port_attributes -ports { HM2/i7 } -related_power_port VDD_HM_P1
    -related_ground_port GND_HM_P1

set_port_attributes -ports { HM2/i8 HM2/o2 HM1/i5 } -related_power_port
    VDD_HM_P2 -related_ground_port GND_HM_P2
```

RTL Attributes

For HM1 —

```
(* UPF_related_power_pin = "VDD_HM_p1", UPF_related_ground_pin =
    "GND_HM_p1" *) input in4;

(* UPF_related_power_pin = "VDD_HM_p2", UPF_related_ground_pin =
    "GND_HM_p2" *) input in5;
```

For HM2 —

```
(* UPF_related_power_pin = "VDD_HM_p1", UPF_related_ground_pin =
    "GND_HM_p1" *) input in7;

(* UPF_related_power_pin = "VDD_HM_p2", UPF_related_ground_pin =
    "GND_HM_p2" *) input in8;

(* UPF_related_power_pin = "VDD_HM_p2", UPF_related_ground_pin =
    "GND_HM_p2" *) output o2;
```

Liberty File Attributes

```
library (PALIB) {
  cell (HM) {
    pg_pin (VDD_HM_P1) {
      pg_type : primary_power;
      user_pg_type : "abc";
      voltage_name : COREVDD1;
    }

    pg_pin (GND_HM_P1) {
      pg_type : primary_ground;
      voltage_name : COREGND1;
    }

    pg_pin (VDD_HM_P2) {
      pg_type : backup_power;
      user_pg_type : "abc";
      voltage_name : COREVDD1;
    }

    pg_pin (GND_HM_P2) {
      pg_type : backup_ground;
      voltage_name : COREGND1;
    }

    pin(in1) {
      direction : input;
      related_ground_pin : GND_HM_P1;
      related_power_pin : VDD_HM_P1;
    }

    pin(in2) {
      direction : input;
      related_ground_pin : GND_HM_P2;
      related_power_pin : VDD_HM_P2;
    }

    pin(out1) {
      direction : input;
      related_ground_pin : GND_HM_P2;
      related_power_pin : VDD_HM_P2;
    }
  }
}
```

Creating Feedthrough For RTL Conversion Functions

In RTL logic, a function call normally creates a driver in the design, to which Power Aware will attempt to apply the specified power intent. However, functions that are intended only to

convert or assign data types should not be considered as part of the power intent—they do not require isolation, retention, or corruption.

You can create a Tcl file to identify such functions in your design so that they are excluded from your power intent. These are referred to as a “feedthrough” functions. In the Tcl file, you use the **set_feedthrough_object** command for each function you want to exclude.

Syntax

```
set_feedthrough_object -function {function_list} [-package package_name]
```

When you run Power Aware, use `vopt -pa_tclfile` to specify the name of this Tcl file.

- `-function {function_list}` — a list of function name. This is a mandatory option and at least one function name should be specified.
- `-package package_name` — (optional) detects functions from the specified package only.

Appendix B

Model Construction for Power Aware Simulation

Power Aware verification uses Verilog behavioral models of Power Aware cells. These models encapsulate the Power Aware behaviors of various types of design state, such as clock-low retention flip-flops and active-high retention latches.

Verilog HDL constructs and attributes for Power Aware models provided by a silicon (or library) vendor. These models trigger relevant events for the simulator to modify the runtime behavior of the design. Typically, these modifications consist of corrupting states and output values and storing or restoring state values based on power control network activity.

Typically, your UPF power specification file relates inferred registers or latches to these models. However, it is possible to capture Power Aware functionality in combination with register and latch functionality in a single model. Because of this, you can create Power Aware models in Verilog to specify Power Aware behavior for inferred registers and latches, as well as provide the functionality directly through direct instantiation of the Power Aware models. Combining both of these functional descriptions in a single model facilitates the testing of the model for use in Power Aware verification.

Note



Use of these models is only allowed in the -novopt simulation flow. Refer to the section “[Using the No-Optimization Flow](#)” for more information.

Assumptions and Advantages

- The silicon foundry is responsible for the specification of these models to match the behavior of their Power Aware cell technology.
- Capturing the Power Aware behavior in standard Verilog gives the vendor the flexibility to add new cell types and behaviors without creating additional simulation requirements for those cells.
- Foundries providing Power Aware models have control over the protection of their intellectual property (IP).

Basic Model Structure

Model vendors can implement the Verilog model in any style. The cells communicate important events to the simulator using named events. Power Aware verification defines a standard set of

named event identifiers that are used in the model. Each named event corresponds to a particular action to be taken by the simulator.

The simulator communicates with the model by connecting the model to the clock, reset, power (on/off) and power retention signals. Through events on these signals, the model determines when the Power Aware events are triggered, notifying the simulator that the normal RTL behavior must be modified to reflect power control network activity. Because the only inputs are single-bit inputs and the only “output” to the simulator is the triggering of named events, the models (at the RTL or higher abstraction level) are general-purpose and can work with inferred registers and latches of any data type (for VHDL and SystemVerilog support).

The model communicates with the simulator only by triggering the defined named events. The simulator will then map the event trigger into the appropriate Power Aware behavior for the inferred register or latch that the Power Aware model is associated (using the Power Specification File).

The port interface to the Power Aware model can contain additional port declarations. For example, a Power Aware latch model might define **enable**, **data in** and **data out** ports. For the purpose of Power Aware RTL verification, these additional ports are ignored and will not be connected to the design's functional network.

As additional ports are permitted (but ignored), it is feasible to define a Power Aware model that can be verified as functionally correct as it can be instantiated into a test circuit and exercised to ensure it triggers the Power Aware events at the appropriate time and that saved and restored values match what is expected.

A benefit of this approach is that a single Power Aware model can be created for both gate-level verification and Power Aware RTL verification. The gate-level functionality can be used in gate level simulations and would include the cell's Power Aware functionality but not the triggering of the Power Aware events that are designed for use in RTL (or higher) abstraction level simulations. The Power Aware events can be used without the overhead of the gate-level functionality, for RTL and higher abstraction simulations. Together, both sets of functionality can be used to verify the correctness of the model. The example below shows the use of conditional compilation to control inclusion of functional code, the Power Aware code or both in a simulation.

Modeling using conditional compilation implies that the model would be compiled twice into different libraries for use in gate-level and RTL simulations. An alternative would be the use of parameters and conditional generates to control the inclusion of functionality. In any case, the ability to specify a single model for use at multiple abstraction levels simplifies the support and maintenance associated with the development and deployment of Power Aware IP models.

Named Events in Power Aware

The following declarations of named events in a Power Aware Verilog model control simulator activity as indicated.

event pa_store_value

The simulator stores the current value of the inferred registers or latches that the Power Aware model is associated within the power specification file.

event pa_store_x

The simulator stores a corruption value for the inferred registers or latches. Corruption values depend on the type of data inferring the register or latch. A table mapping corruption values to data types is specified separately.

event pa_restore_value

The simulator restores the value previously saved for the inferred registers or latches to the corresponding signal(s) in the design. Restoration of a value results in an event on the corresponding signal to facilitate propagation of known, good states throughout a block that has had power restored.

event pa_restore_x

The simulator restores (re-initializes) the inferred registers or latches to an unknown state specified by the corruption value for the signal's data type. The simulator propagates an event on the restored corruption value.

event pa_corrupt_register; // corrupt the register

The simulator corrupts the current value of the signal corresponding to the inferred registers or latches. The corruption value used is determined by the data type/corruption value table specified separately. No event is propagated due to the corruption. NOTE: See [Usage Note for Sequence Requirements](#) (below).

event pa_set_register; // set the register

The simulator sets the current value of the signal corresponding to the inferred register or latch to a set value, which is inferred from the RTL code. The simulator propagates an event on the set signal value.

event pa_reset_register; // reset the register

The simulator sets the current value of the signal corresponding to the inferred register or latch to a reset value, which is inferred from the RTL code. The simulator propagates an event on the reset signal value.

event pa_restore_hold_register

The simulator restores the value previously saved and holds that value until a pa_release_register event is raised. NOTE: See [Usage Note for Sequence Requirements](#) (below).

event pa_release_register

Attributes

The simulator releases any forced values on a register. If the register is combinational, the simulator re-evaluates the register. NOTE: See [Usage Note for Sequence Requirements](#) (below).

```
event pa_release_reeval_register
```

The simulator re-evaluates a latch at powerup. Forces the register to be re-evaluated if the latch enable is active when power is restored.

```
event pa_iso_on
```

The simulator is notified that an isolation period has begun. Use `pa_release_register` event to identify the end of an isolation period.

The model is responsible for raising the named event when the model of the Power Aware cell is in the appropriate state.

For instance, when the retention signal goes high and the clock is in the proper state in a CLRFF (clock-low, retention flip-flop), the model should raise the `pa_store_value` event. When the power goes low, the `pa_corrupt_register` event should be raised.

Usage Note for Sequence Requirements

When you use a `pa_restore_hold_register` or `pa_corrupt_register` event, you must include a corresponding `pa_release_register` or `pa_release_reeval_register` event in the model in the next sequence. The release event can be in a different always block (for example), but it must be next in the sequence.

Attributes

To assist in the identification of Power Aware cells and facilitate their mapping inferred sequential elements, attributes are placed within the module to provide easily located information. The attributes names and allowed values, as well as contexts in which they are used, are as specified.

Retention Cells and Memories

The attribute name is **is_retention** and the allowed attribute values are the strings corresponding to the **pacell_type** specified in the section [Power Model Mapping Statement](#).

```
(* is_retention = <pacell_type_string> *)
```

Where **pacell_type_string** is one of:

```
"FF_CKHI "  
"FF_CKLO "  
"FF_CKFR "
```

```
"LA_ENHI "  
"LA_ENLO "  
"LA_ENFR "  
"RETMEM_CKHI "  
"RETMEM_CKLO "  
"RETMEM_CKFR "
```

Note that within this context, the pacell types of ANY_CKHI, ANY_CKLO and ANY_CKFR have no meaning as these types are used to map any inferred register or latch. Within the context of attributing a retention cell, that cell will be either a register or latch and that information will be known at the time of attribution.

Example

```
(* is_retention = "FF_CKFR" *)           // Clock free register  
(* is_retention = "RETMEM_CKHI" *)      // Retention memory sensitive  
//           on posedge of clock
```

Isolation Cells

The behavior of isolation cells is automatically introduced at the RTL or higher levels through specification of output corruption. However, it is necessary to attribute the gate level library isolation cell models to ensure that the gate level design matches the verified and specified RTL (or higher) functionality. The **is_isolation_cell** attribute is Boolean.

Example

```
(* is_isolation_cell *)                // According to IEEE 1364, this  
// is equivalent to:  
(* is_isolation_cell = 1 *)
```

Level Shifters

Level shifters imply no functional behavior at RTL or higher. However, they are required to ensure proper operation and scaling of signal values from one voltage domain to another. Attributing level shifter cells in the gate level library ensures the gate level design matches the verified and specified RTL (or higher) design specification. The **is_level_shifter** attribute is Boolean.

Example

```
(* is_level_shifter *)                 // According to IEEE 1364, this  
// is equivalent to:  
(* is_level_shifter = 1 *)
```

Model Interface Ports

The model must define the necessary ports using the names as specified in this section. All ports are required even if the optional functionality does not apply to a specific inferred register or latch. This port interface specification allows generic models that can be applied to a variety of inferred registers and latches.

All ports specified below are input ports. The simulator will connect the ports to the corresponding functional and power control network signals by name. Verilog is case sensitive.

- **PWR** — Power control network signal that indicates whether power is on or off for the power island that this model is associated with. This port is always connected.
- **Retention port(s)** — One of the following must be defined by the module and the retention port(s) will always be connected:
 - **RET** — Power control network signal that indicates whether or not the state of the inferred register or latch must be saved (or restored).
 - **SAVE, RESTORE** — Separate ports to signal save and restore separately.
- **CLK** — Functional network data in enable signal. For registers, this would be a clock signal. For latches, this will be the enable signal. This port will always be connected.
- **SET** — Functional network control signal indicating that the inferred sequential model's value should be set or preset. The functionality of this port is optional and the port will not be connected if there are no inferred set or preset conditions. This port needs to be modeled active high or active posedge. The tool infers the control signal and its polarity and will automatically negate the signal's value when it is connected to the model if it is active low or active negedge.
- **RESET** — Functional network control signal indicating that the inferred sequential model's value should be reset or clear. The functionality of this port is optional and the port will not be connected if there are no inferred reset or clear conditions. The tool infers the control signal and its polarity and will automatically negate the signal's value when it is connected to the model if it is active low or active negedge.

Power and retention ports may be connected to an expression involving two or more signals (each).

Customizing Activity at Time Zero

The default models present in the mtiPA library are equipped to handle any power aware activity at time zero (0). However, there may be instances where you want to use your own custom simulation models for power aware verification. In such cases, if you require power aware activity at time 0 to be honored then you need to ensure that your models are equipped to do so.

To utilize time zero corruption through your own models instead of the default ones (mtiPA) you must add some extra logic in his model as follows:

```
parameter int pa_time0param = 0;
initial begin
    #0;
    if(pa_time0param == 1 && (PWR === 1'bx || PWR === 1'b0))
        ->pa_corrupt_register;
end
```

The changed models like corrupt.v andupf_retention_ret.v etc... have same extra logic as follows:

```
module CORRUPT(PWR);
    parameter int pa_time0param = 0;
    input PWR;
    event pa_corrupt_register;
    event pa_release_register;

    initial begin
        #0;
        if(pa_time0param == 1 && (PWR === 1'bx || PWR === 1'b0))
            ->pa_corrupt_register;
        end

        always @(negedge PWR)
            -> pa_corrupt_register;
        always @(posedge PWR)
            -> pa_release_register;
    endmodule
```

Example—Register Model

The following Verilog code represents a behavioral model of a Clock Free Retention Flip Flop (CFRFF) modified to use many of the listed named events. The RTL verification event generation functionality is separated from the “gate-level” cell functionality to demonstrate how a single model can be defined for use at both RTL and gate levels for Power Aware verification as well as facilitate the verification of both functional aspects of the model.

```
module CFRFF (
    PWR, RET, CLK, SET, RESET
    `ifdef PA_GLS_FUNC // Extra ports would be left unconnected
        , D, Q // It isn't necessary to conditionally
    `endif // compile them out for RTL PA
) ;

    input PWR;
    input RET;
    input CLK;
    input SET; // Not used in this model
    input RESET;
    `ifdef PA_GLS_FUNC
```

Model Construction for Power Aware Simulation

Example—Register Model

```
input D;
output Q;

reg Q;

reg reg_q;
reg reg_q_ret;
reg ret_value;
reg restore_value;
reg posedge_power_w_reset;
reg negedge_ret_w_reset;
reg reset_active;
`endif // PA_GLS_FUNC

// MG event declarations
`ifdef PA_RTL_FUNC // Would not be needed in GLS
event pa_store_value;
event pa_store_x;
event pa_restore_value;
event pa_restore_x;
event pa_corrupt_register;
event pa_reset_register;
`endif // PA_RTL_FUNC

`ifdef PA_GLS_FUNC
// Functionality in this section is used only in Gate Level
// simulations or in the verification of the PA RTL functionality
// (triggering of the appropriate events at the appropriate time).

initial
begin
Q = 0;
ret_value = 0;
restore_value = 0;
posedge_power_w_reset = 0;
negedge_ret_w_reset = 0;
reset_active = 0;
end

always @ (PWR, RESET,
RET, reg_q, ret_value,
posedge_power_w_reset, negedge_ret_w_reset)
begin : output_mux
if (~PWR)
begin
Q <= 1'bx;
end
else if (posedge_power_w_reset)
Q <= reg_q;
else if (negedge_ret_w_reset)
Q <= reg_q;
else if (RET)
begin
Q <= reg_q_ret;
end
else
Q <= reg_q;
end
end
```

```

always @( RESET)
begin
reset_active = RESET;
end

always @(posedge CLK or negedge RESET)
begin : ff_process
if (RESET)
reg_q <= 1'b0;
else
reg_q <= D;
end

always @(posedge CLK)
begin : ret_ff_process
if (~RET)
reg_q_ret <= D;
end

always @(posedge RET)
begin
ret_value <= reg_q;
end

always @(negedge RET)
restore_value <= ret_value;

always @(negedge PWR)
begin
begin
wait (PWR);
if (~RESET) posedge_power_w_reset <= 1;
wait (!CLK);
wait (CLK);
posedge_power_w_reset <= 0;
end
end

`endif // PA_GLS_FUNC

`ifdef PA_RTL_FUNC
// Functionality in this section is used for Power Aware RTL (or higher)
// abstraction verification. It can also be combined with the gate
// level functionality for the purpose of verifying both.

always @(posedge RET)
-> pa_store_value;

always @(negedge RET)
begin
if ( (RESET) && PWR)
-> pa_reset_register;
end

always @(posedge PWR)

```

```
begin
if (RET)
begin
-> pa_restore_value;
end
end

always @(negedge PWR)
-> pa_corrupt_register;

`endif // PA_RTL_FUNC

endmodule
```

Example—Corrupt Model

The following Verilog code shows how to create a simple corruption model that initiates and releases corruption on a register:

```
module CORRUPT(PWR);
input PWR;
event pa_corrupt_register;
event pa_release_register;

always @(negedge PWR)
-> pa_corrupt_register;

always @(posedge PWR)
-> pa_release_register;

endmodule // corrupt
```

where

- The `pa_corrupt_register` statement causes the simulator to corrupt the current value of the signal corresponding to the inferred registers or latches.
- The `pa_release_register` statement causes the simulator to release any forced values on a register. If the register is combinational, the simulator re-evaluates the register.

Appendix C

UPF Commands and Reference

This appendix provides information on Unified Power Format (UPF), which is a standardized set of low-power design specifications for use throughout design, analysis, verification, and implementation.

- [Unified Power Format \(UPF\)](#)
- [UPF Standards](#)
- [Supported UPF Commands](#)
- [Supported UPF Package Functions](#)
- [Accessing Generate Blocks in UPF](#)
- [Supported UPF Attributes](#)
- [Supported UPF Extensions](#)
- [UPF Supply Connections](#)
- [Value Conversion Tables](#)
- [Supply Nets](#)

Unified Power Format (UPF)

You apply UPF as a user-defined file that specifies the Power Aware characteristics of a design for use by the simulator. UPF file format adheres to v1.0 of the UPF standard by default. In addition, Mentor also supports portions of v2.0 (IEEE Std1801-2009). Mentor supports the use of the industry-wide standard known as Unified Power Format (UPF), which consists of commands and statements in a Tcl text file that define the low-power intent for a design.

A UPF file is designed to capture all Power Aware characteristics of the design at the RTL or gate level in a compact form that can be easily used by the simulator. The scope of the UPF file is to provide a standardized format for specifying the supply network, switches, power isolation, data retention, and other aspects relevant to power management of an electronic design.

The UPF file is the key to using Power Aware verification on your design. This file provides the following information required to overlay RTL or gate-level verification with the power control network and Power Aware functionality:

- Power regions, voltage domains, and power islands

- Retention sequential models, their type and the regions they are in
- State and output corruption behavior in power-down situations
- Power control signals and the portions of the design they control

Using a UPF File as Part of Power Aware Simulation

The following procedure describes how to use a UPF file as part of basic Power Aware simulation (see [Using the Standard Flow](#)).

1. Use a text editor to create a UPF file that contains commands that specify the Supply Network over your design. These commands are TCL functions, as described in the current UPF standard.
2. Specify this UPF file as part of the `vopt` command, using the `-pa_upf` switch, as follows:

```
vopt -o <opt_top> <Design Top> -pa_upf <upf_file>
```

Result: `vopt` creates a parallel hierarchy that contains the power supply network of the design. This hierarchy also triggers the control signals that are used to control corruption, retention, or isolation.

Also, Power Aware data is written to either the current working directory or to a library that you can specify using the `-pa_lib` switch.

3. If you are importing a UPF package into a Verilog test bench or file, and the `mtiUPF` variable is not defined in your `modelsim.ini` file, you must use the `vlog` command with the `-L` argument to compile the test bench, as follows:

```
vlog <filename> -L mtiUPF
```

Note that this variable is defined by default, so you generally do not have to do this.

Alternatively, if you want to use a UPF package with VHDL, you must include it in your VHDL file, as follows:

```
library IEEE;  
use IEEE.UPF.all;
```

UPF Standards

At this time, two versions of the UPF standard have been published:

- [Version 1.0 of the UPF Standard](#)
- [Version 2.0 of the UPF Standard: IEEE Std 1801-2009](#)

Power Aware simulation currently supports v2.0 of the UPF standard by default.

In addition, Power Aware simulation also supports portions of v1.0, published by Accellera, which you can implement in a given UPF file by inserting the following text as the first line:

