

Power Compiler™ User Guide

Version U-2022.12-SP3, April 2023

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2023 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

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

Free and Open-Source Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

New in This Release	17
Related Products, Publications, and Trademarks	17
Conventions	18
Customer Support	19

Part 1: Power Compiler Concepts

1. Introduction to the Power Compiler Tool	21
Power Compiler Methodology	21
Power Library Models	22
Power Analysis	23
Power Optimization	24
Getting Started With the Power Compiler Tool	24
Library Requirements	25
Command-Line Interface	25
Graphical User Interface	26
License Requirements	26
Reading and Writing Designs	27
2. Power Compiler Design Flow	28
Power in the Design Cycle	28
Power Optimization and Analysis Flow	29
Simulation	31
Enable Power Optimization	31
Synthesis and Power Optimization	31
Power Analysis and Reporting	31
3. Power Modeling and Calculations	32
Power Types	32
Static Power	32

Contents

Dynamic Power	32
Switching Power	33
Internal Power	33
Calculating Power	34
Leakage Power Calculation	34
Multithreshold Voltage Libraries	36
Internal Power Calculation	36
NLDM Models	37
State and Path Dependency	39
Rise and Fall Power	40
Switching Power Calculation	40
Dynamic Power Calculation	40
Dynamic Power Unit Derivation	41
Power Calculation for Multirail Cells	42
Using CCS Power Libraries	43
Voltage Scaling	44
Script Examples for Voltage Scaling	44

Part 2: Power Analysis

4. Generating SAIF Files	47
About Switching Activity	47
Introduction to SAIF Files	48
Generating SAIF Files	49
Generating SAIF Files From Simulation	49
Generating SAIF Files From SystemVerilog or Verilog Simulations	50
Generating SAIF Files From Gate-Level Simulation	51
VCS MX Toggle Commands	52
Generating SAIF Files From VHDL Simulation	58
Generating SAIF Files From VCD Files	59
Converting a VCD File to a SAIF File	60
Generating SAIF Files From FSDB Output Files	60
Verilog Switching Activity Examples	61
RTL Example	61
Verilog Design Description	61
RTL Testbench	62
RTL SAIF File	63

Contents

Gate-Level Example	65
Gate-Level Verilog Module	65
Verilog Testbench	66
Gate-Level SAIF File	67
VHDL Switching Activity Example	68
VHDL Design Description	68
RTL Testbench	68
RTL SAIF File	69
<hr/>	
5. Annotating Switching Activity	70
Types of Switching Activity to Annotate	70
Annotating Switching Activity Using RTL SAIF Files	71
Using the Name-Mapping Database	71
Integrating the RTL Annotation With the PrimePower tool	73
Annotating Switching Activity Using Gate-Level SAIF Files	74
Reading SAIF Files Using the <code>read_saif</code> Command	74
Reading SAIF Files Using the <code>merge_saif</code> Command	75
Annotating Inferred Switching Activity	76
Annotating Switching Activity Using the <code>set_switching_activity</code> Command	77
Fully Versus Partially Annotating the Design	79
Analyzing Switching Activity Annotation	80
Using the <code>report_saif</code> Command to Report Switching Activity	81
Using the <code>report_activity</code> Command to Report Switching Activity	81
Retrieving Switching Activity on a Pin or Net	82
Removing the Switching Activity Annotation	82
Design Objects Without Annotated Switching Activity	83
Default Switching Activity Values	83
Propagating the Switching Activity	84
Deriving the State- and Path-Dependent Switching Activity	84
<hr/>	
6. Performing Power Analysis	85
Overview	85
Identifying Power and Accuracy	86
Factors That Affect the Accuracy of Power Analysis	87
Switching Activity Annotation	87

Contents

Delay Model	87
Switching Activity Correlation	88
Overriding Library Cell Power Characterization	88
Performing Gate-Level Power Analysis	89
Using the report_power Command	89
Using the report_power_calculation Command	91
Analyzing Power With Partially Annotated Designs	92
Power Correlation	93
Performing Power Correlation	93
Power Correlation Script	94
Analyzing the Design For Power Analysis	94
Characterizing a Design for Power	95
Reporting the Power Attributes of Library Cells	97
Using Power Derate Factors	97
Generating Power Reports	98
Power Report Summary	98
Net Power Report	100
Cell Power Report	101
Group Report	102
Wire and Pin Switching Power Report	103
Hierarchical Power Report	103
Power Report for Block Abstractions	104
Register Clock-Pin Internal Power Report	105

Part 3: Power Reduction

7. Clock Gating	109
Introduction to Clock Gating	110
Using Clock-Gating Conditions	112
Clock-Gating Conditions	112
Enable Condition	113
Setup Condition	115
Enabling or Disabling Clock Gating on Design Objects	116
Inserting Clock Gates	117
Using the compile_ultra -gate_clock Command	117