```
upf_version 1.0
```

Version 1.0 of the UPF Standard

The technological foundation for the UPF standard was originally developed by Accellera Organization, Inc.¹ UPF 1.0 was approved as an Accellera standard in February 2007. Version 1.0 of the standard was administered by the P1801 Low Power Working Group of the IEEE. For more information on this working group, refer to the following web location:

http://www.accellera.org/activities/p1801_upf/

You can obtain a copy of v1.0 of the UPF standard (February 2007) in PDF format from the following web location:

http://www.accellera.org/apps/group_public/download.php/989/upf.v1.0.pdf

Version 2.0 of the UPF Standard: IEEE Std 1801-2009

In May 2007, Accellera donated UPF v1.0 to the IEEE for the purposes of creating an IEEE standard. The donation was assigned to the P1801 working group and was eventually developed into a formal standard titled the *IEEE Standard for the Design and Verification of Low Power Integrated Circuits* (IEEE Std1801-2009). Although this standard is the first official IEEE version, it represents the second version of what is more informally referred to as UPF v2.0.

When you use v2.0, Power Aware simulation parses all UPF v2.0 commands and displays a warning message for the following:

- Unsupported commands
- Any unsupported arguments for a supported command

Syntax and Semantic Differences Between UPF 1.0 and UPF 2.0

The release of Version 2.0 of the UPF Standard from IEEE introduced syntactical and semantic differences from the Version 1.0 UPF Standard from Accellera. Power Aware simulation provides functionality to help you control its behavior for these changes.

By default, Power Aware simulation assumes that the UPF file is written in UPF 2.0 and expects proper syntax from that specification and will treat semantic behavior to follow UPF 2.0 specification as well.

1. <http://www.accellera.org>

UPF 2.0 Syntax Differences

You can review the specifications for UPF 2.0 and UPF 1.0 to determine the syntax changes between the two.

The UPF command **upf_version** defines the syntax within the UPF file. If you do not specify this command, Power Aware simulation assumes the version is UPF 2.0.

UPF 2.0 Semantic Differences

By default, Power Aware simulation applies the semantic behavior applicable to UPF 2.0, however you can override this behavior by specifying the **-pa_upfversion=1.0** option with your **vopt** command.

The following are some noticeable differences between UPF 2.0 and UPF 1.0:

- Default port state — OFF in UPF 2.0, while FULL_ON in UPF 1.0. Refer to the section “[Detailed Support for supply_net_type](#)” for more information. You can control the behavior specific to this semantic difference with the **defaultoff** argument to the vopt **-pa_enable** or **-pa_disable** switches.
- Undetermined state functionality — This state did not exist in UPF 1.0. This impacts power switch behavior and resolution semantics. Refer to the section “[Detailed Support for supply_net_type](#)” for more information. You can control this behavior with the **undeterminedstate** argument to the vopt **-pa_enable** or **-pa_disable** switches.
- ACK port behavior — This requires a mandatory supply set on the power switch in UPF 2.0 and was always on in UPF 1.0. You can control this behavior with the **ackportbehavior** argument to the **-pa_enable** or **-pa_disable** options.
- Lower boundary isolation support. — Refer to the section “[set_isolation](#)” for additional information. You can control this behavior with the **lowerboundary** argument to the vopt **-pa_enable** or **-pa_disable** switches.

Supported UPF Commands

This section provides reference information on the UPF commands that you can use with Power Aware simulation. These commands are listed in [Table C-1](#).

Note



Note that some commands in [Table C-1](#) have arguments that are not supported, and some commands or arguments are supported only if the UPF file contains the command:
`upf_version 2.0`

Table C-1 lists UPF commands that are not supported for this release of Power Aware simulation.

Table C-1. List of Supported UPF Commands

Command	UPF Support		Description
	1.0	2.0	
add_domain_elements	Y	Y	Adds design elements to the power domain.
add_port_state	Y	Y	Adds state information to a supply port, which can represent off-chip supply sources that are not driven by the test bench.
add_power_state	N/A	Y	Attributes one or more power states to a supply set or a power domain.
add_pst_state	Y	Y	Specifies a power state table (PST) to define states on a supply net.
associate_supply_set	N/A	Y	Associates a supply set or <code>supply_set_ref</code> to a power domain, power switch, or strategy <code>supply_set_handle</code> .
bind_checker	Y	Y	Inserts a checker module into your design without modifying the design code or introducing functional changes.
connect_logic_net	N/A	Y	Connects a logic net to a logic port.
connect_supply_net	Partial	Partial	Connects a supply net to the specified ports.
connect_supply_set	N/A	Partial	Defines automatic connection semantics on supply sets.
create_composite_domain	N/A	Y	Defines a set of domains.
create_hdl2upf_vct	Y	Y	Defines a value conversion table (VCT) that can be used in converting HDL logic values into state type values.
create_logic_net	N/A	Y	Creates a net reflecting the same value as a port that you connect to it using the <code>connect_logic_net</code> command.
create_logic_port	N/A	Y	Creates a logic port on the active scope.
create_power_domain	Partial	Y	Defines a set of design elements that share a common primary supply set, can specify additional supply sets used within the domain, and any isolation, level-shifting, or retention strategies.

Table C-1. List of Supported UPF Commands

Command	UPF Support		Description
	1.0	2.0	
create_power_switch	Partial	Y	Defines an instance of a power switch in the active scope or the scope of the -domain argument when provided.
create_pst	Y	Y	Creates a power state table (PST).
create_supply_net	Y	Y	Creates a supply net.
create_supply_port	Y	Y	Defines a supply port.
create_supply_set	N/A	Y	Creates the supply set name within the active scope.
create_upf2hdl_vct	Y	Y	Defines a value conversion table for the two LSBs of the supply_state_type.state value when that value is propagated from a UPF supply net into a logic port defined in an HDL.
describe_state_transition	N	N	Not supported
find_objects	N/A	Y	Finds logical hierarchy objects within a scope.
get_supply_set	N	N/A	Not supported
load_simstate_behavior	N/A	Y	Loads a UPF file containing the defaults of simstate behavior for a library.
load_upf	Y	Y	Sets the scope to the specified instance and executes the set of UPF commands in the specified UPF file.
load_upf_protected	N/A	Y	Loads a UPF file in a protected environment that prevents corruption of existing variables
map_isolation_cell	N	Y	Maps an isolation strategy to a library cell or range of library cells to be inserted for isolation.
map_level_shifter_cell	N	Y	Maps a particular level-shifter strategy to a simulation or implementation model.
map_power_switch	N	N	Not supported
map_retention_cell	Partial	Partial	Constrains retention strategy implementation choices and may also specify functional retention behavior for verification.

Table C-1. List of Supported UPF Commands

Command	UPF Support		Description
	1.0	2.0	
merge_power_domains	N	N	Not supported
name_format	Partial	Partial	Defines the format for constructing names of implicitly created objects.
query_design_attributes	N/A	Y	Returns state information about attributes for a design element or model.
query_port_state	N/A	Y	Returns state information for a specified port.
query_power_domain	N/A	Partial	Returns parameters of a power domain.
query_power_state	N/A	Y	Returns the state information for a power domain or supply set.
query_power_switch	N/A	Y	Returns information for a UPF power switch.
query_pst	N/A	Y	Returns information for any defined PSTs.
query_pst_state	N/A	Y	Returns state information for a PST.
query_supply_net	N/A	Y	Returns state information about a previously created supply net.
query_supply_port	N/A	Y	Returns state information about a previously created supply port.
save_upf	Partial	Y	Creates a UPF file of the structures in the relative to the active scope.
set_design_attributes	N/A	Partial	Sets the specified attributes for models or design elements.
set_design_top	Y	Y	Specifies the root of the design.
set_domain_supply_net	Y	Y	Associates the power and ground supply nets with the primary supply set for the domain.
set_isolation	Partial	Partial	Specifies the ports on the domain to isolate using the specified strategy.
set_isolation_control	Partial	Partial	Specifies the control signals for a previously defined isolation strategy.
set_level_shifter	Partial	Partial	Specifies a strategy for level-shifting during implementation.

Table C-1. List of Supported UPF Commands

Command	UPF Support		Description
	1.0	2.0	
set_partial_on_translation	N/A	Yes	Defines the translation of PARTIAL_ON to FULL_ON or OFF for purposes of evaluating the power state of supply sets and power domains.
set_pin_related_supply	Y	Y	Defines the related power and ground pins for signal pins on a library cell.
set_port_attributes	N/A	Y	Specifies information relevant to ports on the interface of power domains.
set_power_switch	Partial	Y	Extends a switch by adding the input supply port, output supply port, and states to the switch.
set_retention	Partial	Partial	Specifies a set of objects in the domain that need to be retention registers and identifies the save and restore behavior.
set_retention_control	Partial	Partial	Specifies the control signals and assertions for a previously defined retention strategy.
set_retention_elements	N/A	N	Not supported
set_scope	Y	Y	Specifies the active UPF scope.
set_simstate_behavior	N/A	Y	Specifies the simulation simstate behavior for a model or library.
upf_version	Y	Y	Specifies the UPF version for interpreting subsequent commands.

add_domain_elements

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument Name	Comments/Restrictions
-elements	

Usage Notes

This command adds design elements to the power domain.

add_port_state

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-state	

Usage Notes

Adds state information to a supply port, which can represent off-chip supply sources that are not driven by the test bench.

add_power_state

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-state	The name you specify here is simply an identifier; it has no semantic meaning.
-supply_expr	Including, but not limited to, support for: <ul style="list-style-type: none"> • power==FULL_ON (similarly, SS.power == FULL_ON) • primary == SS_State1 (similarly, PD.primary ==SS_State1) • primary.power == FULL_ON (similarly, PD.primary.power == FULL_ON) • RET.supply == SS_State2 (similarly, PD.RET.supply == SS_State2) • RET.supply.power == FULL_ON (similarly, PD.RET.supply.power == FULL_ON)
-logic_expr	
-legal -illegal	
-update	

Argument	Comments/Restrictions
-simstate	<p>Values:</p> <ul style="list-style-type: none"> • CORRUPT The power level of the supply set is either off (one or more supply nets in the set are switched off, terminating the flow of current) or at such a low level that it cannot support switching and the retention of the state of logic nets cannot be guaranteed to be maintained even in the absence of changes or activity in the elements powered by the supply. • CORRUPT_ON_ACTIVITY The power level of the supply set is insufficient to support activity. However, the power level is sufficient that logic nets retain their state as long as there is no activity within the elements connected to the supply. • CORRUPT_STATE_ON_ACTIVITY The power level of the supply set is sufficient to support combinational logic, but it is not sufficient to support activity inside state elements, whether that activity would result in any state change or not. • CORRUPT_STATE_ON_CHANGE The power level of the supply set is sufficient to support combinational logic, but it is not sufficient to support a change of state for state elements. • NORMAL The power level of the supply set is sufficient to support full and complete operational (switching) capabilities with characterized timing. • NOT_NORMAL This is a special, placeholder state. It allows early specification of a non-operational power state while deferring the detail of whether the supply set is in the CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_STATE_ON_CHANGE, or CORRUPT_STATE_ON_ACTIVITY simstate. If the supply set matches a power state specified with simstate NOT_NORMAL, the semantics of CORRUPT shall be applied, unless overridden by a tool-specific option. NOT_NORMAL semantics shall never be interpreted as NORMAL.

Usage Notes

This command attributes one or more power states to a supply set or a power domain.

Supply set

A supply set is a grouping of supply nets that collectively define a complete power supply. The power state of a supply set is specified in terms of the supply nets that constitute the set. It is the combined states of the constituent supply nets that determine the following:

- Whether there is current available to power an element.
- The voltage level of the supply. Simstates are associated with power states of supply sets.

Semantics for supply set simulation are applied to the elements connected to the supply set when you enable simstate behavior.

You can also reference supply sets in `add_power_state` as handles. Here, only those handles associated with some supply sets when `add_power_state` is invoked are valid.

Note



It is an error when there are no supply nets associated with the handles and the handles are used in the expression

Power domain

The power state of a domain is determined by the state of supply sets associated with the domain.

For example, the definition of a domain's `MY_DOMAIN_IS_ON` power state would logically require that the primary supply set be in a power state that is a `NORMAL` simstate (all supply nets of the primary supply set are on and the current delivered by the power circuit sufficient to support normal operation.)

Similarly, a `SLEEP` mode for the domain may require the primary supply set to be in power state whose simstate is not `NORMAL` (perhaps `CORRUPT`), while appropriate retention and isolation supplies are `NORMAL`.

You can define the power state for a domain directly in terms of supply nets using `-supply_expr` in addition to the `-logic_expr`.

- If a domain's power state `-logic_expr` specification includes comparison of another domain's active state to a power state defined on that domain, it is equivalent to including the `-logic_expr` and `-supply_expr` specifications for that power state of the referenced domain in the definition of the power state.
- If a domain's power state `-logic_expr` specification includes comparison of a supply set's active state to a power state defined on that supply set, it is equivalent to including the `-logic_expr` and `-supply_expr` specifications for that power state of the referenced supply set in the definition of the power state.

add_pst_state

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-pst	
-state	

Usage Notes

Allows specifying a power state table (PST) to define states on a supply net.

associate_supply_set

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-handle	All predefined handles for supply set are supported.

Usage Notes

Associates a supply set or supply_set_ref to a power domain, power switch, or strategy supply_set_handle.

bind_checker

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-module	Y	Y	
-elements	Y	Y	Elements should be relative to the active scope.
-bind_to [-arch]	n/a	Y	
-ports	Y	Y	The <i>net_name</i> argument accepts symbolic references. Refer to Usage note for additional information

Usage Notes

Note



This command is only supported in the vopt flow.

This command inserts a checker module into your design without modifying the design code or introducing functional changes

net_name argument to -ports

The -ports argument accepts the following symbolic references for the *net_name* argument:

- isolation_signal
`<scope_name>.<pd_name>.<iso_stratgy_name>.isolation_signal`
- isolation_supply_set
`<scope_name>.<pd_name>.<iso_stratgy_name>.isolation_supply_set`
- isolation_power_net
`<scope_name>.<pd_name>.<iso_stratgy_name>.isolation_power_net`
- isolation_ground_net
`<scope_name>.<pd_name>.<iso_stratgy_name>.isolation_ground_net`
- <supply_set>.<function_name>
`ISO_SS.primary`

- `save_signal` of retention strategies
`<scope_name>.<pd_name>.<retention_stratgy_name>.save_signal`
- `restore_signal` of retention strategies
`<scope_name>.<pd_name>.<retention_stratgy_name>.restore_signal`
- `retention_supply_set`
`<scope_name>.<pd_name>.<retention_stratgy_name>.supply`
- `retention_power_net`
`<scope_name>.<pd_name>.retention_power_net`
- `retention_ground_net`
`<scope_name>.<pd_name>.retention_ground_net`

Example

This example shows you could create a low power assertion to: Flag if an isolation supply goes down during an isolation period.

- Checker module:

```
module ISO_SUPPLY_CHECKER(ISO_CTRL, ISO_PWR, ISO_GND);
    import UPF::*;
    input ISO_CTRL;
    input supply_net_type ISO_PWR;
    input supply_net_type ISO_GND;
    reg ISO_pg_sig;

    assign ISO_pg_sig = get_supply_on_state(ISO_PWR) && \
        get_supply_on_state(ISO_GND);

    always @(negedge ISO_pg_sig)
        assert(!(ISO_CTRL)) else \
            $display("\n At time %0d isolation supply is switched OFF \
                during isolation period, ISO_CTRL=%b", $time, ISO_CTRL);
endmodule
```

- Design

```
module tb();
    ...
    top top(...);
    ...
endmodule

module top(...);
    mid_vl test1_vl(...);
    mid_vl test2_vl(...);
    mid_vl test3_vl(...);
endmodule;
```

```
module mid_v1(...);  
...  
endmodule
```

- UPF

```
upf_version 2.0  
  
set_scope /tb/TOP  
  
create_supply_net ISO_PWR  
create_supply_net ISO_GND  
create_supply_port ISO_PWR_PORT  
create_supply_port ISO_GND_PORT  
connect_supply_net ISO_PWR -port ISO_PWR_PORT  
connect_supply_net ISO_GND -port ISO_GND_PORT  
create_supply_set ISO_SS -function {power ISO_PWR} \  
-function {ground ISO_GND}  
create_power_domain PD_mid1 -supply {primary ISO_SS}  
set_isolation iso_PD_mid1 \  
-domain PD_mid1\  
-applies_to outputs\  
-isolation_supply_set ISO_SS\  
-location self\  
-isolation_signal ctrl  
bind_checker iso_supply_chk \  
-module ISO_SUPPLY_CHECKER \  
-bind_to mid_v1 \  
-ports {\  
  {ISO_CTRL PD_mid1.iso_PD_mid1.isolation_signal} \  
  {ISO_PWR ../ISO_SS.power} \  
  {ISO_GND ../ISO_SS.ground} \  
}
```

Power Aware simulation inserts **ISO_SUPPLY_CHECKER** in all instances of the **mid_v1** module in your design.

You should ensure the *net_name* arguments to **-ports** are present in the target instance scope or the parent scope. If you want to associate an object present in the active UPF scope then prefix *net_name* with “./”. For example, **PD_mid1** was created in scope, therefore **PD_mid1.iso_PD_mid1.isolation_signal** is directly accessible. If something is created in active scope, such as if you are not sure if it would be present in scope of the target instance, you should include the prefix “./”, as is done for **./ISO_SS.power** and **./ISO_SS.ground**.

connect_logic_net

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-ports	

Usage Notes

Connects a logic net to a logic port.

connect_supply_net

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-ports	Y	Y	
-pg_type	Y	Y	Using an <element_list> is not supported.
-vct	Y	Y	
-pins	Y	Y	
-cells	Y	Y	
-domain	Y	Y	
-rail_connection	N	N	Not supported.

Usage Notes

Refer to “[UPF Supply Connections](#)”

connect_supply_set

Support for UPF Standard

v1.0 — no

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-connect	N	Y	
-elements	N	Y	
-exclude_elements	N	N	Not supported
-transitive	N	N	Not supported

Usage Notes

Defines automatic connection semantics on supply sets. Supply nets of supply sets are automatically connected to the supply ports of design elements based on the purpose of the supply set in a given domain or strategy context and the function that a supply net performs in the context of supply set.

Restrictions:

- Supply sets specified in strategy context are not automatically connected to design elements.
- Supply nets in supply sets functioning as predefined supply set functions are not automatically connected according to their predefined function. You must specify explicit automatic connections.

Examples

```
connect_supply_set PD.primary \  
-connect {power primary_power} \  
-elements TOP  
  
connect_supply_set PD.ISO.isolation_supply_set \  
-connect {iso_power primary_power} \  
-connect {iso_ground primary_ground} \  
  
connect_supply_set PD.RET.retention_supply_set \  
-connect {ret_backup_power backup_power} \  
-connect {switchtable_supply primary_power}
```

create_composite_domain

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-subdomains	
-supply	
-update	

Usage Notes

A composite power domain is a set of domains (referred to as subdomains), each of which has a primary supply set as necessary common property.

A composite domain contains these attributes:

- primary supply set handles
- power states

Operations on a supply set

UPF v2.0 states that all operations performed on a composite domain are transitively applied to each subdomain. This implies that many UPF commands may refer to a composite domain, either directly using the -domain(s) argument or by using a handle name, in case of supply sets.

UPF commands applied on a composite domain are applied only to those domains that are present at that time in the subdomain tree of that composite domain—they will not be applied to the subdomains added later.

Implementation

- Checks
 - If a composite domain with the given `composite_domain_name` exists in the current scope, you must specify the -update argument, or it will be flagged as an error. Similarly, if the composite domain is new and does not exist in the current scope, it will be an error to use the -update argument.
- Subdomains
 - For each subdomain provided in the -subdomain list, it will be an error if any of given rooted name does not match a power domain or composite domain in the current scope.

- A domain can be a subdomain to multiple composite domains. However, this may lead to conflicting commands being applied on that domain (UPF v2.0 is not clear on this).
- Supply sets
 - Each supply set handle name provided with the `-supply` argument is searched for in the existing supply sets list for that composite domain. If found, the presence of supply set reference is checked for that handle.
 - A warning message is displayed if the reference is present but its name does not match the current the supply set reference name.
 - If the supply set handle did not contain a reference before, but now a reference name has been provided, Power Aware simulation searches the current scope and updates the handle with this reference. It will be an error if any matching reference is not found.
 - If the supply set handle does not exist, Power Aware simulation creates a new handle and populates the reference. Power Aware simulation adds the new handle to the list of supply sets for the current composite domain and adds it to each subdomain.
 - You can add new subdomains and supply set handles to a composite domain subject to the checks mentioned above.
- Power states
 - The `add_power_state` command is supported, which means power states for composite domains are available.
- Support for `save_upf` command
 - In the interpreted mode, Power Aware dumps instances of the `create_composite_domain` command once for each composite domain with all the updates included.
 - All the commands applied on a composite domain are applied to all the subdomains down to the leaf-level power domains. Because a composite domain can contain only subdomains, supply sets, and power states, Power Aware simulation does not store strategies, nets, or other objects on it. As a result, these commands appear as multiple commands applied to each power domain that was a part of the subdomain hierarchy of a composite domain.
 - In uninterpreted mode, Power Aware simulation saves the command texts as-is, and the `create_composite_domain` command may appear multiple times with `-update`. Also, the commands applied on a composite domain are directly dumped.

Interaction of composite domains with other UPF commands

- `add_power_state`
- `associate_supply_set`

In Section 6.15 of UPF v2.0:

It is valid to refer to the primary supply of a composite domain because there is exactly one primary supply common to all subdomains. It is not valid to refer to other `supply_set_handles` or strategies in the composite domain because they are not necessarily common to all sub domains.

This statement asserts that you cannot refer to a supply set handle in a composite domain other than primary supply set. Currently, the `associate_supply_set` command is supported only for the primary supply. For other supplies, you can associate them by using either of the following ways:

- `associate_supply_set` on each sub power domain individually
- using `create_composite_domain -update -supply {supply_set_handle [supply_set_ref]}`

It will be an error if the primary supply handle already exists in a subdomain and points to a different supply set.

Correct usage example:

```
create_composite_domain cd -subdomains {pd2 pd3} -supply {primary}
associate_supply_set pd_ss -handle cd.primary
```

Alternate correct usage example:

```
create_composite_domain cd -subdomains {pd2 pd3}
create_composite_domain cd -supply {ssh pd_ss}
```

Incorrect usage example

```
create_composite_domain cd -subdomains {pd2 pd3} -supply {ssh}
associate_supply_set pd_ss -handle cd.ssh
```

Error message:

```
** Error: ./test.upf(88): UPF: (vopt-9765) It is invalid to use the
'associate_supply_set' command with any composite domain supply set handle
other than the primary handle.
Domain: 'cd', supply set handle: 'pd_ss'
```

- `create_power_switch`

The power switch will be created in the creation scope of `composite_domain`.

- create_supply_net

The supply net will be created in the creation scope of composite_domain. The command will also be applied with a -reuse argument to each subdomain. If the subdomain belongs to a different creation scope, the command will be applied without reuse.

- create_supply_port

The supply port will be created in the creation scope of composite_domain.

- map_isolation_cell

Not supported.

- map_level_shifter_cell

Not supported.

- map_power_switch

Not supported.

- map_retention_cell

Not supported.

- set_isolation

This command will be transitively applied to all the subdomains of a composite domain.

The following are also in effect:

- set_isolation -domain composite_domain cannot have -elements/-exclude_elements/-instance argument. This may cause conflict in the subdomains.
- The elements can be individually updated for each strategy in all the sub power domains.
- If any error occurred while transitively applying this command, it will not be applied to that subdomain.
- If a strategy has already been applied on a subdomain, trying to set a strategy of that same name directly on that subdomain without -update will be an error.

Correct usage example:

```
create_composite_domain cd -subdomains {pd2 pd3} -supply {primary
pdsub_ss}
set_isolation cd_iso1 -domain cd -clamp_value 0 -applies_to inputs -
isolation_signal iso -isolation_sense high -location parent
set_isolation cd_iso2 -domain cd -clamp_value 0 -applies_to outputs -
isolation_signal iso -isolation_sense high -location parent
```

Alternate correct usage:

```
create_composite_domain cd -subdomains {pd2 pd3} -supply {primary
pdsub_ss}
set_isolation cd_iso -domain cd -update -clamp_value 0 -applies_to outputs
-isolation_signal iso -isolation_sense high -location parent
```

```
set_isolation cd_iso -domain pd2 -update -elements
{hier_inst/leaf_inst2/localout_leaf}
set_isolation cd_iso -domain pd3 -update -elements
{hier_inst/leaf_inst3/localout_leaf}
```

Incorrect usage example:

```
create_composite_domain cd -subdomains {pd2 pd3} -supply {primary
pdsusb_ss}
set_isolation cd_iso -domain cd -update -clamp_value 0 -applies_to outputs
-isolation_signal iso -isolation_sense high -location parent -elements
{hier_inst/leaf_inst3/scanout_leaf hier_inst/leaf_inst3/localout_leaf}
```

Error message:

```
** Error: ./test.upf(90): UPF: (vopt-PA-9809) The isolation strategy
'cd_ret' is defined on the composite domain 'cd'.
Specifying elements or instances on this strategy is invalid. Please
specify the member elements/instances on the subpower domains' strategies
by using the -update argument.
```

Help message:

```
power-aware Message # 9809:
A composite domain cannot contain design objects as elements/members. So a
UPF command applied to a composite domain cannot directly refer to
design objects in its elements/instances list. These elements/instances
should be populated individually for each sub power domain using the
strategy/command with -update argument, after applying the
strategy/command on the composite domain.
```

- **set_isolation_control**

This command will be applied to all the subdomains of a composite domain.

- **set_level_shifter**

This command will be applied to all the subdomains of a composite domain.

The following arguments are not supported (they can also refer to `composite_domain`):

- `-source domain_name`
- `-sink domain_name`

The following are also in effect:

- `set_level_shifter -domain composite_domain` cannot use the `-elements`, `-exclude_elements`, or `-instance` arguments. This may cause conflict in the subdomains.
- The elements can be individually updated for each strategy in all the sub-power domains.

- If any error occurred in transitively applying this command on a subdomain, it will not be applied to that subdomain. Usage and error messages for incorrectly specifying `-elements` or `-instance` arguments are similar to those for `set_isolation`.

- `set_port_attributes`

If a composite domain is included in the `-domains` argument of this command, it will be replaced by its sub-power domains, listed transitively.

Example:

```
create_composite_domain cd -subdomains {pd1 pd2 pd3} -supply {primary
pdsusb_ss}
set_port_attributes {-domains {cd} -applies_to inputs} -attribute pin_type
data_in
```

This is equivalent to:

```
set_port_attributes {-domains {pd1 pd2 pd3} -applies_to inputs} -attribute
pin_type data_in
```

- `set_retention`

This command will be applied to all the subdomains of a composite domain.

The following shall also apply:

- `set_isolation` `-domain` `composite_domain` cannot have `-elements`, or `-exclude_elements`, or `-instance` arguments. This may cause conflict in the subdomains.
- The elements can be individually updated for each strategy in all the sub-power domains.
- If any error occurred in transitively applying this command on a subdomain, it will not be applied to that subdomain.

The usage and error messages for incorrectly specifying `-elements` or `-instance` arguments are similar to `set_isolation`.

- `set_retention_control`

This command will be applied to all the subdomains of a composite domain.

- `use_interface_cell`

Not supported.

create_hdl2upf_vct

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-hdl_type	User-defined types not supported.
-table	

Usage Notes

Defines a value conversion table (VCT) from an HDL logic type to the net_state_type of the supply net value when that value is propagated from HDL port to a UPF supply net.

create_logic_net

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
<net_name>	The name of a logic net that you want to create in the active scope.

Usage Notes

Results in the created net reflecting the same value as a port that you connect to it using the `connect_logic_net` command (`net_dut` and `pmb_out` in the example below).

This net can be used as a control signal.

The following are not supported:

- Logic nets with the same name but different creation scope
- Dumping of logic nets with `save_upf` command
- Implicit port/net semantics for logic ports/nets

Example

```
set_scope /tb
create_logic_net net_tb
set_scope /tb/pmb

#connect the logic net to an output port on the
#power management block

connect_logic_net net_tb -ports pmb_out
set_scope /tb/dut
create_logic_port p_dut
create_logic_net net_dut
connect_logic_net net_dut -ports p_dut
set_scope
connect_logic_net net_tb -ports dut/p_dut
```

create_logic_port

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-direction	

Usage Notes

The following are not supported:

- Implicit port/net semantics for logic ports/nets
- Application of isolation strategies on logic ports

create_power_domain

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-simulation_only	N	Y	
-elements	Y	Y	See “Usage Notes” below.
-include_scope	Y	Y	
-supply	N	Y	
-scope	Y	Y	
-define_func_type	N	Y	
-update	N	Y	Allows adding elements and supplies to a previously created power domain for a progressive refinement of power intent.
-exclude_elements	N	Y	

Usage Notes

For the -elements argument:

- You can specify a hierarchical path within the active scope in the list of design elements.
- You can specify bit- or part-selects of one or more signals. However, complex data types of SystemVerilog, such as structs, are not supported.
- According to UPF, only hierarchical path of instances can be accepted inside -elements {element_list}. This has been extended under the above mentioned switch to allow generate hierarchy to be specified in the extent of a power domain. This feature is only available for corruption and retention and not for isolation and level shifting. This implies that if a generate block is added in the extent of a power domain, it will only show corruption and if it contains a retention register, retention behavior as well. However, there will not be any level shifters/isolation cells placed at the generate boundary.

```
create_power_domain PD_forgen -elements { forgen[1] }
```

Examples

- Results in an effective element list of {top1 top2}.

```
create_power_domain PD1 -elements {sig[2] sig1[2:1]}
```

```
create_power_domain PD -elements {top1}  
create_power_domain PD -elements {top2} -update
```

- Defines automatic connection semantics on supply sets. Supply nets of supply sets are automatically connected to the supply ports of design elements based on the purpose of the supply set in a given domain or strategy context and the function that a supply net performs in the context of supply set.

```
create_power_domain PD -elements {top1}  
create_power_domain PD -elements {top2} -update
```

create_power_switch

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-output_supply_port	Y	Y	
-input_supply_port	Y	Y	
-control_port	Y	Y	
-on_state	Y	Y	
-off_state	Y	Y	
-supply_set	N	Y	When the supply set simstate is anything other than NORMAL, the state of the output supply port of a switch is OFF and the acknowledge ports are corrupted. Note that this differs from UPF v2.0, which states that the state of the output supply port of a switch is UNDETERMINED.
-on_partial_state	Y	Y	
-ack_port	Y	Y	
-ack_delay	Y	Y	
-error_state	Y	Y	

Usage Notes

The following is not supported:

- In order to model the definition of a power switch, Power Aware simulation currently treats UNDETERMINED state as OFF state for simulation. This implies that when the state of supply set is NOT_NORMAL, the state on the output supply port will be OFF instead of UNDETERMINED state.

create_pst

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-supplies	

Usage Notes

Creates a power state table (PST).

create_supply_net

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-domain	
-reuse	
-resolve	The value of parallel is implemented differently for v1.0 and v2.0. See “Usage Notes” below.

Usage Notes

UPF v1.0 implemented a slight error in the semantic specification for UPF command `create_supply_net -resolve parallel`, which has been corrected in UPF v2.0.

Power Aware simulation follows the UPF v2.0 specification in this case, since it is the later standard and its definition reflects what was intended to be the semantics for UPF v1.0.

Specifically, if multiple switches drive a supply net with parallel resolution, and the state of at least one (but not all) switch outputs is fully ON, and the remaining switch outputs are OFF, then:

- UPF v1.0 says the state of the driven supply net should be fully ON.
- UPF v2.0 says the state of the driven supply net should be partially ON.

Therefore, in following the UPF v2.0 specification, the state of the driven supply net will be partially ON.

create_supply_port

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-direction	The value of inout is not supported.
-domain	

create_supply_set

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-function	
-reference_gnd	This option is accepted during parsing, however it has no impact on simulation.
-update	

create_upf2hdl_vct

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-hdl_type	User-defined types not supported.
-table	

Usage Notes

Defines a value conversion table (VCT) for the two LSBs of the supply_state_type.state value when that value is propagated from a UPF supply net into a logic port defined in an HDL.

find_objects

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-pattern	
-object_type	
-direction	
-transitive	
-regexp -exact	
-ignore_case	
-non_leaf -leaf_only	not supported

Usage Notes

Use this command to find logical hierarchy objects within a scope. Specifically, use it to search for design elements, nets, or ports that are defined in the HDL. It works on the logical hierarchy and only searches in the scope (or in and below the scope when `-transitive` is specified). The search pattern can be any of the following: exact pattern match, tcl glob expression, or regular expression.

There are different behaviors for case sensitivity:

- VHDL — search is always case-insensitive.
- Verilog — search, by default, is case-sensitive. Use `-ignore_case` to set to case-insensitive

Note



Search for a multi-dimensional array of instances is currently not supported.

You can search for generates and within generate hierarchies when you specify the `-pa_upfextensions=genblk` argument with `vopt`.

Examples

- Tcl glob pattern-matching examples:

```
find_objects top -pattern {a}
find_objects top -pattern {bc[0-3]}
find_objects top -pattern {e*}
```

```
find_objects top -pattern {d?f}  
find_objects top -pattern {g\[0\]}
```

- **Tcl Glob Search: Find all nets**

```
find_objects top -pattern { a* } -object_type net
```

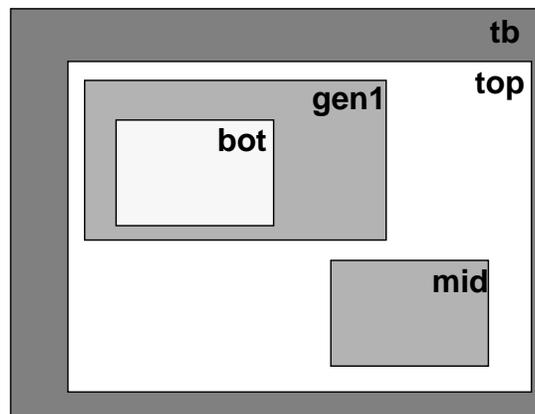
- **Regex Search: Find Out ports**

```
find_objects top -pattern { a_.* } -object_type port -direction out  
-regex
```

- **Exact Search: Find Process**

```
find_objects top -pattern { a_proc } -object_type process -exact
```

- **Finding objects when generates are involved. Given the following design hierarchy:**



- **Default mode**

```
find_objects top -pattern {*} -object_type inst  
Returns: top/mid
```

```
find_objects top -pattern {*} -object_type inst -transitive TRUE  
Returns: top/mid top/gen1/bot
```

```
find_objects top/gen1 -pattern {*} -object_type inst  
Returns: Invalid Scope Error.
```

- **Running with vopt -pa_upfextensions=genblk**

```
find_objects top -pattern {*} -object_type inst  
Returns: top/mid top/gen1
```

```
find_objects top -pattern {*} -object_type inst -transitive TRUE  
Returns: top/mid top/gen1 top/gen1/bot
```

```
find_objects top/gen1 -pattern {*} -object_type inst  
Returns: top/gen1/bot
```

load_simstate_behavior

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
<library_name>	Name of a ModelSim library (Required)
-file	Name of a file containing set_simstate_behavior commands.

Usage Notes

The load_simstate_behavior command loads a UPF file containing the defaults of simstate behavior for a library. This file consists solely of [set_simstate_behavior](#) commands, which are applied to the models in the library specified by the library_name argument.

You can load only one file per occurrence of this command.

Examples

Example C-1. Load Files With load_simstate_behavior Commands

The following example shows how to use the load_simstate_behavior command to load simstate semantics for lib1 and lib2.

```
main.upf
upf_version 2.0
set_scope tb/top1
...
load_simstate_behavior lib1 -file load1.upf
load_simstate_behavior lib2 -file load2.upf
...
create_power_domain pd1
set_simstate_behavior ENABLE -elements {mid1/bot1}
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
...
```

where

load1.upf consists of

```
set_simstate_behavior ENABLE -model mid
set_simstate_behavior DISABLE -model bot
```

load2.upf consists of

```
set_simstate_behavior ENABLE -model top
```

Example C-2. Error/Warning Conditions: Unknown Library Name

It is an error if the specified library cannot be resolved.

```
upf_version 2.0
set_scope tb/top1
...
load_simstate_behavior lib1 -file simstate_file1.upf
...
```

Vopt Message

```
** Error: test.upf(6): UPF: (vopt-9753) Library 'lib1' does not exist.
```

Example C-3. Error/Warning Conditions: Library File Does Not Exist

It is an error if specified file does not exist.

```
upf_version 2.0
set_scope tb/top1
...
load_simstate_behavior lib1 -file sim.upf
...
```

Vopt Message

```
** Error: test.upf(6): UPF: (vopt-9718) Can't open file 'sim.upf'
```

Example C-4. Error/Warning Conditions: Model Cannot Be Found

It is an error if a model specified in file cannot be found.

```
main.upf
upf_version 2.0
set_scope tb/top1
...
load_simstate_behavior lib1 -file sim.upf
...

sim.upf
set_simstate_behavior ENABLE -model mad
```

Vopt Message

```
** Error: sim.upf(1): UPF: (vopt-9655) Model: 'mad' doesn't exist in
library: 'lib1'.
```

Example C-5. Error/Warning Conditions: File Contains Wrong Commands

It is an error if the specified file contains UPF commands other than `set_simstate_behavior`.

```
main.upf
upf_version 2.0
set_scope tb/top1
...
load_simstate_behavior lib1 -file sim.upf
...

sim.upf
set_scope mid1
set_simstate_behavior ENABLE -model mid
```

Vopt Message

```
** Error: sim.upf(1): UPF: (vopt-9754) Command 'set_scope' not allowed in
file specified via command 'load_simstate_behavior'.
```

load_upf

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-scope	
-version	

load_upf_protected

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-hide_globals	
-params	
-scope	
-version	

Usage Notes

This command loads a UPF file in a protected environment that prevents corruption of existing variables.

map_isolation_cell

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
<filename>	Text file that defines isolation strategy and power domain.
-domain	
-elements	This argument takes the list of ports on which user-defined isolation cell is to be placed. If no list specified, then user-defined cells are applied to all the elements belonging to the specified isolation strategy.
-lib_cells	
-lib_cell_type	
-lib_model_name	Specifies the name of the cell/module to be picked up for isolation. This module must be present in a library visible to the Power Aware vopt command.
-port	Provides information about the ports of the library cell. This argument connects the specified net_ref to a port of the model (see Usage Notes, below).

Usage Notes

This command maps a particular isolation strategy to a library cell or range of library cells to be inserted for isolation. It has an effect only if isolation cells have been inserted (see [set_isolation](#)).

The information about the ports of the library model is specified using '-port' argument, which connects the specified net_ref to a port of the model.

A net_ref may be one of the following:

- A logic net name
- A supply net name

- One of the following symbolic references:
 - UPF_ISO_ENABLE (Specific to ModelSim)

Refers to the isolation control signal of associated isolation strategy.
 - UPF_ISO_PWR (Specific to ModelSim)

Refers to the isolation power net of associated isolation strategy.
 - UPF_ISO_GND (Specific to ModelSim)

Refers to the isolation ground net of associated isolation strategy.
 - UPF_GENERIC_DATA

Refers to the port on which the cell is to be placed.
 - UPF_GENERIC_OUTPUT

Refers to the output of the isolation cell.
 - isolation_signal

Refers to the isolation control signal of associated isolation strategy.
 - isolation_signal[index]

Not supported.
 - isolation_supply_set.function_name

The function_name extension refers to the supply net corresponding to the function it provides to the isolation_supply_set.
 - isolation_supply_set[index].function_name

Not supported.

Example C-6. Specifying Argument Values for map_isolation_cell

```
...
set_isolation ISO_GEN \
-domain C0 \
-isolation_power_net PDO_VNET_ISO_OUT \
-clamp_value 1 \
-applies_to outputs

set_isolation_control ISO_GEN \
-domain C0 \
-isolation_signal /tb/iso \
-isolation_sense low \
-location parent
```

```
map_isolation_cell ISO_GEN \  
-domain C0 \  
-elements {mid_inst/out1} \  
-lib_model_name iso_cell_0_vh \  
-port "iso UPF_ISO_ENABLE"  
  
map_isolation_cell ISO_GEN \  
-domain C0 \  
-elements {mid_inst/out1} \  
-lib_model_name iso_cell_0_vh \  
-port "iso !tb_iso"  
-port "cell_in UPF_GENERIC_DATA"  
-port "cell_out UPF_GENERIC_OUTPUT"
```

map_level_shifter_cell

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
<filename>	Text file that defines level shifter strategy.
-domain	
-lib_cells	
-elements	A list of ports to use from the strategy defined in the <textfile> provided with this command.

map_retention_cell

Support for UPF Standard

v1.0 — parital

v2.0 — partial

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-elements	N	N	Not supported.
-exclude_elements	N	N	
-lib_cells	Y	Y	
-lib_cell_type	Y	Y	
-lib_model_name	Y	Y	Port connectivity using -port argument of this option is not supported. Verification semantics of lib_model are not honored for UDP retention, where flops and latches are written as UDPs and retention is applied on them.

name_format

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-isolation_prefix	N	Y	
-isolation_suffix	N	Y	
-level_shift_prefix	N	Y	
-level_shift_suffix	N	Y	
-implicit_supply_suffix	N	N	Not supported.
-implicit_logic_prefix	N	N	Not supported.
-implicit_logic_suffix	N	N	Not supported.

query_design_attributes

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-element	
-model	
-detailed	

Usage Notes

Returns state information about attributes for a design element or model.

query_port_state

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-state	
-detailed	

Usage Notes

Returns state information for a specified port.

query_power_domain

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes (partial)

Arguments

Argument	Comments/Restrictions
-non_leaf	
-all	not supported
-no_elements	
-detailed	

Usage Notes

Returns parameters of a power domain.

query_power_state

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-state	
-detailed	

Usage Notes

Returns the state information for a power domain or supply set. Specifically, listing the previously defined power states for the specified `object_name`, which can be a power domain or a supply set.

query_power_switch

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-detailed	

Usage Notes

Returns information for a UPF power switch.

query_pst

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-detailed	

Usage Notes

Returns information for any defined PSTs.

query_pst_state

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-pst	
-detailed	

Usage Notes

Returns state information for a PST.

query_supply_net

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-domain	
-detailed	

Usage Notes

Returns state information about a previously created supply net.

query_supply_port

Support for UPF Standard

v1.0 — not applicable

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-domain	
-is_supply	
-detailed	

Usage Notes

Returns state information about a previously created supply port.

save_upf

Support for UPF Standard

- v1.0 — yes (partial)
- v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-scope	Y	Y	See Usage Notes, below
-version	Y	Y	Currently, -version is supported only when its value is the same as input file version.
-u	N	Y	(ModelSim extension) Optional argument not specified in UPF v2.0. Uninterpreted mode only, not part of UPF standard You can specify UPF commands in a separate file that is loaded by specifying the filename as the argument to the -pa_tcl command line option. Specifically you can use this functionality to avoid putting non-standard commands, or commands with non-standard arguments, in the main UPF file, which allows you to keep the file portable. Also, navigation commands, such as set_scope, must be used in the -pa_tcl file to ensure that UPF commands in that file are applied in the appropriate scope.

Usage Notes

Power Aware simulation supports two modes for using the save_upf command: interpreted and uninterpreted.

- Interpreted mode (default)

In interpreted mode, any command or argument that is not supported by Power Aware simulation is written as a comment at the end of the saved UPF file. This file contains supported commands as interpreted by Power Aware simulation, which are written after performing various operations such as semantic checks, resolving net-port connections, and resolving design objects related to a command.

Note that save_upf dumps the complete power intent of the scope, not just the UPF commands in effect at the point where save_upf command is given.

- Uninterpreted mode

In uninterpreted mode, all the UPF commands are saved in the output file without any processing (even if the commands are not supported by Power Aware simulation). This mode filters out any TCL-specific constructs in the UPF and writes only the UPF commands to the output UPF file.

To write the output file in uninterpreted mode, do either of the following:

- Specify `save_upf -u`. Note that in uninterpreted mode, the `-scope` and `-version` arguments of the `save_upf` command are not supported. Also, the saved UPF file is a complete replica of the original UPF file (but without any TCL constructs).
- Specify `vopt -pa_dumpupf <filename>`. This saves the UPF file in uninterpreted mode to the `<filename>` file.

set_design_attributes

Support for UPF Standard

v1.0 — no

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-elements	N	Y	
-models	N	Y	
-attribute	N	Y	Refer to Table C-4 for a list of UPF attributes supported for this argument.
-exclude_elements	N	N	Not supported.

set_design_top

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

none

set_domain_supply_net

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-primary_power_net	
-primary_ground_net	

set_isolation

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-elements	Y	Y	Complex data types of SystemVerilog, such as structs, are not supported. However, you can specify bit- or part-selects of one or more signals. Example: set_isolation ISO1 -domain PD1 -elements {out1[2] out2[2:1]}
-applies_to	Y	Y	
-isolation_power_net	Y	Y	
-isolation_ground_net	Y	Y	
-no_isolation	Y	Y	
-isolation_supply_set	Y	Y	Supports only a single value (no list) — multiple clamp values, supplies, and enable isolation strategies are not supported.
-name_prefix	Y	Y	
-name_suffix	Y	Y	
-clamp_value	Y	Y	Specifying a user-defined value for <latch> is not supported.
-location	Y	Y	Supports only the following values: <ul style="list-style-type: none"> • self • parent • fanin • fanout • faninout

Argument	v1.0	v2.0	Comments/Restrictions
-instance	N	Y	<p>Recognizes the special attributes present on the specified instance, applies the appropriate simulation semantics, and makes the connections as follows:</p> <ul style="list-style-type: none"> • If there is a port of type pg_type present on the instance, then Power Aware simulation automatically connects the appropriate power and ground pins of the respective strategy and disable the implicit corruption semantics. • If there is no port of type pg_type present on the instance, then Power Aware simulation applies implicit corruption semantics according to the primary_power and ground nets specified for the strategy.
-update	N	Y	Allows adding elements and supplies to a previously created power domain.
-applies_to_clamp	Y	Y	
-applies_to_sink_off_clamp	Y	Y	
-applies_to_source_off_clamp	Y	Y	
-diff_supply_only	N	Y	No isolation is introduced into the path from the driver to the receiver for an isolation strategy defined on a port on the interface of ref_domain_name, where the driver is powered by the same supply as a receiver of the port.
-force_isolation	N	Y	
-isolation_signal	Y	Y	
-isolation_sense	Y	Y	
-source	N	Y	Filters the ports receiving a net that is driven by logic powered by the supply set.
-sink	N	Y	Filters the ports driving a net that fans out to logic powered by the supply set.
-sink_off_clamp	N	N	Not supported.
-source_off_clamp	N	N	Not supported.
-transitive	N	N	Not supported.

Usage Notes

The isolation strategy defined by `set_isolation` causes Power Aware to perform insertion of isolation cells, together with an analysis of the power state table to see whether adjacent power domains can be in different power states and therefore require isolation.

The `-instance` argument prevents insertion of a redundant isolation cell at a port where one already exists (see [map_isolation_cell](#)).

When both `-source` and `-sink` are specified, a port is included if it has a source as specified and a sink as specified.

This command supports isolation of lower boundary ports of a power domain, which is defined in UPF v2.0 as “The highconn side of ports defined on design elements in other power domains, but instanced within design elements in the extent of the domain.”

```
set_scope /tb
create_power_domain PD_TOP ?elements {top_inst}
create_power_domain PD_BOT ?elements {top_inst/bot_inst}
...
set_isolation PD_TOP_isolation1 -domain PD_TOP \
-isolation_power_net VDD_PD_TOP_net -clamp_value 0 \
-applies_to inputs \
-location parent
```

This command will isolate both the inputs ports of the instance `/tb/top_inst`, as well as outputs of the instance `/tb/top_inst/bot_inst` (lower boundary ports).

set_isolation_control

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-domain	
-isolation_signal	
-isolation_sense	
-location	The sibling value is not supported.

Usage Notes

The -location argument defines the isolation behavior so that it appears in the specified location, which is used by synthesis and/or place-and-route to guide insertion of actual isolation cells.

set_level_shifter

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-elements	Y	Y	Complex data types of SystemVerilog, such as structs, are not supported. However, you can specify bit- or part-selects of one or more signals.
-no_shift	Y	Y	
-threshold	Y	Y	
-applies_to	Y	Y	
-rule	Y	Y	
-location	Y	Y	<p>The -location argument defines where level shifter cells are to be placed in the design.</p> <ul style="list-style-type: none"> • If you specify a value of fanout, a level shifter will be identified for placement at all fanout locations (for valid level shifters) and the count is incremented accordingly in the report. • If you specify self, parent, sibling, or automatic, then only one level shifter will be identified for placement at a fanout location. Thus, the report may show a count of level shifters in some paths to be 0. <p>Supports only the following values:</p> <ul style="list-style-type: none"> • self • parent • fanin • fanout • faninout <p>Example:</p> <pre>set_level_shifter LS1 -domain pd1 - elements {out3} out4[2:1]}</pre>
-instance	N	Y	Recognizes the special attributes present on the specified instance, applies the appropriate simulation semantics, currently disables the Power Aware simstate semantics.
-update	N	Y	Allows adding elements and supplies to a previously created power domain for a progressive refinement of power intent.

Argument	v1.0	v2.0	Comments/Restrictions
-force_shift	N	Y	
-input_supply_set	N	Y	
-output_supply_set	N	Y	
-internal_supply_set	N	Y	
-name_prefix	Y	Y	
-name_suffix	Y	Y	
-source	N	Y	Selects the ports receiving a net that is driven from a port on the interface of the specified domain. See Usage Notes , below.
-sink	N	Y	Selects the ports driving a net that fans out to a port on the interface of the specified domain. See Usage Notes , below.
-transitive	N	N	Not supported.

Usage Notes

When -source and -sink are specified, a port is included if it has a source as specified or a sink as specified.

For a selected output port on the interface of a domain for which this strategy is specified, level-shifting is performed only on the subset of the fanout that drives an element in the domain specified by the -sink option.

For a selected input port on the interface of a domain for which this strategy is specified, level-shifting is performed only on the subset of the fanin driven by an element in the domain specified by the -source option.

Notes about -location:

set_partial_on_translation

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
OFF FULL_ON	OFF defines a default translation from PARTIAL_ON to OFF when evaluating the power state of supply sets and power domains (default). FULL_ON defines a default translation from PARTIAL_ON to FULL_ON when evaluating the power state of supply sets and power domains.
-full_on_tools	Defines a list of tools (products) for which translation of PARTIAL_ON to FULL_ON take place.
-off_tools	Defines a list of tools for which translation of PARTIAL_ON to OFF take place.

Usage Notes

The set_partial_on_translation command defines the translation of PARTIAL_ON to FULL_ON or OFF for purposes of evaluating the power state of supply sets and power domains. The state of a supply set is evaluated after the tool-specific translation of PARTIAL_ON to FULL_ON or OFF for each supply net in the set.

By default Power Aware translates PARTIAL_ON is translated to OFF. Therefore, you must use the set_partial_on_translation command to change the default translation behavior to FULL_ON.

You can also define a list of tools for which translation of PARTIAL_ON to FULL_ON takes place and the tools for which PARTIAL_ON to OFF takes place. All tool names are case-insensitive

You can also specify default behavior for an unlisted tool.

Note



To define PARTIAL_ON translation behavior for Power Aware simulation, specify 'questa' in the tools list. In Power Aware simulation, PARTIAL_ON has an enum value of 2. The supply_partial_on function would also assign the same enum value of 2.

Example C-7. Set the Translation of PARTIAL_ON to FULL_ON For All Tools

```
upf_version 2.0
set_scope tb
...
set_partial_on_translation FULL_ON
...
```

Example C-8. Set the translation of PARTIAL_ON to FULL_ON Only for ModelSim and to OFF for Others

```
upf_version 2.0
set_scope tb
...
set_partial_on_translation OFF -full_on_tools questa
...
```

Example C-9. Set the translation of PARTIAL_ON to OFF Only for ModelSim and to FULL_ON for Others

```
upf_version 2.0
set_scope tb
...
set_partial_on_translation FULL_ON -off_tools questa
...
```

Example C-10. Error/Warning Conditions: Same String in Different Lists

It is an error if the same string for a tool name occurs in both the `-full_on_tools` and `-off_tools` lists. In the following example for Power Aware simulation, the error occur only if `questa` tool name is specified in both the `-full_on_tools` and `-off_tools` string_lists.

```
upf_version 2.0
set_scope tb
...
set_partial_on_translation OFF -full_on_tools {questa} -off_tools questa
...
```

Vopt Message

```
** Error: test.upf(5): UPF: (vopt-9762) The same string occurs in both the
-full_on_tools and -off_tools string_lists.
```

```
Usage: set_partial_on_translation [OFF | FULL_ON] [-full_on_tools  
{string_list}] [-off_tools {string_list}]
```

Example C-11. Error/Warning Conditions: set_partial_on_translation Invoked More Than Once

A Warning message is issued if set_partial_on_translation is invoked more than once.

```
...  
upf_version 2.0  
set_scope tb  
...  
set_partial_on_translation OFF -full_on_tools {questa}  
set_partial_on_translation FULL_ON  
...
```

Vopt Message

```
** Warning: test.upf(6): UPF: (vopt-9763) Command  
'set_partial_on_translation' invoked more than once. Ignoring previous  
specification(s) of this command.
```

set_pin_related_supply

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
-pins	
-related_power_pin	
-related_ground_pin	

Usage Notes

- Questa SIM assumes the driver/receiver logic supply to be the same as specified related supplies—that is, corruption, isolation, and level-shifting behavior is in accordance with the specified related supplies. When the supply specified using `set_pin_related_supply` (using `-related_power_pin` or `-related_ground_pin` is a different supply than that of actual driver or receiver logic, Power Aware simulation gives a `vopt-9814` error message (`vopt-9814`).

You can use the `-warning` argument of `vopt` to change the severity of this message to a warning so that simulation may continue:

```
vopt -warning 9814
```

Refer to [Power Aware Messages](#) for more information on changing the level of message severity.

set_port_attributes

Support for UPF Standard

v1.0 — no

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-ports	N	Y	This option takes precedence over any other set_port_attributes with any other option, such as -elements, -domains, or -model.
-domains	N	Y	
-elements	N	Y	Complex data types of SystemVerilog, such as structs, are not supported.
-applies_to	N	Y	
-model	N	Y	
-attribute	N	Y	
-clamp_value	N	Y	
-sink_off_clamp	N	Y	
-source_off_clamp	N	Y	
-receiver_supply	N	Y	
-driver_supply	N	Y	
-related_power_port	N	Y	
-related_ground_port	N	Y	
-related_bias_ports	N	N	
-repeater_supply	N	Y	
-pg_type	N	Y	No support for -pg_type for UPF-created supply ports.
-exclude_domains	N	N	Not supported.
-exclude_elements	N	N	Not supported.
-exclude_ports	N	N	Not supported.
-transitive	N	N	Not supported.

Usage Notes

- Power Aware Simulation does not support the specification of an attribute on an index or record field specification as appropriate for the type of an object. Specifically, bit and part selects for Verilog, VHDL indexed name (for array elements), slice names (for array subranges), or selected names (for record elements).
- The `-clamp_value`, `-source_off_clamp`, and `-sink_off_clamp` arguments affect the filtering of ports specified by the `set_isolation` command.
- Power Aware simulation assumes the driver/receiver logic supply to be the same as specified related supplies—that is, corruption, isolation, and level-shifting behavior is in accordance with the specified related supplies. When the supply specified using `set_port_attributes` (using `-related_power_port` or `-related_ground_port`) is a different supply than that of actual driver or receiver logic, Power Aware simulation gives a vopt error message (vopt-9814).

You can use the `-warning` argument of vopt to change the severity of this message to a warning so that simulation may continue:

```
vopt -warning 9814
```

Refer to [Power Aware Messages](#) for more information on changing the level of message severity.

Example

```
set_port_attributes -ports top/out -source_off clamp_1
```

set_power_switch

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-output_supply_port	Y	Y	
-input_supply_port	Y	Y	
-control_port	Y	Y	
-on_state	Y	Y	
-supply_set	N	Y	When the supply set simstate is anything other than NORMAL, the state of the output supply port of a switch is OFF and the acknowledge ports are corrupted. Note that this differs from UPF v2.0, which states that the state of the output supply port of a switch is UNDETERMINED.
-on_partial_state	Y	Y	
-off_state	Y	Y	
-error_state	Y	Y	

Usage Notes

The following is not supported:

- In order to model the definition of a power switch, the tool currently treats UNDETERMINED state as OFF state for simulation. This implies that when the state of supply set is NOT_NORMAL, the state on the output supply port will be OFF instead of UNDETERMINED state.

set_retention

Support for UPF Standard

- v1.0 — yes (partial)
- v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-elements	Y	Y	
-retention_power_net	Y	Y	
-retention_ground_net	Y	Y	
-retention_supply_set	N	Y	Powers the register holding the retained value.
-no_retention	N	Y	Specifies that storage elements specified by the retention strategy do not have retention capability added.
-use_retention_as_primary	N	Y	Powers the storage element and the output drivers of the register using the retention supply.
-save_signal	N	Y	
-restore_signal	N	Y	
-instance	N	Y	Recognizes the special attributes present on the specified instance, applies the appropriate simulation semantics, and makes the connections: Refer to Usage Notes for set_retention for more information
-update	N	Y	Allows adding elements and supplies to a previously created power domain.
-exclude_elements	N	N	Not supported.
-restore_condition	N	Y	
-retention_condition	N	Y	
-save_condition	N	Y	
-parameters	N	N	Not supported.
-transitive	N	N	Not supported.

Usage Notes for set_retention

Restrictions:

- Corruption of retention element and saved value is not supported.
- The implicit corruption semantics are also applied to the shadow latch used to preserve the data during retention period. In order to remove the shadow latch from corruption, you must specify it in an exclude file (vopt -pa_excludefile).

Notes about -instance:

- If there is a port of type pg_type present on the instance, then Power Aware simulation automatically connects the primary_power and primary_ground pins with primary power and primary ground nets of the power domain. It connects the backup power and backup ground pin specified on the instance with the retention power and ground nets specified in the strategy.
- If there is no port of type pg_type present on the instance, then Power Aware simulation applies implicit corruption semantics according to the primary_power and ground nets specified for the power domain.

set_retention_control

Support for UPF Standard

v1.0 — yes (partial)

v2.0 — yes (partial)

Arguments

Argument	v1.0	v2.0	Comments/Restrictions
-domain	Y	Y	
-save_signal	Y	Y	
-restore_signal	Y	Y	
-assert_r_mutex	N	N	Not supported.
-assert_s_mutex	N	N	Not supported.
-assert_rs_mutex	N	N	Not supported.

set_scope

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

Argument	Comments/Restrictions
<instance> <pathname>	

The syntax of `set_scope` in UPF only allows hierarchical path up to instances (design elements). This has been extended under the above mentioned switch to accept hierarchical path of generate blocks as well. This allows user to create power domains inside a generate blocks. e.g.