Contents

Clock-Gate Insertion in Multivoltage Designs	117
Clock Gating Flows	117
Inserting Clock Gates in the RTL Design	118
Inserting Clock Gates in Gate-Level Design	119
Specifying Clock-Gate Latency	120
The <code>set_clock_latency</code> Command	120
The <code>set_clock_gate_latency</code> Command	121
Applying Clock-Gate Latency	123
Resetting Clock-Gate Latency	123
Comparison of the Clock-Gate Latency Specification Commands	123
Calculating the Clock Tree Delay From Clock-Gating Cell to Registers	124
Specifying Setup and Hold	125
Predicting the Impact of Clock Tree Synthesis	127
Choosing a Value for Setup	128
Choosing a Value for Hold	129
Clock-Gating Styles	129
Default Clock-Gating Style	130
Selecting Clock-Gating Styles	132
Choosing Gating Logic	133
Choosing an Integrated Clock-Gating Cell	133
Choosing a Configuration for Discrete Gating Logic	135
Choosing a Simple Gating Cell by Name	139
Choosing a Simple Gating Cell and Library by Name	139
Designating Simple Cells Exclusively for Clock Gating	140
Choosing a Specific Latch and Library	141
Choosing a Latch-Free Style	141
Improving Testability	142
Connecting the Test Ports Throughout the Hierarchy	147
Using Instance-Specific Clock-Gating Styles	148
Modifying the Clock-Gating Structure	150
Changing a Clock-Gated Register to Another Clock-Gating Cell	151
Removing Clock-Gating Cells From the Design	151
Rewiring Clock Gating After Retiming	152
Integrated Clock-Gating Cells	152
Integrated Clock-Gating Cell Attributes	153
Pin Attributes	154
Timing Considerations	155
Clock-Gating Naming Conventions	156

Contents

Example Script for Naming Style	158
Example Script of Output Netlist	158
Keeping Clock-Gating Information in a Structural Netlist	159
Identifying and Preserving Clock-Gating Cells	159
Identification of Clock-Gating Cells	159
Explicit Identification of Clock-Gating Cells	160
Preserving the Identified Clock-Gating Cells	161
Identified Clock-Gating Cells and <code>dont_touch</code>	163
Handling Clock-Gating Edge Conflicts	163
Comparison of Clock-Gate Identification Methods	164
Usage Flow With the <code>write_script</code> Command	164
Usage Flow With the <code>identify_clock_gating</code> Command	166
Replacing Clock-Gating Cells	166
Inserting Clock Gates With Safety Registers	169
Clock-Gate Optimization Performed During Compilation	171
Hierarchical Clock Gating	171
Enhanced Register-Based Clock Gating	172
Multistage Clock Gating	174
Multistage Clock-Gating Flow	175
Clock Gate Merging	176
Placement-Aware Clock Gating in Design Compiler Graphical	179
Clock Gating Multibit Registers	180
Performing Clock-Gating on DesignWare Components	181
Reporting Clock Gates	181
The <code>report_clock_gating</code> Command	182
<hr/>	
8. Self-Gating	190
Self-Gating Concepts	190
Self-Gating Flows	193
Library Requirements for Self-Gating	194
Inserting Self-Gates	195
Specifying Objects for Self-Gating	195
Specifying Options for Self-Gating	196
Querying and Reporting Self-Gates	196
<hr/>	
9. Power Optimization	200

Contents

Overview	200
Gate-Level Power Optimization	201
Leakage Power Optimization	202
Dynamic Power Optimization	202
High-Effort Power Optimization	203
Enabling Power Optimization	203
Leakage Optimization for Multicorner-Multimode Designs	204
Leakage Power Optimization Based on Threshold Voltage	204
Multiple Threshold Voltage Library Attributes	204
The <code>set_multi_vth_constraint</code> Command	205
Performing Power Optimization	205
Settings for Power Optimization	206
Power Optimization in the Physical Guidance Flow	206
Settings for Low-Power Placement	206
<hr/>	
10. Multivoltage Design Concepts	208
Multivoltage and Multisupply Designs	208
Library Requirements for Multivoltage Designs	209
Liberty PG Pin Syntax	209
Level-Shifter Cells	210
PG Pin Configuration Support	210
Support for NOR-Type Enable Level-Shifter Cells	211
Isolation Cells	211
Using Standard Cells as Isolation Cells	212
Single-Rail and Dual-Rail Isolation Cells	212
NOR-Style Isolation Cells	213
Isolation Cells With Asynchronous Set or Reset Pins	213
Requirements of Level-Shifter and Isolation Cells	215
Retention Register Cells	215
Multithreshold-CMOS Retention Registers	215
Power-Switch Cells	217
Always-On Logic Cells	218
Power Domains	218
Shut-Down Blocks	220
Marking Pass-Gate Library Pins	220
Voltage Areas	220