```
set_scope mid/forgen\[1\  
create_power_domain PD_forgen -include_scope
```

set_simstate_behavior

Support for UPF Standard

v1.0 — no

v2.0 — yes

Arguments

Argument	Comments/Restrictions
ENABLE DISABLE	<p>ENABLE applies simstate simulation semantics for every supply set automatically connected to an instance of a model. Elements implicitly connected to a particular supply set have simstate semantics enabled by default.</p> <p>DISABLE disables simstate simulation semantics recursively for all the descendants of the instance of the model. Elements automatically or explicitly connected to a particular supply set have simstate semantics disabled by default.</p>
-lib	<p>When both -lib and -model are specified, the simstate behavior is defined for the specified models in the specified library.</p> <p>If -model is not defined and -lib is specified, the simstate behavior is defined for all models in the specified library.</p> <p>The -elements argument has higher priority over -model and -lib.</p>
-model	
-elements <.list>	<p>(ModelSim extension) Optional argument not specified in UPF v2.0.</p> <p>You specify a list of element names as the value to this argument, which determines the simulation simstate behavior for one or more design elements.</p> <p>Because this is a non-standard argument, you can specify this information through the use of vopt -pa_tcl, which allows your UPF file to remain portable.</p>

Usage Notes

The `set_simstate_behavior` command defines the simulation simstate behavior for a model or library. You can use this command to override the default setting of simstate semantics.

Examples

Example C-12. Enable Simstate Behavior Using `set_simstate_behavior`

In the following example, automatic connections led to disablement of simstate semantics of model `mid`. The `set_simstate_behavior` command enables the simstate semantics of instances of the model.

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -model {mid}
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
```

Example C-13. Disable Simstate Behavior Using `set_simstate_behavior`

In the following example, `set_simstate_behavior` disables the simulation semantics all instances of model `bot`.

```
set_scope tb/top1
set_simstate_behavior ENABLE -model {mid} -elements {mid3}
set_simstate_behavior DISABLE -model {bot}
...
create_power_domain pd1
create_supply_net VDD_N -domain pd1
...
```

Vopt Message

```
** Note: test.upf(7): UPF: (vopt-9693) Power Aware simulation semantics
disabled for /tb/top1/mid1/bot1.
** Note: test.upf(7): UPF: (vopt-9693) Power Aware simulation semantics
disabled for /tb/top1/mid1/bot2.
** Note: test.upf(7): UPF: (vopt-9693) Power Aware simulation semantics
disabled for /tb/top1/mid2/bot1.
** Note: test.upf(7): UPF: (vopt-9693) Power Aware simulation semantics
disabled for /tb/top1/mi
```



Tip: You can suppress the above messages by using the following argument with the vopt command: vopt -suppress 9693

Example C-14. When Both -models and -lib Arguments Are Specified

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -model {mid} -lib lib_name1
set_simstate_behavior DISABLE -model {top} -lib lib_name2
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
...
```

Example C-15. Specify Simstate Behavior For All Models in a Library

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -lib lib_name1
set_simstate_behavior DISABLE -lib lib_name2
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
...
```

Example C-16. Use -elements Argument to Override or Specify Simstate Behavior (single element)

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -elements /tb/top1/mid1
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
```

Example C-17. Use -elements Argument to Override or Specify Simstate Behavior (list of elements)

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -model {mid}
set_simstate_behavior DISABLE -elements {mid1 mid2}
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
```

Example C-18. Error/Warning Conditions: -lib Argument Without a Library

It is an error if the specified library with -lib argument does not exist.

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
set_simstate_behavior ENABLE -lib lib1
...
```

Vopt Message

```
** Error: test.upf(4): UPF: (vopt-9753) Library 'lib1' does not exist.
```

Example C-19. Error/Warning Conditions: -model Argument Without a Model

It is an error if the specified model (using -model argument) does not exist in the specified library (using -lib argument).

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
set_simstate_behavior ENABLE -model mad -lib work
...
```

Vopt Message

```
** Error: test.upf(4): UPF: (vopt-9655) Model: 'mad' doesn't exist in
library: 'work'.
```

Example C-20. Error/Warning Conditions: Conflicting Simstate Behaviors

It is an error if a model has conflicting simstate behaviors specified.

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior ENABLE -model { mid top }
set_simstate_behavior DISABLE -model { mid }
...
```

Vopt Message

```
** Error: (vopt-9730) Attribute:'upf_simstate_behavior DISABLE' on design
object:'mid1' conflict
```

Example C-21. Error/Warning Conditions: DISABLE Argument Used Without Supply Ports

It is an error if DISABLE is specified and the model has no supply ports.

```
upf_version 2.0
set_scope tb/top1
create_power_domain pd1
...
set_simstate_behavior DISABLE -model {mid}
...
create_supply_net VDD_N -domain pd1
create_supply_net GND_N -domain pd1
...
```

Vopt Message

```
** Error: test.upf(6): UPF: (vopt-9756) Power aware simulation semantics
cannot be disabled for design element '/tb/top1/mid1'.
** Error: test.upf(6): UPF: (vopt-9756) Power aware simulation semantics
cannot be disabled for
```

upf_version

Support for UPF Standard

v1.0 — yes

v2.0 — yes

Arguments

none

Supported UPF Package Functions

Power Aware simulation Supports the UPF 2.0 package that is part of IEEE Std 1801™-2009.

The following tables list the UPF 2.0 package functions supported for VHDL ([Table C-2](#)) and SystemVerilog ([Table C-3](#)). Refer to IEEE Std 1801™-2009 for complete details

Table C-2. Supported UPF Package Functions for VHDL

Function Name
function supply_on (pad_name : IN string ; value : IN real) return boolean;
function supply_off (pad_name : IN string) return boolean;
function supply_partial_on (pad_name : IN string; value : real) return boolean;
function get_supply_value (pad_name : IN string) return supply_net_type;
function get_supply_voltage (value : IN supply_net_type) return real;
function get_supply_on_state (value : IN supply_net_type) return boolean;
function get_supply_on_state (value : IN supply_net_type) return bit;
function get_supply_state (value : IN supply_net_type) return net_state;

Table C-3. Supported UPF Package Functions for SystemVerilog

Function Name
function bit supply_on(string pad_name, real value);
function bit supply_off(string pad_name);
function bit supply_partial_on(string pad_name, real value);
function supply_net_type get_supply_value(string name);
function real get_supply_voltage(supply_net_type arg);
function bit get_supply_on_state(supply_net_type arg);

Table C-3. Supported UPF Package Functions for SystemVerilog

Function Name
function bit [1:0] get_supply_state(supply_net_type arg);

Note



UPF 2.0 introduced “state” as a reserved keyword.

Detailed Support for supply_net_type

For UPF 2.0, the definition of **supply_net_type** was redefined from UPF 1.0 (Accellera UPF Standard 1.0). Key changes include:

- ON state changed to FULL_ON. Ensure you rewrite any UPF v1.0 files to match this change.
- Type state changed from 32-bit vector to an enum.
- Addition of the UNDETERMINED state for **supply_net_type**, which has the following impact on the power aware functionality. Ensure you rewrite any UPF v1.0 files to take account of this new state.
 - Power Aware switches can go to the UNDETERMINED state, as described in the following situations (defined in the IEEE Std 1801-2009, Section 6.20).
 - Multiple-on or partial-on conditions are true with different root supply drivers.
 - Both off_state and on_state conditions or on_partial_state condition are true.
 - Any error state condition is true.
 - Control signals go to x or z.
 - Simstate of the supply set associated with the switch is anything other than NORMAL.

For UPF 1.0, under these conditions the output port of the switch would go to the OFF state. You can switch to UPF 1.0 type functionality with the vopt - pa_upfversion=1.0 option, which sets the upf_version of the UPF file to UPF 1.0

Due to introduction of this UNDETERMINED state, switch output may go to UNDETERMINED state in delta periods if any of these conditions is true due to input delta activities

- A supply net may go to UNDETERMINED state as defined by resolution methods provided for create_supply_net command (defined in the IEEE Std 1801-2009, Section 6.22.2). Specifically, for any error condition in resolution techniques, such as:

- For ONE_HOT resolution, if at any point in time more than one supply source that is driving the net is anything other than OFF
- For PARALLEL resolution, if any of the potentially conducting paths can be traced to more than one root supply driver

For UPF 1.0, under these conditions the state of the supply net was set to OFF.

- For UPF 2.0, the default state of supply nets and ports is set to OFF. You can change this to FULL_ON with the vopt option `-pa_disable=defaultoff`.

Accessing Generate Blocks in UPF

In order to access hierarchy as defined in individual languages or as Power Aware simulation accepts it, UPF uses square bracket pairs, []. This is because UPF is written in Tcl, so in some contexts square brackets can be interpreted as special command substitution characters—though you usually need to use escape characters with them for that purpose. The exception to this when using a UPF command argument that takes a list of values enclosed in braces, { }, because that is not actually command substitution.

The following examples show how to generate references in UPF for both Verilog and VHDL.

```
create_power_domain pd -elements { top_vh/for_genvh__1/bot_vh_inst  
top_vl/for_gen_vl[1]/bot_vl_inst }
```

```
create_power_domain pd -elements { top_vh/for_genvh(1)/bot_vh_inst  
top_vl/for_gen_vl[1]/bot_vl_inst }
```

Limitation

Connections from UPF to HDL port of inout type are not supported for inout capability. These connections are made in such a way that UPF is driving data to the HDL port.

Supported UPF Attributes

Power Aware simulation supports the use of UPF attributes used to express the power intent in an HDL model. You can specify these attributes using the following methods:

- UPF commands (`set_design_attributes`, `set_port_attributes`, `set_isolation`, `set_pin_related_supply`, `set_simstate_behavior`, `set_retention_elements`)

- VHDL or SystemVerilog attributes
- Liberty cell specification

Table C-4 lists the UPF attributes supported by Power Aware simulation.

Table C-4. Supported UPF Attributes

HDL Attribute	Value(s)	UPF Command
UPF_clamp_value	0 1 Z latch any <value>	set_isolation -clamp_value set_port_attributes -clamp_value
UPF_sink_off_clamp_value	0 1 Z latch any <value>	set_isolation -sink_off_clamp_value set_port_attributes -sink_off_clamp_value
UPF_source_off_clamp_value	0 1 Z latch any <value>	set_isolation -source_off_clamp_value set_port_attributes -source_off_clamp_value
UPF_pg_type	<pg_type_value>	set_port_attributes -pg_type
UPF_related_ground_pin	<port_name>	set_pin_related_supply -related_ground_pin set_port_attributes -related_ground_port
UPF_related_power_pin	<port_name>	set_pin_related_supply -related_power_pin set_port_attributes -related_power_port
UPF_related_bias_pin	<port_name>	set_port_attributes -related_bias_port
UPF_retention	required optional	set_retention_elements -retention
UPF_simstate_behavior	ENABLE DISABLE	set_simstate_behavior

Specifying Attributes

You can use the `set_design_attributes` and `set_port_attributes` commands to specify attributes for Power Aware simulation. The following arguments for these UPF commands are now supported:

- `set_design_attributes`

- elements
- models
- attribute
- set_port_attributes
 - ports
 - domains
 - elements
 - model
 - attribute
 - clamp_value
 - sink_off_clamp_value
 - source_off_clamp_value
 - related_power_port
 - related_ground_port
 - pg_type

Limitations

- In some cases, -related_power_port/-related_ground_port might not work properly with -model.
- -pg_type might not work on UPF-created supply_ports/nets.
- -source_off_clamp_value -sink_off_clamp_value -clamp_value only affects the filtering of ports affected by the set_isolation command.

Attributes in VHDL or SystemVerilog

You can also specify all supported attributes using VHDL or SystemVerilog.

VHDL example:

```
(attribute UPF_pg_type of vdd_backup : signal is "backup_power"
```

System Verilog example:

```
(* UPF_pg_type = "backup_power" *) input vdd_backup;
```

Specifying Supply Nets in UPF

The connect_supply_net command specifies connection to a supply net that conforms to the behavior described in 6.13 of IEEE Std 1801-2009. A UPF supply net is propagated through implicitly created ports and throughout the logic hierarchy of the scope in which the net is created.

Format of Assigned Net Values

A supply net connected by this command is a composite signal consisting of a real voltage value (in μV) and an enumerated supply state (ON, OFF). For Power Aware simulation, the default voltage value is $81\mu\text{V}$ on both VDD and VSS / GND nets (a voltage value is not used for dynamic simulation).

For simulating a digital design, the more important information for the net is whether its supply state is ON or OFF. Regardless of whether the power supply net is VDD or GND, when the power supply state is ON, the state value assumes an integer value of 1. When the supply state is OFF, the state value assumes an integer value of 0. By default, the power state is ON.

Consequently, Power Aware simulation reports the two default values of 81 and 1 to a power net as 81_0000000001. Note that the output of a switch in UPF whose control port has been driven to the OFF state is always $0\mu\text{V}$ and a 0 supply state (reported by Power Aware simulation as 0_0000000000).

Changing the Default Supply State Values for VHDL Models

When you connect a supply net to an HDL port to extend the UPF supply network directly to an HDL model, Power Aware simulation connects the net to the HDL port and converts the enumerated supply state to the HDL port type.

By default, Power Aware simulation performs the following conversion:

```
{supply_state_type.state = ON} => {HDL bit type = 1'b1}  
{supply_state_type.state = OFF} => {HDL bit type = 1'b0}
```

However, for VHDL models in the design (such as memory models), this results in a logical '1' on ground and VSS nets.

To change this default conversion, you need to define a UPF-to-VHDL Value Conversion Table (VCT) from that converts UPF supply states to their corresponding VHDL bit values. You then need to apply the conversion when connecting the supply net in the UPF file.

1. Define a Value Change Table for UPF-to-VHDL conversion, by doing either of the following:
 - Use the `create_upf2hdl_vct` command to create a VCT that defines the conversion. For example:

```
create_upf2hdl_vct upf2vhdl_vss  
-hdl_type {vhdl std_logic}  
-table {{OFF 1} {ON 0} {PARTIAL_ON X}}
```

where `upf2vhdl_vss` is the name of the table, `-hdl_type` specifies VHDL `std_logic` for the data type and `-table` maps OFF to 1, ON to 0, and PARTIAL_ON to X. Note that this reverses the default conversion values.

- Use a predefined VCT provided by Questa (see [Predefined VCTs Supported from the UPF Standard](#)) to perform the conversion mapping. In this example, the `UPF_GNDZERO2VHDL_SL` table performs this conversion.

Note

Using a predefined VCT is easier, but it requires implementation of UPF 2.0 (IEEE Std 1801-2009), so both methods are shown here.

2. Implement the VCT conversion in your UPF file by using the `-vct` argument of the `connect_supply_net` command. Continuing with the examples from Step 1, use either one of the following:

- Assign the name of the VCT you defined (`upf2vhdl_vss`) to the `-vct` argument:

```
connect_supply_net power_supply_net { HDL_port .. } -vct upf2vhdl_vss
```

- Assign the name of the predefined VCT you selected (`UPF_GNDZERO2VHDL_SL`) to the `-vct` argument:

```
connect_supply_net power_supply_net { HDL_port .. }  
-vct UPF_GNDZERO2VHDL_SL
```

Supported UPF Extensions

Using `-pa_upfextensions`

The `-pa_upfextensions` argument to the `vopt` command allows you to define and apply various UPF behaviors that are not supported by the current standard (UPF v2.0). [Table C-5](#) lists the values for this argument that you can specify to override supported UPF behavior.

Syntax

```
vopt -pa_upfextensions=[default] | [all] | [ackport] | [altgenname] | [case] | [flathiername] |  
[genblk] | [ignorepgports] | [ignorepgportsaon] | [ignoresupply_expr] | [mapcells] |  
[nonameclash] | [nonlrmstatenames] | [relatedsnet] | [s] | [v] | [wildcard]
```

Description

- To specify more than one value for this argument, use the `+` operator between values (there is no order dependency when specifying multiple values). For example:

```
vopt -pa_upfextensions=relatedsnet+genblk+v
```

- To enable all values, specify the following:

```
vopt -pa_upfextensions=all
```

- To enable a limited number of values (see [default](#) in [Table C-5](#)), specify either of the following:

```
vopt -pa_upfextensions=default
vopt -pa_upfextensions
```

Table C-5. Power Aware Actions for vopt -pa_upfextensions

Value	Action
all	Enables (specifies) all values of the -pa_upfextensions argument.
ackport	Enforces the rule of the the ack_port of create_power_switch UPF command requires a mandatory supply set on the power switch in UPF 2.0. Disable this to revert to UPF 1.0 behavior.
altgenname	<p>Supports the synthesis style hierarchical paths for generate blocks. Power Aware simulation recognizes a hierarchical path with an escaped generate scope of the form that a synthesis tool generates, and maps such a name to a hierarchical name of conventional form.</p> <p>For example, each of the following styles—</p> <pre>{< prefix >}gen_label[index].name2 {/< suffix >} /* 'gen_label[index].name2' is the new name of the instance 'name2' within scope gen_label[index].*/</pre> <pre>{< prefix >}\\gen_label[index].name2 {/< suffix >} /* The new name could also be double- escaped. */</pre> <pre>{< prefix >.)gen_label[index].name2 {/< suffix >} /* Use '.' as a path separator for generate scopes */</pre> <p>would map to—</p> <pre>{< prefix >}gen_label[index]/name2 {/< suffix >}</pre>
case	Allows you to specify predefined option values like TRUE, FALSE, high, low, posedge, and latch in case-insensitive form

Table C-5. Power Aware Actions for vopt -pa_upfextensions (cont.)

Value	Action
default	<p>Enables (specifies) only the following values of the -pa_upfextensions argument:</p> <ul style="list-style-type: none"> • case • genblk • ignorepgports • mapcell • nonameclash • nonameclash • relatedsnet • s • v <p>NOTE: Specifying vopt -pa_upfextensions with no values has the same effect.</p>
flathiername	Allows hierarchical names within the UPF file that refer to flattened gate level hierarchies in the design.
genblk	Allows generate block to be used in set_scope , find_objects , and create_power_domain (-elements argument) commands.
ignorepgports	<p>Bypasses connection of a supply net to a port using the connect_supply_net command when the port is missing in the verification model but is a power or ground (PG) pin in the Liberty model.</p> <p>In this case, the connect_supply_net command to these ports is ignored.</p> <p>This is equivalent to using vopt -pa_connectpgpin=i</p>
ignorepgportsaon	<p>Bypasses connection of a supply net to a port using the connect_supply_net command when the port is missing in the verification model but is a power or ground (PG) pin in the Liberty model.</p> <p>In this case, connect_supply_net command to these ports is ignored and the Power Aware simulation semantics of the parent instance of the port are disabled.</p> <p>This is equivalent to using vopt -pa_connectpgpin=a</p>
ignoresupply_expr	Ignores supply_expr when both supply and logic expressions are present in an add_power_state command.

Table C-5. Power Aware Actions for vopt -pa_upfextensions (cont.)

Value	Action
mapcell	<p>Supports the use of -lib_cells specified in map_power_switch, map_isolation_cell and map_level_shifter_cell UPF commands in <libraryname/cellname> format. For example</p> <pre>map_power_switch PD_ONE -domain PD_SUP -lib_cells{LIB/NODE}</pre>
nonameclash	<p>Ignores the name clash error that occurs for ports of the set_power_switch command that are specified for the -input_supply_port or -output_supply_port arguments and that already exist in RTL.</p>
nonlrmstatenames	<p>Allows non-standard UPF names in state name of add_port_state command. For example:</p> <pre>add_port_state VN1 -state {1p1 1.0}</pre>
relatedsnet	<p>Supports the behavior for the Tcl command set_related_supply_net.</p>
s	<p>Allows relative paths in set_scope command. For example:</p> <pre>set_scope ../../</pre>
v	<p>Allows automatic insertion of vct for pins detected as power and ground pins.</p>
mapcell	<p>Supports the use of -lib_cells specified in map_power_switch, map_isolation_cell and map_level_shifter_cell UPF commands in <libraryname/cellname> format. For example</p> <pre>map_power_switch PD_ONE -domain PD_SUP -lib_cells{LIB/NODE}</pre>

Table C-5. Power Aware Actions for vopt -pa_upfextensions (cont.)

Value	Action
wildcard	<p>Supports the use of the wildcard character (*) to the following UPF commands:</p> <ul style="list-style-type: none"> • add_domain_elements -elements • bind_checker -elements • connect_logic_net -ports • connect_supply_net -ports • connect_supply_set -elements • create_power_domain -elements -exclude_elements • map_isolation_cell -elements • map_level_shifter_cell -elements • map_retention_cell -elements • set_design_attribute -elements • set_isolation -elements • set_level_shifter -elements • set_retention -elements • set_pin_related_supply -pins • set_related_supply_net -objects • set_port_attribute -elements -ports

UPF Supply Connections

Supply connections between various UPF objects can be made between supply nets, supply sets, supply ports, power switches and UPF objects such as retention, isolation, level shifter cells. You can define these connections in the following ways:

- [Implicit Connections](#)
- [Explicit Connections](#)
- [Automatic Connections](#)
- [Power State Composition](#)

Implicit Connections

Implicit connections provide a way to connect supply nets to elements that do not have supply ports. Any design element that is present in the extent of power domain and does not have supply ports connected or is excluded from Power Aware processing (-pa_excludefile) will be implicitly connected with the primary supplies (power and ground) of the power domain. Thus, the corruption of that element will depend on the state of the primary power net and primary ground net of the power domain.

Explicit Connections

You can explicitly connect a supply net to a supply port using the following UPF command:

```
connect_supply_net
```

This explicit connection overrides (has higher precedence than) the implicit and automatic connection semantics that might otherwise apply.

Explicit connections include:

- Connections to UPF-created supply ports
- Connections to HDL-created supply ports (supply_net_type or 1-bit type)
- Connections to supply ports of power switches

Explicit connection to an HDL supply port has simulation semantics disabled by default (see “[Simulation Semantics for UPF Supply Connections](#)”). Driving an HDL port from UPF overrides its RTL connection.

Explicit Connections to HDL Ports

Using the connect_supply_net command, you can connect a UPF-created supply net to an HDL-created supply port and a UPF-created supply port to an HDL-created supply net. To ensure the supply net state and voltage values are propagated and modeled in the various HDLs, you must use the supply_net_type datatype. You can make these datatypes visible by importing UPF defined HDL packages (refer to Appendix B in UPF v1.0 and Annex B in IEEE Std 1801-2009).

Examples

HDL specification (Verilog):

```
module memory(input supply_net_type vdd, ..);  
  ...  
endmodule
```

HDL specification (VHDL):

```
entity memory is  
  port (vdd : in supply_net_type; ..)  
  ...  
end entity
```

UPF specification:

```
connect_supply_net vdd_switchable -ports mem/vdd
```

Explicit Connections to 1-bit HDL Ports

UPF also allows supply connections between supply nets and 1-bit Verilog/VHDL ports for building simple functional models. In these cases, HDL supply ports are connected to the ON/OFF state bit of the supply net. For complex modeling, you can use value change tables (VCTs) for the conversion from the supply net state to values relevant to an HDL type, or vice-versa.

Limitation

Enumerated HDL types are not supported.

Examples

HDL specification (Verilog):

```
module memory(input vdd, ..);  
  ...  
endmodule
```

HDL specification (VHDL):

```
entity memory is  
  port (vdd : in std_logic; ..)  
  ..  
end entity
```

UPF specification:

```
connect_supply_net vdd_switchable -ports mem/vdd
```

Explicit Connections to Supply Ports of Power Switch

You can use the `connect_supply_net` command to define connections to supply ports of power switch. For example:

```
create_power_switch SW \  
  -input_supply_ports {IN_SW} \  
  ..  
connect_supply_net VDD_IN -ports {SW.IN_SW}
```

You can also specify connections using the `create_power_switch` command.
For example:

```
create_power_switch SW \  
  -input_supply_ports {IN_SW VDD_IN} \  
  ..
```

Automatic Connections

Questa supports the UPF automatic connection semantics of supply nets and supply sets as defined by IEEE Std 1801-2009 (UPF v2.0).

Automatic connections semantics are defined for:

- Supply nets using the `connect_supply_net` command
 (see “[Automatic Connections for Supply Nets](#)”)
- Supply sets using either the `connect_supply_set` or `create_power_domain` command
 (see “[Automatic Connections for Supply Sets](#)”)

The necessary conditions for the supply nets (or supply nets of supply sets) to automatically connected to the supply ports of design elements are:

- Design element has a port with `-pg_type` attribute. The `pg_type` attribute includes
 - String type HDL attribute named either `pg_type` or `UPF_pg_type`.
 - Implementation library model with `pg_type` attribute.
- Value of the `pg_type` attribute of the port matches with the `pg_type` value specified using UPF commands.

Simulation semantics of all the design elements whose ports are automatically connected to power supply are disabled (see “[Simulation Semantics for UPF Supply Connections](#)”).

Automatic Connections for Supply Nets

You can define automatic connection semantics on individual supply nets using the following UPF command:

```
connect_supply_net -pg_type -domain -cells
```

Questa will automatically connect the specified supply net with the supply ports on the specified cell or the design elements within the extent of power domain which fulfills the necessary conditions for automatic connection semantics. You can also specify Value Conversion Tables (VCTs) for automatic connection semantics by using the `-vct` argument.

Using the `-vct` argument has a restriction that all the ports that are connected using `connect_supply_net` should have matching types, since the type matching for VCT will produce error during connections.

For example:

```
connect_supply_net snet -domain PD -pg_type primary_power -vct my_sv_vct
```

This will produce an error message when domain PD has a VHDL instance contain supply port with `pg_type primary_power` and different type and create the connection, assuming a 1-bit connection.

Command Syntax

UPF v1.0:

```
connect_supply_net net_name
  [-ports list] [-pins list]
  [<-cells list | -domain domain_name>]
  [<-rail_connection rail_type | -pg_type pg_type>]*
  [-vct vct_name]
```

UPF2.0:

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

Currently, the `connect_supply_net` command in UPF2.0 is modeled as its equivalent command in UPF v1.0. That is, `connect_supply_net` command in UPF v2.0 will not accept `element_list` in `-pg_type` argument.

Examples

```
connect_supply_net VDD -domain PD_SW -pg_type primary_power
connect_supply_net Vdd_backup -cells {RET_CELL} -pg_type backup_power
connect_supply_net VSS -domain PD_SW -pg_type primary_ground -vct
  UPF_GNDZERO2SV_LOGIC.
```

Automatic Connections for Supply Sets

You can define automatic connection semantics on supply sets using either of the following UPF commands:

```
connect_supply_set -connect
create_power_domain -define_func_type
```

Power Aware simulation automatically connects supply nets of supply sets to the supply ports of design elements. This connection is based on the purpose of the supply set in a given domain or strategy context and the function that a supply net performs in the context of supply set.

Limitations

- Supply sets specified in strategy context will not be automatically connected to design elements owing to separate infrastructure in `-instance`.
- Supply nets in supply sets functioning as predefined supply set functions are not automatically connected as per their predefined function. Explicit specification of automatic connections must be specified.

Command Syntax

Using `connect_supply_set`:

```
connect_supply_set supply_set_ref
  {-connect {supply_function {pg_type_list}}}*
  [-elements element_list]
  [-exclude_elements exclude_list]
  [-transitive ]
```

Using `create_power_domain`:

```
create_power_domain
  [-define_func_type {supply_function {pg_type_list}}]*
```

Examples

```
create_power_domain PD \
  -define_func_type {power primary_power} \
  -define_func_type {always_on backup_power} \
  ..

connect_supply_set PD.primary \
  -connect {power primary_power} \
  -elements TOP

connect_supply_set PD.ISO.isolation_supply_set \
  -connect {iso_power primary_power} \
  -connect {iso_ground primary_ground} \

connect_supply_set PD.RET.retention_supply_set \
  -connect {ret_backup_power backup_power} \
  -connect {switchtable_supply primary_power}
```

Power State Composition

States of ports and nets carry voltage and on/off information, which means combinations of power states of supply ports and/or nets are important in identifying the requirement of isolation cells and level shifters on boundaries between two power domains. Because a power state added on one domain/supply set can reference other power states added on another domain/supply set, Power Aware simulation can statically infer such dependencies. These dependencies help in determining the combinations of power states of supply nets and/or ports.

You can add composite power states to your design as follows:

- Supply net — power state information (state and voltage level) is defined by the `supply_net_type` declaration of the HDL cell (see [Explicit Connections to HDL Ports](#)).
- Supply port — power state information (state and voltage level) is defined by the `supply_net_type` declaration of the HDL cell (see [Explicit Connections to HDL Ports](#)).
- Supply set — composed of two or more supply nets, so its power state is specified in terms of those supply nets.
- Power domain — power state is determined by the state of supply sets associated with the domain.

Power Aware simulation performs level shifter and isolation cell analysis/checks from information provided with `add_power_state` UPF commands and also reports the power domain state dependencies in report files.

To know whether an isolation or level-shifting is required on a boundary between power domains, a static analysis is required to determine state dependencies between these power domains (or supply sets associated with these domains). The dependency information is extracted from `supply/logic` expression (mentioned in `add_power_state` command) of various power states from power domains or supply sets.

A valid combination of two power states of domains/supply sets is equivalent to a valid combination of power states of certain supply ports/supply nets (state and voltages). Power Aware simulation uses this net state and voltage information to determine whether isolation or level-shifting is required or not.

Note



For the cases where Power Aware simulation is statically not able to determine whether the state combination of two power domain/supply sets is invalid (or these power states cannot co-exist at any point of time), Power Aware simulation assumes it to be a valid state combination.

In the following example for static analysis, Power Aware simulation assumes that `PD1_ON` and `PD2_ON` may co-exist at some point of time.

```
add_power_state PD1 -state PD1_ON
  {-supply_expr {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD2 -state PD2_ON
  {-supply_expr {VDD2 == FULL_ON && GND2 == FULL_ON}}
```

This information corresponds that (VDD1 is FULL_ON, GND1 is FULL_ON, VDD2 is FULL_ON, GND2 is FULL_ON) may exist at some point of time.

Note



For better static analysis, it is recommended to include the state information of primary nets of power domain in supply expression of `add_power_state` for power domain (or its associated supply set).

Also, to determine the state relation between two power domains/supply sets, Power Aware simulation uses the information specified in power states of those two power domains/supply sets.

For example:

```
create_supply_net VDD ...
create_supply_net GND ...
set_domain_supply_net PD1 -primary_power_net VDD -primary_ground_net GND
add_power_state PD1 -state PD1_ON {-logic_expr {PD2 == PD2_OFF}
  -supply_expr {VDD == FULL_ON && GND == FULL_ON}}
```

Determining State Dependency with `add_power_state` Arguments

You can use the following arguments of the UPF `add_power_state` command to determine whether power states of different power domains can co-exist:

- `-logic_expr`
- `-supply_expr`

The following examples show various ways of using these arguments.

Example C-22. Dependency Specified with `add_power_state -logic_expr` — Case 1

In this example, from `logic_expr` of PD1_S1 Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist.

```
add_power_state PD1 -state PD1_S1 {-logic_expr {PD2 != PD2_S2}
  -supply_expr {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD2 -state PD2_S2 {-supply_expr {VDD1 == FULL_ON &&
```

```
GND1 == FULL_ON}}
```

Example C-23. Dependency Specified with `add_power_state -logic_expr` — Case 2

In this example, from `logic_expr` of PD1_S1 and PD2_S2 Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist, as one requires `ctrl` to be 1 and other requires it to be 0.

```
add_power_state PD1 -state PD1_S1 {-logic_expr {ctrl} -supply_expr
  {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD2 -state PD2_S2 {-logic_expr {!ctrl} -supply_expr
  {VDD1 == FULL_ON && GND1 == FULL_ON}}
```

Example C-24. Dependency Specified with `add_power_state -logic_expr` — Case 3

In this example, from `logic_expr` of PD1_S1, PD3_S3 Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist.

```
add_power_state PD1 -state PD1_S1 {-logic_expr {PD3 == PD3_S3}
  -supply_expr {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD2 -state PD2_S2 {-logic_expr {!ctrl} -supply_expr
  {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD3 -state PD3_S3 {-logic_expr {PD2 != PD2_S2}
  -supply_expr {VDD3 == FULL_ON && GND3 == FULL_ON}}
```

Example C-25. Dependency Specified with `add_power_state -logic_expr` — Case 4

In this example, from `logic_expr` of PD1_S1 and PD2_S2 Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist, as one requires `ctrl` to be 1 and other requires it to be 0.

```
add_power_state PD1 -state PD1_S1 {-logic_expr {ctrl == 1} -supply_expr
  {VDD1 == FULL_ON && GND1 == FULL_ON}}

add_power_state PD2 -state PD2_S2 {-logic_expr {!ctrl} -supply_expr
  {VDD1 == FULL_ON && GND1 == FULL_ON}}
```

Example C-26. Dependency Specified with `add_power_state -supply_expr` — Case 1

In this example, from `supply_expr` of PD1_S1 and PD2_S2 (VDD state) Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist.

```
add_power_state PD1 -state PD1_S1 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD == FULL_ON}}

add_power_state PD2 -state PD2_S2 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD == OFF}}
```

Example C-27. Dependency Specified with `add_power_state -supply_expr` — Case 2

In this example, from `supply_expr` of PD1_S1 and PD2_S2 (VDD state), Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist.

```
add_power_state PD1 -state PD1_S1 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD == FULL_ON}}
add_power_state PD2 -state PD2_S2 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD != FULL_ON}}
```

Example C-28. Dependency Specified with `add_power_state -supply_expr` — Case 3

In this example, from `supply_expr` of PD1_S1 and PD2_S2 (VDD voltage range), Power Aware simulation determines that state PD1_S1 and PD2_S2 cannot co-exist.

```
add_power_state PD1 -state PD1_S1 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD == '{FULL_ON, 0.8, 1.4}'}}
add_power_state PD2 -state PD2_S2 {-supply_expr {VDD1 == FULL_ON &&
GND1 == FULL_ON && VDD == '{FULL_ON, 1.6, 2.4}'}}
```

Power State Reporting

Power Aware simulation provides the following types of information in the report files:

- Information about the power state added on power domain/supply sets.
- Power State dependencies between connected power domains.

Example Report (Excerpt)

```

Power state info of Power domain : 'PD_tb'.
  1. Power state 'PD_tb_normal', File: src/testcase4/test.upf(15).
     Supply Expression : (((tb_pow)==('{FULL_ON , 3.30
})))&&(((tb_gnd)==('{FULL_ON , 1.00 }))).
     Function States :
       1. Ground(tb_gnd) == {FULL_ON,1.00}, Power(tb_pow) ==
          {FULL_ON,3.30}.

Power state info of Power domain : 'PD_mid2'.
  1. Power state 'PD_mid2_off', File: src/testcase4/test.upf(67).
     Supply Expression : (((mid2_MAIN_NET)==('{OFF
})))&&(((mid2_GND_NET)==('{OFF }))).
     Function States :
       1. Ground(mid2_GND_NET) == {OFF}, Power(mid2_MAIN_NET) ==
          {OFF}.

  2. Power state 'PD_mid2_normal', File: src/testcase4/test.upf(66).
     Supply Expression : (((mid2_MAIN_NET)==('{FULL_ON , 4.20
})))&&(((mid2_GND_NET)==('{FULL_ON , 1.30 }))).
     Function States :
       1. Ground(mid2_GND_NET) == {FULL_ON,1.30}, Power(mid2_MAIN_NET)
          == {FULL_ON,4.20}.

```

Power State Combinations for connected domains :

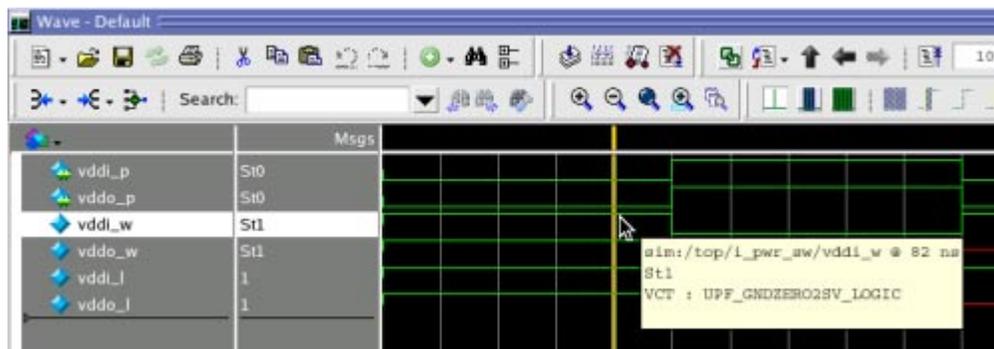
Power Domain 'PD_mid1'	Power Domain 'PD_wrapper1'
-----	-----
PD_mid1_off	PD_wrapper1_normal
Power Domain 'PD_mid1'	Power Domain 'PD_wrapper2'
-----	-----
PD_mid1_off	PD_wrapper2_normal
PD_mid1_normal	PD_wrapper2_normal
Power Domain 'PD_mid1'	Power Domain 'PD_mid3'
-----	-----
PD_mid1_off	PD_mid3_off
PD_mid1_off	PD_mid3_normal
PD_mid1_normal	PD_mid3_off
PD_mid1_normal	PD_mid3_normal
Power Domain 'PD_wrapper1'	Power Domain 'PD_mid3'
-----	-----
PD_wrapper1_normal	PD_mid3_off
PD_wrapper1_normal	PD_mid3_normal

Value Conversion Tables

A value conversion table (VCT) is a UPF definition (IEEE Std1801-2009) of how to convert, or map, a value from a supply net state relevant to an HDL variable type (and from an HDL variable type to a supply net state). This mapping enables complex modeling of connections between supply nets and RTL ports.

You can visualize which VCT is used for a given pin in the Wave window by hovering over the waveform for the port/net, as shown in [Figure C-1](#).

Figure C-1. Viewing VCT Information in the Wave Window



Using VCT Commands

You can use the `create_upf2hdl_vct` and `create_hdl2upf_vct` commands in the UPF file for modeling complex connections between a supply net and RTL port.

Power Aware simulation also automatically inserts VCT for pins detected as power and ground pins when you specify `vopt -pa_upfextensions`.

Examples

The following examples show the `create_upf2hdl_vct` and `create_hdl2upf_vct` commands in a UPF file.

```
create_hdl2upf_vct VHDL_SL2UPF \  
-hdl_type {vhdl} \  
-table {{'U' OFF} \  
      {'X' OFF} \  
      {'0' OFF} \  
      {'1' ON} \  
      {'Z' OFF} \  
      {'L' OFF} \  
      {'H' ON} \  
      {'W' OFF} \  
      {'-' OFF}}  
  
create_upf2hdl_vct UPF_GNDZERO2SV_LOGIC \  

```

```
-hdl_type sv \
-table {{PARTIAL_ON X} \
      {OFF 1} \
      {ON 0}}
```

Note

If VCT definition for a connection is not applicable or fails, then default 1-bit connection semantics will be applied. For example:

```
module memory (input gnd, ...);
...
endmodule
```

```
connect_supply_net gnd_switchable -ports mem_sv/gnd -vct
SV_LOGIC2UPF_GNDZERO
```

Data flow for the connection is UPF to RTL but VCT is specified as RTL to UPF. Hence, specified VCT is not applicable.

Limitations

Only the following HDL data types are supported:

- System Verilog — logic, bit, wire, reg
- Verilog — wire, reg
- VHDL — bit, std_logic, std_ulogic, Boolean
(NOTE: Any subtypes of these are not supported.)

Predefined VCTs Supported from the UPF Standard

Power Aware simulation provides several predefined VCTs described in Annex C of IEEE Std1801-2009, which you can use with the `connect_supply_net -vct` command.

The predefined VCT definitions provided with Power Aware simulation are listed below.

```
create_hdl2upf_vct VHDL_SL2UPF \
-hdl_type {vhdl} \
-table { {'U' OFF} \
      {'X' OFF} \
      {'0' OFF} \
      {'1' ON} \
      {'Z' OFF} \
      {'L' OFF} \
      {'H' ON} \
      {'W' OFF} \
      {'-' OFF}}
```

```
create_upf2hdl_vct UPF2VHDL_SL \  
-hdl_type {vhdl} \  
-table {{PARTIAL_ON 'X'} \  
{ON '1'} \  
{OFF '0'}}
```

```
create_hdl2upf_vct VHDL_SL2UPF_GNDZERO \  
-hdl_type {vhdl} \  
-table { {'U' OFF} \  
{'X' OFF} \  
{'0' ON} \  
{'1' OFF} \  
{'Z' OFF} \  
{'L' ON} \  
{'H' OFF} \  
{'W' OFF} \  
{'-' OFF}}
```

```
create_upf2hdl_vct UPF_GNDZERO2VHDL_SL \  
-hdl_type {vhdl} \  
-table {{PARTIAL_ON 'X'} \  
{OFF '1'} \  
{ON '0'}}
```

```
create_hdl2upf_vct SV_LOGIC2UPF \  
-hdl_type sv \  
-table {{X OFF} \  
{Z PARTIAL_ON } \  
{1 ON } \  
{0 OFF }}
```

```
create_upf2hdl_vct UPF2SV_LOGIC \  
-hdl_type sv \  
-table {{PARTIAL_ON X} \  
{ON 1} \  
{OFF 0}}
```

```
create_hdl2upf_vct SV_LOGIC2UPF_GNDZERO \  
-hdl_type sv \  
-table {{X OFF} \  
{0 ON} \  
{1 OFF} \  
{Z OFF}}
```

```
create_upf2hdl_vct UPF_GNDZERO2SV_LOGIC \  
-hdl_type sv \  
-table {{PARTIAL_ON X} \  
{OFF 1} \  
{ON 0}}
```

```
create_upf2hdl_vct VHDL_TIED_HI \  
-hdl_type {vhdl} \  
-table {{ON '1'} \  
{PARTIAL_ON 'X'} \  
{OFF 'X'}}  
  
create_upf2hdl_vct SV_TIED_HI \  
-hdl_type sv \  
-table {{ON 1} \  
{PARTIAL_ON X} \  
{OFF X}}  
  
create_upf2hdl_vct VHDL_TIED_LO \  
-hdl_type {vhdl} \  
-table {{ON '0'} \  
{PARTIAL_ON '0'} \  
{OFF 'X'}}  
  
create_upf2hdl_vct SV_TIED_LO \  
-hdl_type sv \  
-table {{ON 0} \  
{PARTIAL_ON X} \  
{OFF X}}  
  
create_hdl2upf_vct SV_BIT2UPF \  
-hdl_type {sv bit}\  
-table {{1 ON } \  
{0 OFF }}  
  
create_upf2hdl_vct UPF2SV_BIT \  
-hdl_type {sv bit} \  
-table {{PARTIAL_ON 0} \  
{ON 1} \  
{OFF 0}}  
  
create_hdl2upf_vct SV_BIT2UPF_GNDZERO \  
-hdl_type {sv bit}\  
-table {{0 ON} \  
{1 OFF}}  
  
create_upf2hdl_vct UPF_GNDZERO2SV_BIT \  
-hdl_type {sv bit}\  
-table {{PARTIAL_ON 1} \  
{OFF 1} \  
{ON 0}}  
  
create_hdl2upf_vct VHDL_BIT2UPF \  
-hdl_type {vhdl bit} \  
-table {{'1' ON } \  
{'0' OFF } }
```

```
{'0' OFF }}

create_upf2hdl_vct UPF2VHDL_BIT \
-hdl_type {vhdl bit} \
-table {{PARTIAL_ON '0'}}
{ON '1'} \
{OFF '0'}}
create_hdl2upf_vct VHDL_BIT2UPF_GNDZERO \
-hdl_type {vhdl bit}\
-table {{'0' ON} \
{'1' OFF}}

create_upf2hdl_vct UPF_GNDZERO2VHDL_BIT \
-hdl_type {vhdl bit}\
-table {{PARTIAL_ON '1'} \
{OFF '1'} \
{ON '0'}}

create_hdl2upf_vct VHDL_BOOL2UPF \
-hdl_type {vhdl boolean} \
-table {{TRUE ON} \
{FALSE OFF }}

create_upf2hdl_vct UPF2VHDL_BOOL \
-hdl_type {vhdl boolean} \
-table {{PARTIAL_ON FALSE}
{ON TRUE} \
{OFF FALSE}}

create_hdl2upf_vct VHDL_BOOL2UPF_GNDZERO \
-hdl_type {vhdl boolean}\
-table {{FALSE ON} \
{TRUE OFF}}

create_upf2hdl_vct UPF_GNDZERO2VHDL_BOOL \
-hdl_type {vhdl bit}\
-table {{PARTIAL_ON TRUE} \
{OFF TRUE} \
{ON FALSE}}
```

Connections Using Value Conversion Tables (VCTs)

You can model more complex connections between supply nets and HDL ports by using either of the following UPF commands to define value conversions:

```
create_upf2hdl_vct
create_hdl2upf_vct
```

Limitations

Only following types are supported in HDL data types:

- SystemVerilog (logic, bit, wire, reg)
- Verilog (wire, reg)
- VHDL (bit, std_logic, std_ulogic, Boolean).

Any subtypes of these are not supported.

Examples

```
create_hdl2upf_vct VHDL_SL2UPF \  
  -hdl_type {vhdl} \  
  -table {{'U' OFF} \  
         {'X' OFF} \  
         {'0' OFF} \  
         {'1' ON} \  
         {'Z' OFF} \  
         {'L' OFF} \  
         {'H' ON} \  
         {'W' OFF} \  
         {'-' OFF}}  
  
create_upf2hdl_vct UPF_GNDZERO2SV_LOGIC \  
  -hdl_type sv \  
  -table {{PARTIAL_ON X} \  
         {OFF 1} \  
         {ON 0}}
```

In addition, you can specify an existing value conversion table (VCT) using the following UPF command and argument:

```
connect_supply_net -vct
```

Note



Power Aware simulation provides several predefined VCT definitions (see “[Value Conversion Tables](#)”) that you with `connect_supply_net -vct`. However, if the VCT specification for a connection is not applicable or valid, then Power Aware simulation applies the default 1-bit connection semantics.

Examples

RTL Specification (Verilog):

```
module memory (input VSS, VDD);  
  ..  
endmodule
```

RTL Specification (VHDL):

```
entity memory is
port (VSS : in std_logic; VDD: in std_logic;..)
..
end entity
```

UPF specification:

```
connect_supply_net VDD_switchable -ports mem_sv/VDD
connect_supply_net VSS_switchable -ports mem_sv/VSS -vct
UPF_GNDZERO2SV_LOGIC

connect_supply_net VDD_switchable -ports mem_vhd/VDD
connect_supply_net VSS_switchable -ports mem_vhd/gnd -vct
UPF_GNDZERO2VHDL_SL
```

RTL Specification (Verilog):

```
module memory (input VSS, ..);
...
endmodule
```

UPF specification:

```
connect_supply_net VSS_switchable -ports mem_sv/VSS -vct
SV_LOGIC2UPF_GNDZERO
```

Data flow for the connection is UPF to HDL, but VCT is specified as HDL to UPF. As a result, the specified VCT is not applicable.

Simulation Semantics for UPF Supply Connections

Power Aware simulation semantics are automatically disabled for design elements that are explicitly or automatically connected to a particular supply net or supply set. Simulation semantics are disabled for all the descendants of the instance of model.

Examples

RTL Specification (Verilog):

```
module memory(input vdd, ..);
...
endmodule
```

RTL Specification (VHDL):

```
entity memory is
port (vdd : in std_logic; ..)
..
end entity
```

UPF specification:

```
create_power_domain mem_pd -elements mem
connect_supply_net vdd_switchable -ports mem/vdd
```

Usage Note

When automatic connection semantics are applied, simulation semantics of all the instances whose ports are automatically connected will be disabled. This may display a notification message similar to the following:

```
** Note: top.upf(12): (vopt-9693) Power Aware simulation semantics
disabled for /testbench/rtl_top/mem
```

It is recommended that you use `vopt -suppress 9693` to suppress this message.

Supply Nets

This section describes how to use UPF commands with Power Aware simulation to define supply net behavior for Power Aware.

Resolving Drivers on a Supply Net

Supply nets are often connected to the output of a single switch. However, if you have a design where the output of multiple switches are connected to the same supply net, you need a resolution mechanism to determine the state and voltage of the net, depending on the values supplied by the each of the individual switches.

[Version 1.0 of the UPF Standard](#) (UPF v1.0) defines the semantics of using the `create_supply_net` command to determine the resolution of different drivers on a supply net. In UPF v1.0, the resolution is mainly dependent on the state and voltage value of the supply network. In particular, the `-resolve` argument provides the following values, which you to select a method of multiple drivers on a supply net (refer to Section 5.1.5):

- unresolved
- one-hot
- parallel

Note that resolution can only be defined on the supply net and not on supply ports. As a result you can have multiple ports driving a supply net, but not multiple supply nets driving a supply port.

Example

In the following example, ports p1, p2, and p3 are driving the resolved supply net, resol_net, which has 'one_hot' resolution specified.

A voltage value and supply state would be driven on resol_net, depending on the resolution (one_hot) specified and the state of the drivers (p1, p2, p3).

In a case of a error in resolution, the supply net is turned off and voltage value is driven to 0. An error message is displayed that lists the voltages and states of all the drivers.

The following is an example of the error message for one_hot and parallel resolution respectively.

```
set_scope tb
create_power_domain PD_TOP -elements { top1 } -include_scope
create_supply_port p1 -domain PD_TOP -direction in
create_supply_port p2 -domain PD_TOP -direction in
create_supply_port p3 -domain PD_TOP -direction in
create_supply_net resol_net -domain PD_TOP -resolve one_hot

##### Supply Net Resolution #####
connect_supply_net resol_net -ports p1

connect_supply_net resol_net -ports p2

connect_supply_net resol_net -ports p3

** Error: (vsim-8928) MSPA_RSLV_ONE_HOT: Time: 20 ns, More than one driver
is ON for one_hot supply net: /tb/resol_net

# Drivers are:-

# /tb/p1: {0 uV, OFF}
# /tb/p2: {5 uV, ON}
# /tb/p3: {10 uV,ON}

# ** Error: (vsim-8927) MSPA_RSLV_PARALLEL: Time: 15 ns, Different
voltages driven by ON drivers of parallel supply net /tb/resol_net

# Drivers are:-

# /tb/p1: {0 uV, OFF}
# /tb/p2: {5 uV, ON}
# /tb/p3: {10 uV, ON}
```

Defining Isolation

Isolation is used to separate signals that originate in a design element with power off from a part of the design that has power on and that can still read the signals from the powered down element. A particular domain may be powered off while another domain is operating in normal mode.

There are two methods for defining an isolation cell.

Method 1: Isolation is already explicitly present

In this case, the design (in either a RTL or a GL netlist) contains an explicit cell instance that functions as an isolation cell. The UPF file includes a `set_isolation -instance` command that identifies this explicit instance as an isolation cell. In this case:

- No isolation needs to be added
- Power Aware will not insert an additional isolation cell
- Power Aware will not modify the explicit isolation cell that is present (it is assumed that the user-provided isolation cell does what it is supposed to do)

Method 2: Isolation needs to be added

In this case, you use the `set_isolation` command to provide an isolation strategy (without the `-instance` option). Power Aware simulation then implicitly inserts an isolation cell on any port that satisfies all of the following:

- Matches the criteria defined in the isolation strategy
- Requires isolation because the power state table indicates that the two power domains on either side of the port can be ON/OFF or OFF/ON respectively
- Does not already have an explicit isolation cell present and identified (per Method 1)

By default, if Power Aware simulation does insert an isolation cell, Power Aware simulation will use a built-in behavioral model for isolation. However, you can cause the simulation to insert a different model (when required) by using the `map_isolation_cell` UPF command to identify the isolation model to be used.

Note



To prevent a redundant isolation cell from being inserted on a port, use the `set_isolation -instance` command to identify the instance of the existing isolation cell.

Specifying Isolation Cells

Power Aware simulation reads the `set_isolation` command in the UPF file and identifies ports as candidates for isolation cell insertion. These ports are also dumped into the UPF report file (`report.upf.txt`), as in [Example C-29](#).

Example C-29. UPF Report File for Isolation Ports (`report.upf.txt`)

```
Isolation Strategy: iso_PD_mid1, File: test.upf(44).
Isolation Supplies:
  power   : /tb/TOP/tb_pow
  ground  : /tb/TOP/mid1_GND_NET
Isolation Control (/tb/TOP/ctrl), Isolation Sense (HIGH), Clamp Value
(1), Location (automatic)

Isolated Signals:
  1. Signal : /tb/TOP/mid1/out2_bot
  2. Signal : /tb/TOP/mid1/out1_bot
```

Isolation Cell Instances

If you have used the `set_isolation -instance` command in a UPF file to instantiate RTL isolation cells, Power Aware simulation will infer those cells and perform isolation checks on them.



Power Aware simulation automatically infers the right UPF strategy for cells that are not specified in the UPF file. For more information, refer to “[Detection of Power Management Cells](#).”

A cell instance is identified as an isolation cell in any of the following cases:

- Instance is specified with `-instance` argument of `set_isolation` command.
- For a GLS design, any of the following is present:

- Synopsys pragma `synopsys isolation_upf`.

Example:

```
ISOLOD1BWP trafficWriteEnable_UPF_ISO ( .I(trafficWriteEnable),
.ISO(n16),.Z(n57) ); //synopsys isolation_upf PD_C2_ISS1+PD_C2
```

- Liberty attribute `is_isolation_cell`. Example:

```
cell (ISO_LO) {
    is_isolation_cell : true;
    ...
}
```

- HDL attribute `is_isolation_cell`. Example:

```
(* is_isolation_cell = 1 *)
  module ISOCELL ( I, ISO, Z ) ;
    ....
  endmodule
```

- o Cell in accordance with `map_isolation_cell` UPF command. Example:

```
UPF Command : map_isolation_cell mem_ctrl_iso_0 -domain PD_mem_ctrl
-lib_cells {ISO_LO}
  Instantiation in hdl : ISO_LO addr_0 ( .I(n158), .ISO(n174),
.Z(address[0]) );
```

- o Instance name in accordance with `name_format` UPF command. Example:

```
UPF Command : name_format -isolation_prefix "MY_ISO_"
-isolation_suffix "UPF_ISO"
  Instantiation in hdl : SEN_OR2_4 MY_ISO_t_state_21__UPF_ISO
(.A1 ( n151 ) , .A2 ( n354 ) , .X ( t_state[21] ) ) ;
```

Limitations

If isolation is required and is not present (and identified with `set_isolation -instance`), then Power Aware simulation effectively inserts an isolation cell.

The current Power Aware architecture implements isolation by *actually* inserting a cell—an instance of a behavioral model for isolation—into the design where isolation is required. However, ports on an isolation cell inserted this way have a limitation of not supporting enum types.

For ports of type enum, Power Aware reverts to the implementation approach used in previous releases. In that approach, an isolation cell is not *actually* inserted into the design. Instead, internal mechanisms (such as the `force` command) make an existing design port (`highconn` or `lowconn`) behave as if isolation were present.

Note that the isolation effect of these two approaches is identical—you will see exactly the same results in reports and in the GUI. The only difference is that the new architecture shows explicit instances of isolation cells that have been added to the design.

Defining Retention

To define retention registers in a power domain and set the corresponding save and restore signals for the retention registers, you use the `set_retention` command in your UPF file. In particular, you can use the following arguments (which are supported only in UPF v2.0) to define your retention strategy:

- `-retention_supply_set`—Defines the supply set used to power the logic inferred by the `<retention_name>` strategy.

- `-no_retention` — Storage elements specified by this argument are prevented from having retention capability.
- `-use_retention_as_primary` — Specifies that the storage element and its output are powered by the retention supply.

`-retention_supply_set`

Description

This argument defines the supply set used to power the register holding the retained value.

Note the following cases:

- If you have specified both retention power and retention ground nets, using this command creates an implicit retention supply set and is used with the specified strategy. The retention power net serves the power function in the retention supply set and the retention ground net serves the ground function in the retention supply set.
- If you have specified the retention power net but not the retention ground net, then the domain's primary supply sets ground function is used as the retention ground.
- If you have specified the retention ground net but not the retention power net, then the domain's primary supply sets power function is used as the retention power.

Note that the power of the retention cell is preserved (an extra port RETPWR (Retention Power) is added to the retention models).

Whenever the power of a retention cell (RETPWR) goes down, the value stored in retention cell gets corrupted, and the 'X' value is restored when restore signal is triggered.

Example

Consider the following logic added in a retention model:

```
// store x in RETPWRDOWN
always @(negedge RETPWR)
begin
  -> pa_store_x ;
end
```

An extra port RETPWR has been added in the Power Aware retention models.

Whenever you use your own retention models using the `map_retention_cell` UPF command:

- For a user-defined model with RETPWR port and corresponding logic—Behavior will be in accordance with the RETPWR logic.
- For a user-defined model without a RETPWR port — A warning is issued during the vsim simulation. For example:

```
# ** Warning: (vsim-PA-8944) Retention Model : 'MyRetModel' does not have  
Port 'RETPWR'. Ignoring it.  
# Region: /mspa_top/blk0/inst0_MyRetModel
```

-no_retention

Description

This argument disables retention on specified storage elements.

Example

```
upf_version 2.0  
set_scope tbl  
create_power_domain PD_TOP -elements { top1/bot1/tdsig}  
create_power_domain PD_TOP2 -elements { top1/q1 }  
create_power_domain PD_TOP3 -elements { top1/}  
...  
set_domain_supply_net PD_TOP -primary_power_net PD_TOP_primary_power -  
primary_ground_net GND_net  
set_domain_supply_net PD_TOP2 -primary_power_net PD_TOP_primary_power2 -  
primary_ground_net GND_net  
set_domain_supply_net PD_TOP3 -primary_power_net PD_TOP_primary_power3 -  
primary_ground_net GND_net  
create_power_switch PD_TOP_sw \  
-domain PD_TOP \  
-output_supply_port { out_sw_PD_TOP PD_TOP_primary_power } \  
-input_supply_port { in_sw_PD_TOP VDD_net } \  
-control_port { ctrl_sw_PD_TOP pg } \  
-on_state { normal_working in_sw_PD_TOP {ctrl_sw_PD_TOP } } \  
-off_state { off_state {!ctrl_sw_PD_TOP} }  
...  
set_retention PD_TOP_retention3 -domain PD_TOP -retention_power_net  
VDD_net -elements { top1/bot1/tdsig[0][1] } -no_retention  
set_retention_control PD_TOP_retention3 -domain PD_TOP -save_signal { ret3  
posedge } -restore_signal { ret1 negedge }  
  
set_retention PD_TOP_retention2 -domain PD_TOP -retention_power_net  
VDD_net -elements { top1/bot1/tdsig }  
set_retention_control PD_TOP_retention2 -domain PD_TOP -save_signal { ret2  
posedge } -restore_signal { ret1 negedge }  
  
set_retention PD_TOP_retention1 -domain PD_TOP -retention_power_net  
VDD_net -elements { top1/bot1/tdsig[0][0] }  
set_retention_control PD_TOP_retention1 -domain PD_TOP -save_signal { ret1  
posedge } -restore_signal { ret1 negedge }  
...
```

As a result of this UPF command, no retention strategy is applied to:

```
top1/bot1/tdsig[0][1]
```

Reports

```
report.upf.txt
-----
Power Domain: PD_TOP, File: ./src/bitwise_1/test.upf(7).
  Creation Scope: /tb1
  Primary Supplies:
    power   : /tb1/PD_TOP_primary_power
    ground  : /tb1/GND_net
  Power Switch: PD_TOP_sw, File: ./src/bitwise_1/test.upf(34).
  Output Supply port:
    out_sw_PD_TOP(/tb1/PD_TOP_primary_power)
  Input Supply ports:
    1. in_sw_PD_TOP(/tb1/VDD_net)
  Control Ports:
    1. ctrl_sw_PD_TOP(/tb1/pg)
  Switch States:
    1. normal_working(ON) : (ctrl_sw_PD_TOP )
    2. off_state(OFF)    : (!ctrl_sw_PD_TOP)
  Retention Strategy: PD_TOP_retention3, File:
./src/bitwise_1/test.upf(58).
  Retention Supplies:
    power   : /tb1/VDD_net
    ground  : /tb1/GND_net
  No Retention
  Retention SAVE (/tb1/ret3), Retention Sense (posedge)
  Retention RESTORE (/tb1/ret1), Retention Sense (negedge)
  Retention Strategy: PD_TOP_retention2, File:
./src/bitwise_1/test.upf(61).
  Retention Supplies:
    power   : /tb1/VDD_net
    ground  : /tb1/GND_net
  Retention SAVE (/tb1/ret2), Retention Sense (posedge)
  Retention RESTORE (/tb1/ret1), Retention Sense (negedge)
  Retention Strategy: PD_TOP_retention1, File:
./src/bitwise_1/test.upf(64).
  Retention Supplies:
    power   : /tb1/VDD_net
    ground  : /tb1/GND_net
  Retention SAVE (/tb1/ret1), Retention Sense (posedge)
  Retention RESTORE (/tb1/ret1), Retention Sense (negedge)

-----
Power Domain: PD_BOT, File: ./src/bitwise_1/test.upf(70).
...

report.mspa.txt

-----
----- ModelSim Power Aware Report File -----
-----
Total ( tb1 )
  upf_retention_ret # 1
  upf_retention_sr # 12
  NPM_FF # 12
  NPM_LA # 2
  OUTPUT # 38
...
```

```

-----
PD_TOP sub_total ( /tb1/top1/bot1 )
upf_retention_ret # 1
  /tb1/top1/bot1/tdsig[0] 1
upf_retention_sr # 12
  /tb1/top1/bot1/tdsig[2] 1
  /tb1/top1/bot1/tdsig[3] 1
  /tb1/top1/bot1/tdsig[4] 1
  /tb1/top1/bot1/tdsig[5] 1
  /tb1/top1/bot1/tdsig[6] 1
  /tb1/top1/bot1/tdsig[7] 1
  /tb1/top1/bot1/tdsig[8] 1
  /tb1/top1/bot1/tdsig[9] 1
  /tb1/top1/bot1/tdsig[10] 1
  /tb1/top1/bot1/tdsig[11] 1
  /tb1/top1/bot1/tdsig[12] 1
  /tb1/top1/bot1/tdsig[13] 1
NPM_FF # 1
  /tb1/top1/bot1/tdsig[1] 1
-----
PD_TOP2 sub_total ( /tb1/top1 )
OUTPUT # 1
  /tb1/top1/q1 1
-----
...
PD_TOP3 sub_total ( /tb1/top1 )
NPM_FF # 7
  /tb1/top1/d1[6:4] 3
  /tb1/top1/d1[3:0] 4
OUTPUT # 11
  /tb1/top1/out_buf 1
  /tb1/top1/out_reg 1
  /tb1/top1/out_lat 1
  /tb1/top1/out_uniso 1
  /tb1/top1/d2 7
-----
-- NPM_FF => Denotes all Non Power Management Flip Flops of a Power
Domain.
-- NPM_LA => Denotes all Non Power Management Latches of a Power Domain.
-- OUTPUT => Denotes all outputs and power signals, which are not
sequential elements, of a Power Domain.

```

-use_retention_as_primary

Description

This argument powers the storage element and the output drivers of the register using the retention supply.

Example

```

upf_version 2.0
set_scope tb
create_power_domain pd_aon -include_scope
...

```

```
connect_supply_net    vdd_net    -ports { vdd_port }
connect_supply_net    gnd_net    -ports { gnd_port }
create_supply_set pd_aon_ss \
    -function { power vdd_net } \
    -function { ground gnd_net }
...
#####
# Retention Strategy for pd
#####
set_retention          pd_retention1 -domain pd -save_signal { ret posedge
} -restore_signal { ret negedge } -elements {top_vh} -
use_retention_as_primary
map_retention_cell     pd_retention1 -domain pd -lib_model_name
upf_retention_ret -lib_cell_type FF_CKHI
associate_supply_set pd_aon_ss -handle pd.pd_retention1.supply
...
```

Report

report.mspa.txt

```
-----
----- ModelSim Power Aware Report File -----
-----
Total ( tb )
upf_retention_ret # 2
NPM_FF # 1
NPM_LA # 3
OUTPUT # 3
-----
pd sub_total ( /tb/top_vh /tb/top_vl )
upf_retention_ret # 1
/tb/top_vl/q_regvl 1
NPM_LA # 2
/tb/top_vh/q_latvh 1
/tb/top_vl/q_latvl 1
OUTPUT # 2
/tb/top_vh/q_combvh 1
/tb/top_vl/q_combvl 1
-----
pd.pd_retention1.use_retention_as_primary sub_total ( /tb/top_vh )
upf_retention_ret # 1
/tb/top_vh/q_regvh 1
-----
pd_aon sub_total ( /tb/top_aon )
NPM_FF # 1
/tb/top_aon/q_regvl 1
NPM_LA # 1
/tb/top_aon/q_latvl 1
OUTPUT # 1
/tb/top_aon/q_combvl 1
-----
-- NPM_FF => Denotes all Non Power Management Flip Flops of a Power
Domain.
-- NPM_LA => Denotes all Non Power Management Latches of a Power Domain.
```

-- OUTPUT => Denotes all outputs and power signals, which are not sequential elements, of a Power Domain.

Appendix D

Power Configuration File Reference

In addition to the UPF format (see [UPF Commands and Reference](#)), Mentor also supports a format for specifying the power intent of a design that is referred to as Power Configuration File (PCF).

Power Specification File

To perform Power Aware verification, you need to provide a power specification file that identifies the low-power specification of the design. A power specification file is analogous to a standard delay file (SDF) used for annotating timing and timing check information into a simulation.

Using a power specification file is the key to the verification flow using Power Aware. This file provides the following information required to overlay verification with the power control network and Power Aware functionality:

- Power regions, voltage domains, and power islands
- Retention sequential models, their type, and the regions they are in (including nodebug, encrypted, and protected regions)
- State and output corruption behavior in power-down situations
- Power control signals and the portions of the design they control

The power specification file is designed to capture all Power Aware characteristics of the design at the RTL (or higher) in a compact form that can be easily used by the simulator.

Formats

This power specification file can be written using:

- Power Configuration File (PCF) — A preliminary file format specific to ModelSim that was developed to meet the specific needs of various customers and semiconductor companies. Refer to [Power Configuration File Reference](#) for more information on PCF.

Using a PCF as Part of Power Aware Verification

You can use a power specification file written as a Power Configuration File (PCF) as part of a Power Aware analysis at the RTL level as follows:

1. Code the functional design using normal RTL coding guidelines. Specifically, the power control network is not wired through the RTL functional network.
2. Model the power management blocks, including the definition of the power control signals. These signals either indicate the on/off status of power to specific power islands or flag the storing and restoring of state information.
3. Use the PCF to specify how the power control network overlays connect to the functional network.
4. Use a Power Aware library of Verilog models provided by the silicon (or library) vendor. The library models trigger relevant events for the simulator so that the simulator can modify the run-time behavior of the RTL design to corrupt state and outputs and store or restore state values based on power control network activity. For more information refer to [Model Construction for Power Aware Simulation](#)
5. Use the PCF to map the Power Aware models to sequential elements in the design.
6. Use the Power Aware model library, it is presumed to be precompiled as it does not change.
7. Compile the RTL design normally.
8. Simulate the design (see [Using the Standard Flow](#)). When the design is simulated, the simulator will take the PCF as input as well as the compiled design.
9. Proceed with the simulation normally, as the test bench triggers power down signals for various power islands while save and restore occur as required. The purpose of the verification is to ensure the design functions correctly within the dynamic context of power islands being turned off and on.

PCF Syntax and Contents

This section describes the syntax and semantics of the PCF, using Backus-Naur Format (BNF) grammar to specify the syntax.

Basic PCF Statement Types

The PCF consists of header and power or context statements. The header statement must be the first statement in the PCF. Power and context statements can be intermixed. Context statements apply to all statements that follow the context statement unless and until overridden by a subsequent context statement.

Statement Termination

You should terminate all statements with a semicolon (;). However, for backward compatibility with earlier PCF formats, a statement may be terminated by a new line (CRLF).

When using a new line to terminate a statement, use the backslash (\) character to indicate line continuation instead of termination immediately prior to the new line.

Only one form of statement termination can be used in a simulation session.

```
stmt_end ::=  
    ;  
    | CRLF
```

Header Statement

The header statement is mandatory for PCF version 2.0 or later. It is an error if the PCF version is not specified or if there is no header statement prior to the occurrence of the first power or context statement.

Future versions of the PCF may define additional header information. That is why the header statement is defined as a comma-separated list of keyword-value pairs that are specified only once in a PCF file.

```
header_statement ::=  
    HEADER keyword_value_pair {, keyword_value_pair }  
    stmt_end  
  
keyword_value_pair ::=  
    ( keyword = value )  
  
keyword ::=  
    VERSION  
    | SEPARATOR  
    | STMT_TERM  
    | ... (user or vendor defined)
```

Note



Currently, the only information defined in the header statement is the PCF version number and separator character. The verification tool provider will specify which versions numbers they support. Future versions of the PCF specification may identify additional keywords and the values associated with them.

- **VERSION** — (Required) This keyword-value pair identifies the PCF specification version number and must be the first keyword specified in the header statement as this information can determine whether or not subsequent keywords are recognizable and supported. The version number will be of the form:

<positive>.<natural>

Vendors may not extend or add to the version number as it applies to the version of the PCF specification and not to a vendor's tool version.

- **SEPARATOR** — (Optional) This keyword-value pair identifies the separator character used to separate one scope level from the next. If not specified, the separator character is defined by the tool processing the PCF or, if the tool defines no default, it is the slash character (/). Although defined by a string value, the separator character string value must be exactly one character in length.
- **STMT_TERM** — (Optional) This keyword-value pair identifies the statement termination. In PCF 1.0, the default statement termination is CRLF. For PCF 2.0 or later, the default statement termination is semicolon. The only permissible values are “CRLF” or “;”.
- **VENDOR_<string>** — (Optional) This keyword-value pair specifies a vendor’s tool-specific entry.

Vendors may define their own tool-specific header information. Vendor-specific information must be tagged by the prefix “VENDOR_” in the keyword. Vendor-specific header information cannot be used to change the PCF syntactic or semantic information as defined in this specification. It can be used for general informational purposes, for example to identify the tool and tool version that generated or is targeted for consuming the PCF. Vendor-specific keyword values must be double quoted strings.

Example

This example shows the use of a secondary prefix (“MGC”) that identifies the specific vendor, in this case Mentor Graphics Corporation. Such keyword naming conventions are encouraged as they facilitate the ability of tools to easily identify header information that has meaning to them.

```
HEADER ( VERSION = "2.0" ),  
        (SEPARATOR = "/" ),  
        (VENDOR_MGC_SOURCE = "PX" ),  
        (VENDOR_MGC_SOURCE_VERSION = "2006.05" ) ;
```

Context Statements

By setting the current design scope as the implicit prefix to all instance paths, you can significantly reduce the amount of text required in the PCF. Context statements provide the ability to define specific context items that apply to all power (and context) statements that follow until another context statement makes a subsequent change.

```
context_statement ::=  
  scope_statement  
  | variable_statement  
  | include_statement  
  | corruption_map_statement
```

```
| corruption_extent
```

Scope Statement

The scope statement sets the current scope context within the hierarchy of the design being verified (including the test bench). All relative paths used in subsequent context or power statements will be equivalent to an absolute path starting with the current scope and proceeding through the relative path specified in the statement. The most recent scope statement shall apply to each relative path specified in the PCF.

```
scope_statement ::=
    SCOPE instance_path stmt_end

instance_path ::=
    relative_path
    | absolute_path

relative_path ::=
    hdl_identifier { separator_character hdl_identifier }

absolute_path ::=
    separator_character relative_path
```

Another application may provide a mechanism, external to the PCF, to specify information equivalent to a scope statement being specified immediately after the header statement. This usage model supports the use of the same PCF with different test benches or in block level as well as chip level verification.

Examples

```
SCOPE /root/dut ; -- Set initial scope to DUT

SCOPE memctrl/arbitrer ; -- Set scope to memctrl.arbitrer
                        -- instance down design hierarchy
                        -- from current scope.
```

Variable Statement

Variable statements are used to store frequently used values. The values are treated as untyped and are applied to the context specified and then interpreted. They are very similar to UNIX environment variables or macros. An example usage would be to define a path to a common level of hierarchy from which many instance, signal, etc. paths will then be defined.

```
variable_statement ::=
    $identifier = value stmt_end
```

- `$<identifier>` — The lexical form of the identifier adheres to the rules for identifiers in Verilog (IEEE Std 1364-2005). The identifier must be prefixed by the dollar sign (\$) symbol in both the specification of its value and the referencing of that value. No white space is permitted between the “\$” symbol and the identifier.

- value — a double quoted (“...” string.

Example

```
$memctrl_scope = "/top/dut/memory_unit/memctrl" ;  
SCOPE $memctrl_scope/arbiter ;  
  
$inst_path = "top/pd1/inst1/inst11";  
POWER PDINST11 $inst_path/cell1,  
  ($inst_path/gl obal_vdd1 &  
  $inst_path/vd1/local_vdd_11);
```

Include Statement

The ability to textually include another PCF file is provided through the include statement. The include statement allows PCF files to be created on a hierarchical or power island or voltage domain basis. It also allows the specification of common, reusable information in a separate file for easier maintenance. For example, the data type corruption mapping information can be specified once and then included in any design PCF that uses that specific mapping.

```
include_statement ::=  
  include filepath_name stmt_end
```

- `filepath_name` — a simple file name, a relative file path name or an absolute file path name. Simple filenames will be searched in the current working directory of the tool processing the PCF. Relative file path names will be searched using the current working directory of the tool as the starting point for the search. Absolute file path names are searched without regard to the current working directory.

The *filepath_name* must conform to the UNIX file path name convention.

It is an error if the specified include file does not exist.

Example

```
include corruption_map.pcf ; -- corruption_map.pcf  
                             -- must exist in current  
                             -- working directory  
  
include memctrl/memctrl.pcf ; -- Search in memctrl  
                             -- subdirectory of  
                             -- current working dir  
  
include /u/home/tsmith/projs/memctrl/memctrl.pcf ;  
                             -- Absolute path to a PCF file to include
```

Corruption Extent Statement

By default, corruption semantics are applied only to what has been specified. Frequently, corruption in power down situations should be applied more extensively. The corruption extent

statement allows the specification of the extent that corruption is applied in power-down situations.

```
corruption_extent ::=
    CEXTENT = extent_keyword stmt_end

extent_keyword ::=
    OUTS_SEQ_AND_WIRES
    | OUTS_AND_SEQ
    | OUTPUTS
```

- **OUTS_SEQ_AND_WIRES** — corrupts registers and signals that are driven by logic which is powered down.
- **OUTS_AND_SEQ** — corrupts output ports and sequential elements within the power domain.
- **OUTPUTS** — corrupts only the output ports of a power domain.

Power Statements

Power statements define voltage domains, power islands, mapping of inferred registers and latches to Power Aware models and corruption behavior.

```
power_statement ::=
    power_control statement
    | power_model_mapping
```

Power Control Statement

The power control statement defines the power islands by mapping a power control signal to the design power elements to which it applies.

```
power_control_statement ::=
    POWER tag [-osw | -os | -o] region_definitions , ( boolean_expr )
    [ retention_specification ]
    [ POWER_VDD voltage_specification ]
    stmt_end
```

- tag — a name without any white space (spaces, tabs, etc.) that provides a descriptive handle or identifier for the power island or power control mapping being defined.
- [-osw | -os | -o] — command options that allow you to override the global corruption extent of the vopt -pa_ce command for specific power domains.
 - osw — sets the corruption extent to outputs and sequential and non-sequential wires.
 - os — sets the corruption extent to outputs and sequential elements.
 - o — sets the corruption extent to outputs only.

For example, you can use `-o` to define a specific power domain with non-synthesizable RTL to prevent register/latch detection, even though you the command line specifies `vopt -pa_ce=os`.

- `region_definitions` — a list of signal, instance, and process definitions that allow the declaration of a power island by specifying various combinations of elements which are controlled by the power island on/off state. Refer to the section [Region Definitions](#) for detailed information.
- `boolean_expr` — a Verilog boolean expression. When the expression evaluates TRUE, power to the specified elements in `rtl_region_definition` is powered ON. When it evaluates FALSE, power to the specified elements in `rtl_region_definition` is OFF.
- `retention_specification` — (optional) The power control statement may optionally specify the signal(s) used to control retention behavior throughout the power island defined by the power control statement. The retention specification is described in the section [Retention Statement](#).
- `POWER_VDD voltage_specification` — (optional) To facilitate a concise specification in the situation where the voltage domain and the power island are the same, the power control statement may optionally specify the operating voltages for the power island/voltage domain. The voltage specification is described in the section [Voltage Domains](#).

Region Definitions

The region definitions are a list of signal, instance and process definitions. This will allow declaring a power island by specifying various combinations of elements which are controlled by the power island on/off state.

```
region_definitions ::=
  region_definition { , region_definition }

region_definition ::=
  signal_definition
  | instance_definition
  | process_definition

signal_definition ::=
  -s hdl_path

instance_definition ::=
  [ -i ] [ -nr ] hdl_path

process_definition ::=
  -p hdl_path
```

In each region definition, the HDL path is relative to the context that has previously been set through a `SCOPE` statement. The path must terminate at an object or scope appropriate for the type of definition (a signal or port, block instance or process instance).

- Signal Definition
 - **-s** — specifies that the given signal (*hdl_path*) is part of the power island and must be corrupted on power down. The signal is the lowest level of granularity for power region definition.

- Instance Definition

Instance regions are recursive by default. That is, the corruption includes the specified instance and extends to any additional instances contained within the design hierarchy sub-tree rooted at the specified instance through the lowest leaves of that sub-tree.

You can target individual instances within generate using two underscore characters (__) instead of square brackets, [].

- **-i** — For block instances, the argument **-i** is optional, since it is the only region definition that does not require an argument to identify the kind of region. It is recommended that you specify the **-i** argument for clarity.
- **-nr** — If the corruption should not extend beyond the immediately specified instance, then you can specify the **-nr** argument, since **-nr** specifies non-recursive corruption.

Note

The **-nr** argument applies to all instance names that match a regular expression.

The **-nr** argument is meaningful only during 'all' corruption: `CEXTENT = ALL`. When you specify this argument, the simulator applies the corruption only to the inferred registers and latches contained in the root of the sub-tree defined by the instance specification and not to the instances declared below that.

Corruption of instance outputs will be applied to the outputs of all the instances listed in the `POWER` statements. In the case of an instance declared without **-nr**, output corruption will be applied to the outputs of the top of the instance and not to the instances below that. That is, recursive application of corruption applies only to inferred registers and latches—never to outputs.

- Process Definition

To identify specific processes within a block instance and not all processes within a block instance, the PCF supports the specification of processes individually.

- **-p** — indicates that the region definition applies to only the specified process. The HDL path identifies the relative hierarchical path to the process's label.

The sequential elements in the process and the outputs (drivers) of the process will be corrupted. If two processes in one single instance are controlled by different power control signals, the sequential elements of the different processes need to be corrupted separately.

Example

```
POWER PD0
-s top/pd11/s1,
-nr top/pd11/w11,
-p top/pd11/c111/p1,
top/pd11/c112,
(top/global_vdd_1 & top/vd1/local_vdd_11) ;
```

A power island named PD0 is defined by the signals *top/global_vdd_1* bit-wise anded with *top/vd1/local_vdd_11*. In other words, the power is ON for PD0 only when both *vdd_1* and *vdd_11* are on (1). When the power for PD0 is off, the following are corrupted:

- The signal *top/pd11/s1*
- The block instance *top/pd11/w11* (inferred registers, latches and outputs)
- The inferred registers or latches for the process *top/pd11/c112*

Example

```
POWER VD1 top/vd1 , (top/global/vdd1)
```

The instance *top/vd1* is controlled by power control signal *top/global_vdd1*.

If corruption extent is set to ALL, then the outputs of *top/vd1* are corrupted when *top/global/vdd1* is 0 as are inferred registers and latches in *top/vd1* and all sub-blocks below *top/vd1*.

Example

```
POWER PD1
top/pd11 ,
(top/global_vdd1 & top/vd1/local_vdd_12);

POWER signal_always_on
-s top/pd11/out1, 1;
```

With this power specification file, the output *top/pd11/out1* will not be corrupted when the power for power domain PD1 goes off.

Example

Note



This example describes the combinatorial logic (powered by *top/VDD1*) which is an output *out1* of an instance *m2_00* which is powered from VDD2.

```
POWER PD1
-s top/m2_00/out1,
(top/VDD1);
```

```
POWER PD0
  top/m2_00,
  (top/VDD2);
```

This means if the VDD1 goes off, the 'out1' needs to go 'X', but VDD2 has no effect on the 'out1'.

Example

```
POWER PD0
  -s top/pd11/s1,
  -nr top/pd11/w11,
  -p top/pd11/c111/p1,
  top/pd11/c112,
  (top/global_vdd_1 & top/vd1/local_vdd_11)
  RETENTION top/vd1/save top/vd1/restore
  POWER_VDD {0.5} ;
```

Example

```
POWER PD1          -- Process p1 is in a different
  -p /top/m2_00/p1, -- power island
  (top/VDD1);

POWER PD2          -- Than process p2
  -p /top/m2_00/p2, -- But both exist in the same block
  (top/VDD2);

module m2(i1, i2, i3, i4, iso, a1, a2, clock, reset)
  output a1, a2;
  reg a11, a21;
  input i1, i2, i3, i4;
  input clk, reset;
  .....
  // Belong to PD1
  always @(i1 or i2 or i3)
  begin : p1
    .....
    a11 <= i1 & i2 & i3;
    ....
  end

  // Belong to PD2
  always @(i1 or i2 or i3)
  begin : p2
    ....
    a21 <= (i1 | i2 | i3);
    ....
  end
  .....
endmodule
```

Power Model Mapping Statement

The mapping statement defines how an inferred register or latch signal or variable in the design maps to a model from the Power Aware model library that defines its run-time behavior.

Need to allow mapping of a Power Aware model to all inferred sequential items (regs/latches) in a scope or to specific signals (whole signals, not bit, part or selected elements of a signal) in a scope. Wildcard matching will apply.

```
power_model_mapping ::=
  MAP model_module_name pacell_spec
    ret_ctrl_signals,
    region_definitions stmt_end

ret_ctrl_signals ::=
  ret_signal_name
  | ret_save_sig_name ret_restore_sig_name

pacell_spec ::=
  pacell_type [ interface_information ]

pacell_type ::=
  FF_CKHI
  | FF_CKLO
  | FF_CKFR
  | LA_ENHI
  | LA_ENLO
  | LA_ENFR
  | ANY_CKHI
  | ANY_CKLO
  | ANY_CKFR
  | RETMEM
  | NON_RETMEM

interface_information ::=
  ( named_parameter_assignment
    { , named_parameter_assignment } )
```

- `model_module_name` — specifies a Verilog module that defines the Power Aware (retention) behavior that will be mapped to the regions specified by `region_definition`. Normal Verilog library search will be used to locate the module (e.g., `-L my_lib`). It is an error if the specified module cannot be found or, if found, does not conform to the modeling guidelines for Power Aware models. For more information refer to [Model Construction for Power Aware Simulation](#).
- `ret_ctrl_signals` — specifies either a single signal name that will be connected to the RET port of the Power Aware model, or a pair of retention signal names that are connected to the SAVE and RESTORE ports of the Power Aware model. For more information refer to the chapter [Model Construction for Power Aware Simulation](#).
- `region_definitions` — determines which regions, defined in the section [Region Definitions](#), shall be supported in the power model mapping statement with the following semantics for each region:

- Signal — The specified signal shall infer a sequential element (register or latch) that will be mapped to the Power Aware model. A warning shall be issued if a sequential element has not been inferred for the specified signal. An error shall be issued if the `pacell_type` is specified and the Power Aware cell type inferred for the specified signal does not match.
- Instance — All sequential elements inferred within the instance will be mapped to the specified Power Aware model. By default, the mapping is recursive to all child instances of the specified instance. The `-nr` switch may be used to specify non-recursive mapping to the instance only (not to any child instances). A warning shall be issued if `pacell_type` is specified and no inferred sequential elements of that type are present in the region.
- Process — All sequential elements inferred within the process will be mapped to the specified Power Aware model. A warning shall be issued if there are no inferred sequential elements for the specified process. A warning shall be issued if `pacell_type` is specified and no inferred sequential elements of that type are present in the region.
- `pacell_type` — specifies the type of inferred sequential cell to which the mapping statement applies.

Specification of the Power Aware cell type avoids unintentional mapping of the incorrect Power Aware model to an inferred sequential element of a different type while permitting all types of cells to be inferred in the same region definition and PCF coverage of all inferred sequential elements with as few power model mapping statements as possible. When the cell type is specified, then only inferred sequential elements that match that type are mapped to the Power Aware model specified in the mapping statement. The cell types are defined as follows:

- `FF_CKHI` — Inferred registers (flip-flops) active on the positive edge of the clock signal.
- `FF_CKLO` — Inferred registers active on the negative edge of the clock signal.
- `FF_CKFR` — All inferred registers regardless of the edge sensitivity of the clock signal.
- `LA_ENHI` — Inferred latches active when the enable signal is high.
- `LA_ENLO` — Inferred latches active when the enable signal is low.
- `LA_ENFR` — All inferred latches regardless of the level sensitivity of the enable signal.
- `ANY_CKHI` — Generically matches any inferred sequential element (register or latch) that is active on the posedge of a clock or high level of an enable signal. That is, this cell type is equivalent to two mapping statements that are identical except for cell type where one statement cell type is `FF_CKHI` and the cell type of the other equivalent mapping statement is `LA_ENHI`.

- ANY_CKLO — Generically matches any inferred sequential element that is active on the engaged of a clock or low level of an enable signal. That is, this cell type is equivalent to two mapping statements that are identical except for cell type where one statement cell type is FF_CKLO and the cell type of the other equivalent mapping statement is LA_ENLO.
- ANY_CKFR — Generically matches any inferred sequential element without regard to edge or level sensitivity. A mapping statement specifying this cell type is equivalent to four mapping statements that are identical except for the cell type. The 1st equivalent statement has cell type FF_CKHI, the 2nd equivalent statement has cell type FF_CKLO, the 3rd equivalent statement has cell type LA_ENHI and the 4th equivalent statement has cell type LA_ENLO.
- RETMEM — Matches an inferred memory to a model that will determine under what conditions the memory contents would be corrupted. If the pacell type is RETMEM, then the region definition must specify a signal (-s <signal_for_memory_contents>). It is an error if any other region definition is specified.
- NON_RETMEM — Matches an inferred memory to corruption only semantics (memory contents are always corrupted in power down situations). In the case of NON_RETMEM pacell type, the model module name must indicate the built-in corruption model bi_nonret_memory. The builtin corruption model applies the corruption value as governed by the corruption map in effect for that scope.
- interface_information — (optional) The pacell_spec can optionally specify a named mapping of power control signals to the ports of the PA model. This allows the names of the signals in the design to differ from the names of the pins on the PA cell. The syntax used is straight from the Verilog 1364 LRM for named parameter assignment. The actual may be a simple name, a hierarchical name, a bit or part select or struct (record) element selection.

During verification, the Power Aware behavior is mapped from the specified Power Aware model to all inferred sequential elements matching the cell type in order to determine when to save and restore the state (or specified corruption value) of the sequential element according to the semantics defined in the PA Verification Guidelines for Power Aware Modeling.

Mapping Statement Precedence

The general precedence ordering defined in section [Rule Precedence](#) applies when multiple mapping statements refer to the same region. Specifically, the precedence for mapping statements is as follows:

- A mapping to an instance lower in the design hierarchy takes precedence over a mapping to an instance of a parent scope in the hierarchy.
- A mapping to a process takes precedence over a mapping applied to a parent instance scope.

- A mapping to a signal takes precedence over either a process or instance mapping applied to a parent scope.
- If two or more mapping statements map to the same region and the same Power Aware cell type but map to different retention models, then the first mapping statement is used and a warning message issued.
- The precedence of Power Aware cell types is as follows from highest to lowest precedence:
 - a. Clock/enable specific and sequential element specific types (FF_CKHI, FF_CKLO, LA_ENHI, LA_ENLO)
 - b. Clock/enable free, sequential element specific types (FF_CKFR, LA_ENFR).
 - c. Clock/enable specific, sequential element generic types (ANY_CKHI, ANY_CKLO).
 - d. Clock/enable free, sequential element generic type (ANY_CKFR).

The default mapping for an inferred latch is to LA_ENHI. This default mapping is considered only under the following conditions:

- The enable condition for latches may be complex whether the latch is intentional or unintentional. For enable conditions that are not simply low or high values such as an equality expression, e.g, en_cond = "101", the mapping will be made to an LA_ENHI model (if such a mapping is specified) where the true condition is considered high.
- For latches inferred from a typically unintentional situation, such as the failure to assign all outputs in all branches of a process, the inferred latch will be mapped to a LA_ENHI Power Aware type.
- If there is an ambiguity as to which latch Power Aware type to map an inferred latch, it will be mapped to LA_ENHI.

The power model mapping statement may be terminated by a CRLF or by a semicolon. If terminated by a CRLF, then line continuation must be used if a CRLF is inserted in the middle of the statement. Semicolon termination is encouraged as CRLF is provided primarily for backward compatibility.

Specifying Default Model Mappings

A likely scenario is for a design to use a single technology library from a single vendor. In this situation, repeating the model mapping specifics repeatedly for multiple parts of the design becomes excessively redundant. The DEFAULT_MAP statement is provided for the purpose of defining default mappings of Power Aware models to various Power Aware cell types. To increase the usefulness of the default mapping statements, their lifetimes are defined to extend to the context (scope) they are defined within. To allow a global set of default mappings together with locally overridden mappings, the default mapping statements conform to the precedence rules defined in the sections [Rule Precedence](#) and [Mapping Statement Precedence](#).

Each default mapping statement is independent of other default mapping statements defined within the same scope. This allows the default mapping for one type of Power Aware cell to be overridden in a nested context while other default mappings are not. Furthermore, a [Power Model Mapping Statement](#) overrides any default mapping that might otherwise apply.

```
DEFAULT_MAP model_module_name : pacell_type  
    stmt_end
```

A warning will result if a default mapping within a context (scope) creates an ambiguity with another default mapping within that same context. In such ambiguous situations, the first default mapping will be used and subsequent default mappings ignored.

Retention Statement

Normally, the [Power Control Statement](#) will specify the retention control signal(s) for an entire power island. However, to provide the ability for a power island to have more than one set of retention control signals as well as the flexibility to specify retention control signals separate from the power control statement and the default model mappings, the retention statement is provided. The retention statement may be paired with the use of default mappings to specify the information that is region-specific for inferred sequential elements. Specifically, the retention control signals that are specific to a region are specified so they can be connected to the Power Aware models associated with the default mapping to inferred sequential elements.

```
retention_statement ::=  
    retention_specification, region_definitions  
    stmt_end  
  
retention_specification ::=  
    RETENTION ret_ctrl_signals
```

The retention control signals specified must match the Power Aware model that is mapped to the inferred sequential elements. The [Model Construction for Power Aware Simulation](#) allows the use of one or two retention control signals.

Corruption Semantics

Corruption occurs on power down. Corruption can be applied to state variables (inferred registers and latches) and to the outputs of a block. The corruption of inferred registers and latches and outputs results in a change in the current value of these signal objects in the simulation to the corruption value. No events are propagated as a result in these changes in values. That is, corruption shall not result in the activation of any processes sensitive to the state variables (if signals) or outputs that are corrupted.

Upon restoration of a retained value for inferred registers and latches, the current value of these signal objects will be changed and events are propagated to ensure that the restored state is propagated to outputs. Only inferred registers and latches may have retained values restored. The restored values are specified by the Power Aware model mapped by the PCF to the inferred

registers and latches. For more information, refer to [Model Construction for Power Aware Simulation](#).

Voltage Domains

Voltage domains may be specified along with power islands. There are 2 key pieces of information for a voltage domain:

- Whether power is on or off. This information is already specified by the power control statement.
- The voltage level or range of levels under which the domain operates.

Voltage domain statements are optional. Whereas power control statements specify power islands and retention specifications along with model mappings specify retention sequential objects and both impact the simulated behavior of the design, the voltage statement and voltage specification have no simulation semantics defined.

```
voltage_domain_statement ::=
    POWER_VDD tag region_definitions ,
    voltage_specification
    stmt_end

voltage_specification ::=
    op_voltages
    [, vdd_level_sig ]

op_voltages ::=
    { real_literal {, real_literal} }
```

The voltage domain statement begins with the keyword `POWER_VDD`.

- `tag` — a mnemonic name for the VDD region.
- `region_definition` — the same as defined in section [Region Definitions](#).
- `op_voltages` — a list of one or more real values that specify the allowable voltages under which the domain may operate in units of volts. A list of voltage values provides the ability to specify voltage domains where clock frequency is reduced to lower dynamic power consumption.
- `vdd_level_sig` — (optional) specifies a voltage level signal (or legal Verilog signal expression). This signal must be an integral type. Its value indicates which operating voltage level is in operation at any given point in time. The integral value shall match the positional value of the operating voltages with the value of 0 corresponding to the first operating voltage level specified, 1 corresponding to the 2nd operating voltage level specified, etc.

Comments

A comment may appear anywhere on the line. The start of a comment is denoted by "--". The comment extends to the end of the line.

```
comment ::=  
  -- <any text> CRLF
```

Regular Expressions and Variables

In order to make it easy to group instances, signals and labels within instances, regular expressions are supported. The common subset of syntax and rules of Perl and Tcl are supported. More specifically, the following sets of meta-characters are supported.

- Single Character Matches — "." or "[]", for example:

```
"/ab.c/" -- a, b, any character, c  
"/a[bc]d/" -- abd, or acd
```

- Multiple Character Matches — "*" or "{n,m}", for example:

```
"/ab*c/" -- a, zero or more b's, c  
"/ab{2,4}c/" # a, two to four b's, c
```

The only difference is that the regular expression will be delimited by quotation marks.

Examples

The following matches all the instances 'DFF_0' to 'DFF_99' within the 'top/pd11' instance.

```
POWER PD0  
top/pd11/"DFF_[0- 9]{1,2}",  
(top/global_vdd1 & top/vd1/local_vdd_11);
```

The following matches all instances 'DFF_0' to 'DFF_99' for any sub-instance with a name beginning with 'pd2' under 'top'.

```
POWER PD2  
top/"pd2.*"/"DFF_[0- 9]{1,2}",  
(top/global_vdd1 & top/vd1/local_vdd_12);
```

Rule Precedence

When more than one rule applies to a given power element, the rule defined at the lowest hierarchical level takes precedence over the rules defined at higher hierarchical level. This applies to power control statements, power model mapping statements and voltage domain statements.

When a rule is defined for an instance and a rule is defined for a signal belonging to that instance (at same hierarchical level), then, for that signal, the rule defined for the signal takes precedence over the rule defined for the process. In case more than one rule is defined at the same power element, a fatal error shall be reported and no verification will be terminated.

Appendix E

Supplemental Information

This appendix provides supplemental information on applications of Power Aware.

Power Aware Verification of ARM-Based Designs

This section contains an application note written by Ping Yeung and Erich Marschner of Mentor Graphics Corporation.

Abstract

Power dissipation has become a key constraint for the design of today's complex chips. Minimizing power dissipation is essential for battery-powered portable devices, as well as for reducing cooling requirements for non-portable systems. Such minimization requires active power management built into a device.

In a System-on-Chip (SoC) design with active power management, various subsystems can be independently powered up or down, and/or powered at different voltage levels. It is important to verify that the SoC works correctly under active power management. When a given subsystem is turned off, its state will be lost, unless some or all of the state is explicitly retained during power down. When that subsystem is powered up again, it must either be reset, or it must restore its previous state from the retained state, or some combination thereof. When a subsystem is powered down, it must not interfere with the normal operation of the rest of the SoC.

Power aware verification is essential to verify the operation of a design under active power management, including the power management architecture, state retention and restoration of subsystems when powered down, and the interaction of subsystems in various power states. In this presentation, we summarize the challenges of power aware verification and describe the use of IEEE Std 1801™-2009 UPF to define power management architecture. We outline the requirements and essential coverage goals for verifying a power-managed ARM-based SoC design.

Introduction

The continual scaling of transistors and the end of voltage scaling has made power one of the critical design constraints in the design flow. Trying to maintain performance levels and achieve faster speeds by scaling supply and threshold voltages increases the subthreshold leakage current due to its exponential relationship with the threshold voltage [1]. Leakage currents lead to power dissipation even when the circuit is not doing any useful work, which limits operation time between charges for battery-operated devices, and creates a heat dissipation problem for all devices.

Minimizing power dissipation starts with minimizing the dynamic power dissipation associated with the clock tree, by turning off the clock for subsystems that are not in use. This technique has been in use for many years. But at 90nm and below, static leakage becomes the dominant form of power dissipation. Active power management minimizes static leakage through various techniques, such as shutting off the power to unused subsystems or varying the supply voltage or threshold voltage for a given component to achieve the functionality and performance required with minimum power.

Active Power Management

Active power management can be thought of as having three major aspects:

- the power management architecture, which involves the partitioning of the system into separately controlled power domains, and the logic required to power those domains; mediate their interactions, and control their behavior;
- the power managed behavior of the design, which involves the dynamic operation of power domains as they are powered up and down under active power management, as well as the dynamic interactions of those power domains to achieve system functionality;
- the power control logic that ultimately drives the control inputs to the power management architecture, which may be implemented in hardware or software or a combination thereof.

All three of these aspects need to be verified to ensure that the design will work properly under active power management. Ideally such verification should be done at the RTL stage. This enables verification of the active power management capability much more efficiently than would be possible at the gate level, which in turn allows more time for consideration of alternative power management architectures and simplifies debugging.

Power Management Techniques

Several power management techniques are used to minimize power dissipation: clock gating, power gating, voltage scaling, and body biasing are four of them. Clock gating disables the

clock of an unused device, to eliminate dynamic power consumption by the clock tree. Power gating uses a current switch to cut off a circuit from its power supply rails during standby mode, to eliminate static leakage when the circuit is not in use. Voltage scaling changes the voltage and clock frequency to match the performance required for a given operation so as to minimize leakage. Body biasing changes the threshold voltage to reduce leakage current at the expense of slower switching times.

Power gating is one of the most common active power management techniques. Switching off the power to a subsystem when it is not in use eliminates the leakage current in that subsystem when it is powered down, and hence the overall leakage power dissipation through that subsystem is reduced. However, this technique also results in loss of state in the subsystem when it is switched off. Also, the outputs of a power domain can float to unpredictable values when they are powered down.

Another common technique is the use of different supply voltage levels for different subsystems. A subsystem that has a higher voltage supply can change state more quickly and therefore operate with higher performance, at the expense of higher static leakage and dynamic power. A subsystem with a lower voltage supply cannot change state as quickly, and consequently operates with lower performance, but also with less static leakage and dynamic power. This technique allows designers to minimize static leakage in areas where higher performance is not required.

Multiple voltage supplies can also be used for a single subsystem, for example, by enabling it to dynamically switch between a higher voltage supply and a lower voltage supply. This allows the system to select higher performance for that subsystem when necessary, but minimize static leakage when high performance is not required. Multi-voltage and power gating techniques can be combined to give a range of power/performance options.

All of these power management techniques must be implemented in a manner that preserves the intended functionality of the design. This requires creation of power management logic to ensure that the design operates correctly as the power supplies to its various components are switched on and off or switched between voltage levels. Since this power management logic could potentially affect the functionality of the design, it is important to verify the power management logic early in the design cycle, to avoid costly respins.

Power Management Specification

The power management architecture for a given design could be defined as part of the design, and ultimately it will be a part of the design's implementation. A better approach, however, is to specify the power management architecture separate from the design. This simplifies exploration of alternative power management architectures, reduces the likelihood of unintended changes to the golden design functionality, and maintains the reusability of the design data. This is the approach supported by IEEE Std 1801™-2009, "Standard for Design and Verification of Low Power Integrated Circuits." This standard is also known as the Unified Power Format (UPF) version 2.0. Initially developed by Accellera, UPF is currently supported by multiple vendors and is in use worldwide [5].

UPF provides the concepts and notation required to define the power management architecture for a design. A UPF specification can be used to drive the implementation of power management for a given design, during synthesis or subsequent implementation steps. A UPF specification can also be used to drive verification of power management, during RTL simulation, gate-level simulation, or even via static verification methods. The ability to use UPF in conjunction with RTL simulation enables early verification of the power management architecture. The ability to use UPF across all of these applications eases implementation and validation by enabling reuse of power management specifications throughout the flow.

UPF syntax is defined as an extension of Tcl [6], which enables UPF descriptions to leverage all of the control features of Tcl. UPF captures the power management architecture in a portable form for use in simulation, synthesis, and routing, reducing potential omissions during translation of that intent from tool to tool. Because it is separate from the HDL description and can be read by all of the tools in the flow, the UPF side file is as portable and interoperable as the logic design's HDL code.

The concepts introduced in the following sections are illustrated with the UPF commands used to specify them.

Power Management Architecture

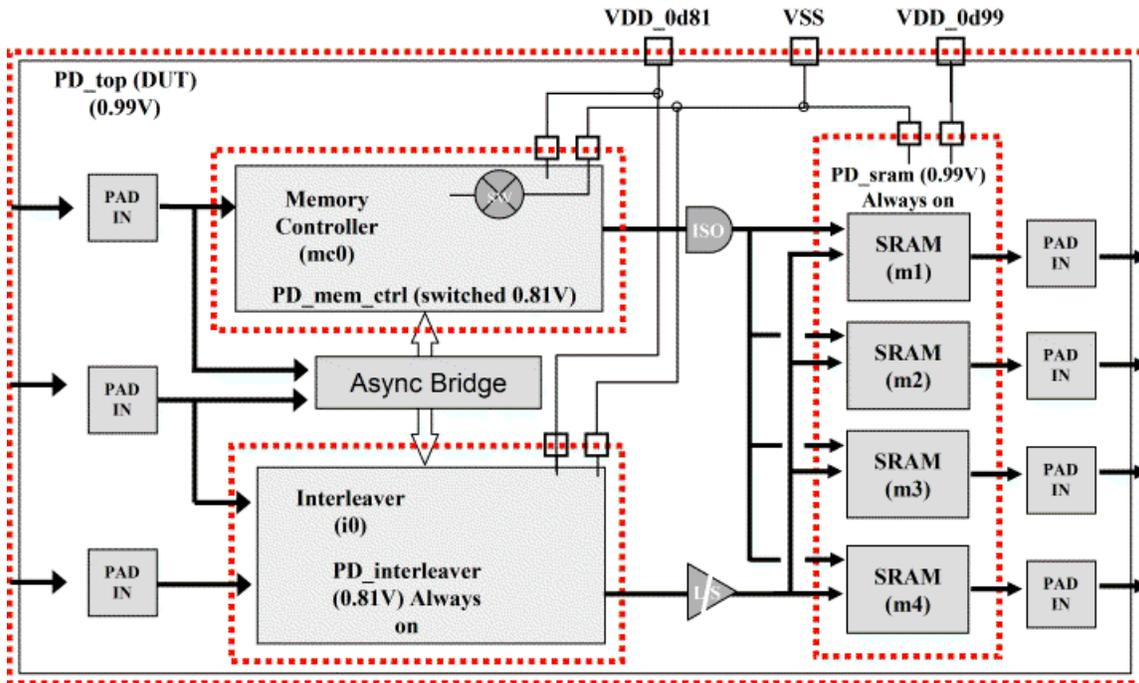
In order to employ active power management techniques such as power gating and multiple voltage supplies, the design must be partitioned into separate functional areas that can be independently powered. Additional logic must be inserted into the design to perform special functions such as power switching, state retention, isolation, and level shifting. These additional components constitute the power management architecture for a given system.

Operating Modes

Designing the power management architecture for a given system starts with characterization of the functions and operating modes of the system. Since the goal of active power management is to optimize the use of power based on the function and performance required of the system at any given time, the first step involves identifying the distinct combinations of functionality and performance that will be required of the device in use. Analysis of the set of distinct operating modes allows the designer to determine how to partition the design into independently powered subsystems or subcomponents, so that any given operating mode can be supported by providing the necessary subset of system components with the appropriate power.

For example, [Figure E-1](#) shows a block diagram of a design that has two operating modes: ON and SLEEP.

Figure E-1. A Power-Managed Design



In the ON mode, it reads input data streams, interleaves them, and stores them in the memory before driving them onto outputs. In the SLEEP mode, it monitors inputs and maintains the state of its memory, but it does not process inputs.

Power Domains

Each independently powered subsystem or subcomponent is called a power domain. At the RTL stage, a power domain is typically somewhat abstract, consisting of some or all of the RTL logic within a given portion of the design hierarchy. At the logical netlist stage, a power domain consists of a collection of cells that will share the same primary power and ground supplies. At the physical level, the cells associated with a given power domain may be placed in a contiguous region of a chip or distributed over multiple discontinuous regions of the chip.

The design in Figure E-1 has several major components. These include the interleaver block, the memory controller block, and the memory itself. Each of these can be defined as a separate power domain. The top-level of the design is also a separate power domain. The dotted lines in Figure E-1 indicate power domain boundaries. The following UPF commands would be used to define these power domains:

```
#-----
# Create power domains
#-----
create_power_domain PD_top
create_power_domain PD_interleaver -elements {i0}
create_power_domain PD_mem_ctrl -elements {mc0}
create_power_domain PD_sram -elements {m1 m2 m3 m4}
```

The `-elements` option on each command lists the instance names of the elements to be included in the specified power domain.

Power Distribution

Each power domain may have one or more power supplies. The primary supply provides power for most of the functional elements in that domain. Additional supplies may provide power for retention, isolation, or level shifting cells associated with the power domain.

The primary supply may be a switched supply, which can be turned on and off via a control input to the switch. Either the VDD or VSS supply may be switched. A supply may be driven by multiple switches connected to the same voltage source. The switches are turned on incrementally, to minimize rush currents when the supply is switched on. A supply may also be driven by multiple switches connected to different voltage sources, so that the supply voltage level delivered to elements of the power domain may be varied. Switches may be on-chip or off-chip.

Power is distributed to power domains via supply ports interconnected by supply nets. Supply ports may represent external supplies or may be driven by internal supply sources. Supply ports are connected to supply nets, each of which is ultimately connected to a power domain. Each supply port has one or more supply states defined. The port may drive only one state at any given time. That state is propagated by the supply net connected to the port.

For the example in [Figure E-1](#), the following UPF commands could be used to define top-level supply ports, and to define and connect supply nets to those ports:

```
#-----  
# Create top level power domain supply ports  
#-----  
create_supply_port VDD_0d99 -domain PD_top  
create_supply_port VDD_0d81 -domain PD_top  
create_supply_port VSS -domain PD_top  
  
#-----  
# Create top level power domain supply nets  
#-----  
create_supply_net VDD_0d99 -domain PD_top  
create_supply_net VDD_0d81 -domain PD_top  
create_supply_net VSS -domain PD_top  
  
#-----  
# Connect top level power domain supply ports  
# to supply nets  
#-----  
connect_supply_net VDD_0d99 -ports VDD_0d99  
connect_supply_net VDD_0d81 -ports VDD_0d81  
connect_supply_net VSS -ports VSS
```

The following UPF command would be used to identify a particular pair of supply nets as the primary power and ground supplies for a given power domain:

```
#-----
# Set the default for top level power domain
#-----
set_domain_supply_net PD_top \
    -primary_power_net VDD_0d99 \
    -primary_ground_net VSS
```

Additional UPF commands could be used to propagate the top-level supply nets into subordinate power domains and to define a power switch to create a switched ground (VSS_SW) for one of the power domains:

```
#-----
# Create sub domain supply nets
#-----
create_supply_net VDD_0d81 -domain PD_interleaver -reuse
create_supply_net VDD_0d81 -domain PD_mem_ctrl -reuse
create_supply_net VDD_0d99 -domain PD_sram -reuse

create_supply_net VSS -domain PD_interleaver -reuse
create_supply_net VSS -domain PD_mem_ctrl -reuse
create_supply_net VSS -domain PD_sram -reuse

#-----
# Create supply net for switch output
#-----
create_supply_net VSS_SW -domain PD_mem_ctrl

#-----
# Create power switch for memory controller domain
# - switch on ground side of supply network
#-----
create_power_switch mem_ctrl_sw \
    -domain PD_mem_ctrl \
    -output_supply_port {vout_p VSS_SW} \
    -input_supply_port {vin_p VSS} \
    -control_port {ctrl_p mc_pwr_c} \
    -on_state {normal_working vin_p {ctrl_p}} \
    -off_state {off_state {!ctrl_p}}
```

Power distribution logic may also include on-chip analog components such as regulators and sensors. A regulator takes an input supply voltage and generates a specific output voltage. A sensor monitors a supply rail and signals when the voltage has stabilized at its nominal value with respect to ground. Sensors enable construction of a feedback loop so that power control logic can determine when a power rail has completed transitioning. Analog components such as these are not specifiable in UPF, but can be modeled in HDL code using UPF package functions to model the ramp-up and ramp-down of power supplies as they switch on and off.

Power States

UPF provides commands for defining a power state table that captures the possible power states of the system. The power state table defines system power states in terms of the states of supply ports or nets.

For the example in [Figure E-1](#), the following UPF commands define the possible states of the supply ports VDD_0d81, VDD_0d99, and VSS, as well as the switched ground supply VSS_SW:

```
#-----  
# Define power states  
#-----  
add_port_state VDD_0d99 -state {ON 0.99 1.10 1.21}  
add_port_state VDD_0d99 -state {OFF off}  
  
add_port_state VDD_0d81 -state {ON 0.81 0.90 0.99}  
add_port_state VDD_0d81 -state {OFF off}  
  
add_port_state VSS      -state {ON 0 0 0}  
add_port_state VSS_SW   -state {ON 0} -state {OFF off}  
  
#-----  
# Create power state table  
#-----  
create_pst top_pst \  
    -supplies { VDD_0d99  VDD_0d81  VSS  VSS_SW }  
  
add_pst_state ON \  
    -pst top_pst -state {    ON          ON      ON      ON    }  
add_pst_state SLEEP \  
    -pst top_pst -state {    ON          ON      ON      OFF   }  
add_pst_state OFF \  
    -pst top_pst -state {    OFF         OFF     ON      OFF   }
```

The power states of the system in [Figure E-1](#) are defined in the above power state table. Note that these power states are the same as the operating modes of the system, plus the state in which the system is completely turned off.

Isolation and Level Shifting

Even though each power domain may be independently powered on and off, their logical and physical connections to other power domains remain; therefore, when one domain is turned off, it is still connected logically and electrically to other domains. These connections between power domains require special cells to mediate the interaction between domains as their respective power states change. Two kinds of cells are involved: isolation cells, and level shifting cells.

Isolation cells ensure that signals coming from unpowered domains are clamped to a well-defined logic value while the source domain is powered down, so that any sink domain that is powered up sees reliable inputs. Depending upon the architecture of the design, and the

particular characteristics of a signal that crosses from one power domain to another (e.g., how many power domains it fans out to, and when those power domains are on or off with respect to the source domain), it may be appropriate to insert isolation cells at either the source of the signal or at its sink(s). However, since the isolation cell must be powered on when the source domain is powered off, isolation cells are typically powered by a separate, “always-on” supply voltage.

The following UPF commands specify the addition of isolation for the PD_mem_ctrl power domain in the example in [Figure E-1](#). The first command defines the supplies powering the isolation cell and specifies its clamp value. The second command defines the control signal for the isolation cell.

```
#-----
# Setup isolation strategy for memory controller
#-----

# Mem ctrl chip & write enables: clamp to '1'

set_isolation mem_ctrl_iso_1 \
  -domain PD_mem_ctrl \
  -isolation_power_net VDD_0d99 \
  -isolation_ground_net VSS \
  -clamp_value 1 \
  -elements {mc0/ceb mc0/web}

set_isolation_control mem_ctrl_iso_1 \
  -domain PD_mem_ctrl \
  -isolation_signal mc_iso_c \
  -isolation_sense high \
  -location parent
```

Level shifting cells ensure that a signal coming from a power domain operating at one voltage is correctly interpreted when it is received by a power domain operating at a different voltage. Depending upon the relative voltage levels of the two power domains, a level shifter may increase or decrease the operating voltage of the signal. As with isolation cells, level shifters may have separate power supplies that are always on, or they may be powered by the primary supplies of the source and sink domains, respectively.

```
#-----
# Define level shifters
#-----

set_level_shifter interleaver_ls_in \
  -domain PD_interleaver \
  -applies_to inputs \
  -location self

set_level_shifter interleaver_ls_out \
  -domain PD_interleaver \
  -applies_to outputs \
  -location parent
```

The UPF commands above specify addition of level shifters for the PD_interleaver power domain in the example in [Figure E-1](#). The first command specifies addition of level shifters for inputs; the second command specifies addition of level shifters for outputs. Whether level shifters will actually be inserted depends upon the respective supply voltages of the source and sink domains involved.

State Retention

When a power domain is powered down, any normal state elements within the power domain will lose their state. When the power domain is powered on again, the power domain must be brought to a predictable state again. This may involve resetting all state elements in the domain, or resetting some subset that will be sufficient to cause the rest of the domain to reach a well-defined state after a few clock cycles. Another alternative is to save the state of certain state elements before the domain is powered down, and restore those statements to their saved state after the power domain is powered up again.

Retention cells are special memory elements that preserve their data during power down. Such cells involve extra logic and possibly complex timing to save and restore their values across powered-down periods [2]. Various kinds of retention cells have been designed [2], [3], [4]. Some of these use balloon latch mechanisms [2], which are made up of high threshold transistors to minimize leakage through them. They are separated from the critical path of the design by transmission gates and thus are not required to be timing critical. Others depend on complex sequences of different controls to achieve data retention.

The following UPF command specifies where retention should occur in the example in [Figure E-1](#).

```
#-----  
# Setup retention strategy for mem cntrl  
#-----  
set_retention mem_ctrl_ret \  
  -domain PD_mem_ctrl \  
  -retention_power_net VDD_0d81 \  
  -save_signal {mc_save_c high} \  
  -restore_signal {mc_restore_c low}
```

This command identifies the power domain (PD_mem_ctrl) within which retention registers should be used, specifies the power supply used to maintain retained values, and specifies the control signals required for saving and restoring the values of retention registers.

Power Managed Behavior

With the power management architecture implemented on top of the design, it should be possible to repeatedly power up each power domain and later power it down again. Each time a domain is powered up, it should reach a well-defined state from which to continue its

operations. That state may be the initial reset state, or a state saved before the power domain was last powered down, or a combination of the two.

While a power domain is powered down, its elements cannot drive their outputs to well-defined logic values. As a result, those outputs may float to 1 or float to 0, or may be at an intermediate value. In this situation, those outputs are considered corrupted. This is not a problem as long as no other active power domain sinks those corrupted values; otherwise logical and/or electrical problems could result. During the power down process, it is essential for the isolation cells in the design to be enabled before the domain's primary supply is shut off, for those isolation cells to clamp signals from the powered-down domain to appropriate values, and for the isolation cells to remain enabled until the shut-off domain's primary supply is turned on again.

Similarly, when two interconnected power domains have been put into respective power states that involve different supply voltages, the level shifters in the design must convert logic 1 signal voltage levels in the source domain to logic 1 signal voltage levels in the sink domain. Although level shifters function continuously and therefore do not need to be enabled, dynamic changes in the supply voltages for the respective power domains may result in unexpected situations.

Power Control Logic

Power management may involve both software and hardware control. For example, a power control unit (PCU) can be specified in RTL internal to the SoC. The PCU may be under software control by an embedded processor. The combination of these power management controls drive the signals that define the PCN, based on the system's power management strategy—signaling power domains to retain state, enable isolation, power down (turn off switches), power up (turn on switches), disable isolation, and restore state.

Correct operation of the power management architecture depends upon correct sequencing of power control signals. For example, outputs of a domain must be isolated before the power is shut off, and must remain isolated until after power is turned on again. Thus the control signal initiating isolation must come before the control signal that turns off the power switch. Similarly, the control signal that turns on the power switch must occur before the control signal that terminates isolation. In fact, turning power on and off may involve handshaking between the PCU and the supply source (or a sensor monitoring the supply) to ensure that voltage-dependent delays in ramping up or down the power supply are factored into control signal sequencing.

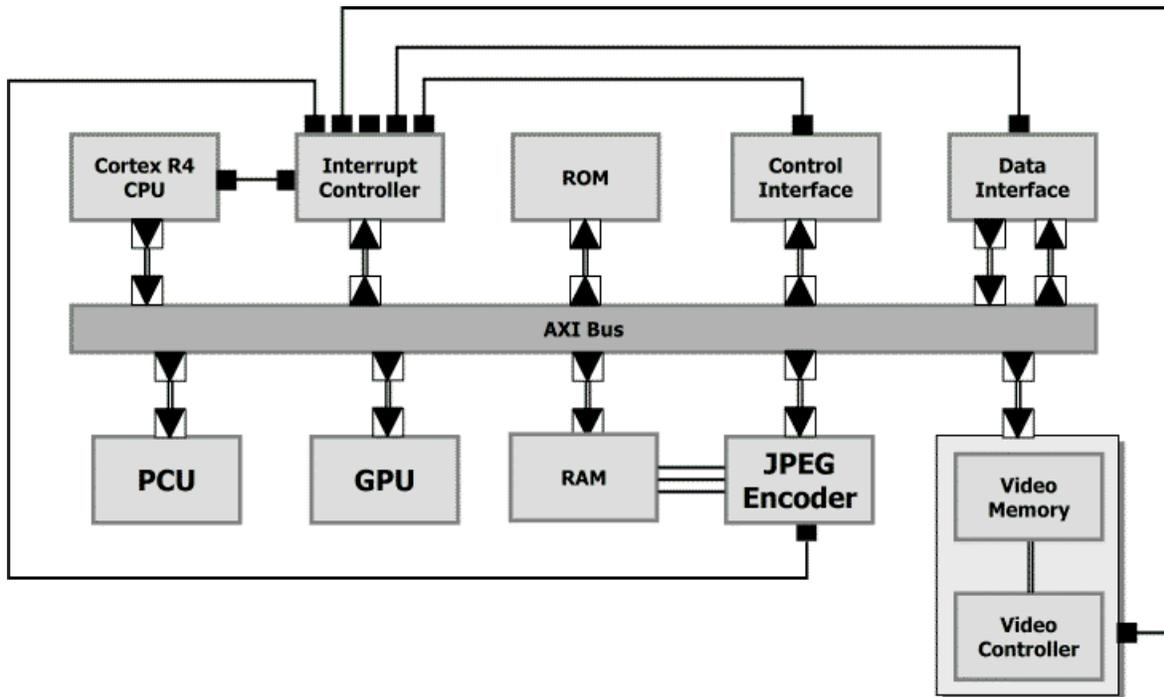
Power Aware Verification Flow

Verifying RTL-level specification of active power management for a given design involves several steps. First, we need to verify that the power management architecture is correctly structured, given the operating modes of the device and the power states that have been defined to reflect those modes. Second, we need to verify that the design (both each power domain individually and all of them collectively) behave correctly when power management control

signals are given in the correct sequence. Third, we need to verify that the power control logic will always generate power control signals in the correct sequence.

Figure E-2 shows the high-level design of an ARM-based SoC with active power management. The above verification steps as applied to this example are described below.

Figure E-2. An ARM-based SoC with Active Power Management



This design consists of multiple functional units communicating over the AXI bus. Each functional unit may be defined as a separate power domain, or even as a collection of power domains. A UPF file for this design would specify the power management architecture for the whole system, including the specific requirements for power distribution, switching, and state retention for each power domain, and the requirements for isolation and level shifting between interacting power domains. The Power Control Unit (PCU) is a hardware implementation of power control logic that drives power control signals for each domain in the correct order. The Cortex R4 CPU is an embedded ARM processor that drives system-level power state changes by sending transactions to the PCU.

Verifying the Power Management Architecture

Verifying the sufficiency of the power management architecture can be done in part through static analysis. Given a complete definition of the power domains and power states for a given design, it is relatively straightforward to verify that the necessary isolation cells and level shifters are present (or implied by a UPF specification) to ensure that the power domains will interact correctly and will not be adversely affected when their neighboring power domains are

powered down. Static analysis can also ensure that the necessary supply structures are present to provide the ability to control power to each power domain.

However, static analysis is not always possible. Depending upon the sequencing of power state changes, and the ramping of power supplies as they transition, there may be a requirement for level shifters that is not obvious from the power state table. Also, the power state table may not be complete, and power states that are not defined might actually occur during operation of the device. Finally, external supply sources may be switched or may vary in voltage beyond what is defined in the power state table. For these and other reasons, simulation is often required. In this case, power aware simulation is necessary, to ensure that the power management architecture and its controls are taken into account during simulation.

Power Aware simulation enables functional verification of power management in the context of an RTL design. A power aware simulation run does the following:

- Compiles the design and UPF specifications
- Infers sequential elements from the RTL design (registers, latches and memories)
- Applies the UPF-specified power management architecture to the RTL design
- Augments the simulation model with appropriate power aware models
- Dynamically modifies the RTL behavior to reflect the impact of active power management.

Using the UPF and sequential element information, the simulator is able to augment the normal RTL behavior with the UPF-specified power aware behavior (power distribution and control, retention, corruption, and isolation). This involves selecting the appropriate simulation models to implement the UPF-specified power management architecture. It may also involve recognizing and integrating user-supplied power aware simulation models.

Verifying Power Managed Behavior

Power aware simulation can be used to visualize the effects of active power management on the dynamic behavior of the design, as well as visualizing the behavior of the power management architecture itself under control of power management logic. In a power aware simulation, the internal state and outputs of a power domain will be set to X to reflect the corruption of those signals when the primary supply to that domain is turned off. When the supply is turned on again, the X values will be replaced as the power domain reinitializes or has its state restored. Signals driven by outputs of a powered-down domain should be clamped to 0 or 1, so that downstream power domains see a well-defined value and won't be affected by corrupted outputs of a power domain that has been powered down. Retention should be evident in that the state of signals following power up will correspond to the state of signals prior to the previous power down.

Visualizing the effects of active power management helps the designer confirm that all of the necessary power domains and power states required to implement the operation modes of this

device have been defined, and that all the necessary isolation, level-shifting, and retention cells necessary to enable power management have been added. If there are errors in the power management architecture, they will very likely cause signal corruption that does not go away after power up, which in turn will lead to functional errors in the design.

Debugging power management errors can be performed by tracking corruption of signals in the waveform view, but that method is tedious and error-prone. A much more effective method is the use of assertions to check for correct operation of the design under active power management. For example, an assertion to check that an output of a power domain is clamped to the correct value when the power domain is powered down will immediately catch any error related to the clamp value, or the powering of the isolation cell involved, rather than just generating an X and letting it propagate. Such assertions can be automatically generated by the power aware simulator.

Verifying Power Control Logic

Power aware simulation can also be used to verify the control logic driving the power management architecture, provided that the control logic is part of the design rather than being implemented in a test bench. For software-based power control logic, simulation is the only method available. In particular, hardware/software co-simulation is necessary if the power control logic is split between hardware and software components, as is often the case. For hardware-based power control logic, such as a power control unit, another alternative is available.

Formal verification is particularly suited to verifying complex control logic. In contrast to simulation, which runs one input sequence at a time to test a device, formal verification considers all valid input sequences in one pass. A formal verification tool can therefore identify all possible behaviors of the power control logic, which enables it to automatically find any corner cases in which the generated control sequences may not be complete or in the correct order. Formal verification is driven by assertions, so use of formal verification requires creation of assertions about the expected behavior of the power control unit. Although this takes some effort, the ability to thoroughly verify the power control logic makes it worthwhile.

Summary

Active power management is becoming a necessary part of today's SoC designs. To add active power management to a design and verify that it is working correctly, it is critical to have a well-defined methodology that addresses all aspects of active power management. The methodology needs to support defining and verifying an appropriate power management architecture, verifying that the design behaves correctly under the power management architecture, and verifying that the power control signals controlling the power management architecture are generated correctly. IEEE Std 1801™-2009 UPF supports such a methodology, as does static analysis of power management architecture, power-aware simulation of power-managed designs, and formal verification of power control logic. These methods provide a comprehensive solution for defining and verifying active power management.

Acknowledgements

The authors would like to acknowledge the thoughtful commentary and suggestions for improvement provided by Barry Pangrle on the penultimate draft of this paper.

References

1. N.S. Kim, T. Austin, T. Blaauw, T. Mudge, K. Flautner, H.S. Hu, M.J.Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *IEEE Computer*, 36(12):68--75, 2003.
2. S. Shigematsu, S. Mutoh, Y. Matsuya, Y. Tanabe and J. Yamada, "A 1-V High-Speed MTCMOS Circuit Scheme for Power-Down Application Circuits," *IEEE J. Solid-State Circuits*, Vol. 32, No. 6, pp. 861--869, 1997.
3. Hyo-Sig Won; Kyo-Sun Kim; Kwang-Ok Jeong; Ki-Tae Park; Kyu-Myung Choi; Jeong-Taek Kong, "An MTCMOS design methodology and its application to mobile computing," *Low Power Electronics and Design*, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on, vol., no., pp. 110-115, 25-27 Aug. 2003.
4. Zyuban, V.; Kosonocky, S.V., "Low power integrated scan-retention mechanism," *Low Power Electronics and Design*, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on, vol., no., pp. 98-102, 2002.
5. IEEE 1801™-2009, "Standard for Design and Verification of Low Power Integrated Circuits", IEEE.
6. Tcl/Tk Documentation, Tcl Developer Xchange, <http://www.tcl.tk>.

— A —

Accellera, 171

— B —

Backus-Naur Format, 300

BNF, 300

— D —

Design flow, 14

— F —

Feedthrough, 157

— H —

Hard macro, 152

— I —

IEEE standard for low power, 18

— L —

Liberty libraries, 35

 database, 35

Low power, 13

 IEEE standard for, 18

Lower boundary ports, 238

— M —

Macro, hard, 152

Macromodels, 152

Messages, 136

MTI_LIBERTY_PATH, 39

Multi-voltage analysis, 145

— N —

Named events, 160

— P —

PA-GL, 13

PA-RTL, 13

PCF (Power Configuration File), 299

Power Aware, 13

 documentation, 16

Power Aware verification, 170

Power Configuration File, 299

Power gating, 13

Power intent specification file, 19

Power State Table (PST), 145, 203

PST, 145, 203

— U —

Unified Power Format (UPF), 19, 169

UPF, 19, 169, 170

 supported commands, 172

— V —

Value conversion table (VCT), 280

VCT, 280

Verilog

 named events, 160

Voltage level-shifting, 145

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented via any electronic portal or other automated order management system will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products,

whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.
4. **BETA CODE.**
 - 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
 - 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
 - 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.
5. **RESTRICTIONS ON USE.**
 - 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.
 - 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
 - 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/about/legal/>.

7. **LIMITED WARRANTY.**

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INFRINGEMENT.**

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

- 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
- 11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
- 11.4. **THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.**

12. TERMINATION AND EFFECT OF TERMINATION.

- 12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.
- 12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.
14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.
15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if

Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 130502, Part No. 255853