11. UPF Multivoltage Design Implementation	222
Multivoltage Design Flow Using UPF	223
Power Intent Concepts	226
UPF Script Example	229
Defining Power Intent With UPF Commands	232
Name Spacing Rules for UPF Objects and Attributes	232
Defining the Power Intent in the GUI	233
UPF Diagram View	235
Setting the UPF Command Scope	235
Creating Power Domains	237
Power Domain Boundaries	239
Excluding Elements From Power Domains	240
Representation of Power Domain in the UPF Diagram View	241
Scope	242
Expanding and Collapsing Power Domains in the GUI	243
Viewing Hierarchical Cell and Power Domain Boundaries	244
Creating Atomic Power Domains	245
Examples	246
Reporting Atomic Power Domains	247
Hierarchical Flow Support for Atomic Power Domains	247
Top-Down Hierarchical Flow	247
Bottom-Up Hierarchical Flow	248
Creating Supply Ports	252
Adding Port State Information to Supply Ports	254
Representation of Supply Ports in the UPF Diagram View	254
Creating Supply Nets	255
Creating Custom Resolution Functions	256
Specifying Primary Supply Nets for a Power Domain	256
Representing Supply Nets in the UPF Diagram View	257
Connecting Supply Nets	258
Interpreting PG Connections From the RTL	259
Converting PG Information in the RTL to UPF	259
Preserving Assign Statements on PG Nets	262
Specifying Supply Sets	263
Creating Supply Sets	264
Creating Supply Set Handles	267

Contents

Restricting Supply Sets Available to a Power Domain	268
Refining Supply Sets	268
Associating Supply Sets	270
Rules for Associating Supply Sets	271
Refining Bias Supply Functions Automatically	271
Example 1: No Bias Functions Defined	272
Example 2: Implicit Supply Sets With Resolved Power and Ground	273
Example 3: Implicit Supply Sets With Unresolved Power and Ground	274
Example 4: N-Well Only Support	274
Defining the Power States for a Supply Set	275
Specifying Supply Expressions	276
Specifying Logic Expressions	280
Successive Refinement	283
Correlated Grouping of Supply Voltage Triplets	284
Querying for Supply Sets	285
Limitations	285
Querying for Related Supply Sets	285
Always-On Logic	286
Attributes for Always-On Cells	286
Always-On Optimization	287
Always-On Optimization on Feedthrough Nets	288
Always-On Optimization on Disjoint Voltage Area	289
Always-On Tie Cells	290
Basic Always-On Tie Cell Mapping	290
Enhanced Constant Propagation	291
Enhanced Always-On Tie Cell Mapping	292
UPF Support for Custom Always-On Wrapper Cells	292
leaf_cell_as_domain_boundary Design Attribute	294
upf_control_signal_trace Port Attribute	294
Example	295
Fixing Multivoltage Violations	296
Comparing Voltage Levels and Voltage Status	297
Specifying Level-Shifter Strategies	298
Controlling Level-Shifter Locations	300
Resolving Level Shifter Strategy Precedence	303
Automatically Deriving Level Shifter Strategies for DFT Paths	303
Using Specific Library Cells With the Level-Shifter Strategy	304
Allowing Insertion of Level-Shifters on Clock Nets and Ideal Nets	305
Representing Level-Shifter Strategies in the UPF Diagram View	305

Contents

Specifying Isolation Strategies	308
Isolation Cells With Multiple Control Signals	311
Using the <code>set_isolation_control</code> Command	313
Rules for the Location Fanout Option	314
Order of Precedence of Isolation Strategies	315
Resolving Isolation Strategy Conflicts	315
Automatically Deriving Isolation Strategies for DFT Paths	315
Using Specific Library Cells With Isolation Strategies	316
Aligning Isolation Strategies to Constant Drivers	317
Optimizing Isolation Cell Insertion on Constants	320
Preventing Unnecessary Isolation Cell Insertion	320
Isolation Cells and Heterogeneous Loads	320
Insertion of Isolation Cells on Heterogeneous Fanout Paths	324
Isolation Handling on Control Signals	325
Smart Derivation of <code>-no_isolation</code> Strategy	326
Macro Cells With Internal NOR Isolation Cells	328
Voltage Checking	329
Representing Isolation Strategies in the UPF Diagram View	329
Merging and Cloning Multivoltage Cells	331
Limitations	332
Setting UPF Attributes on Ports and Hierarchical Cells	332
Setting Attributes on Ports	332
Setting Attributes on Macros	334
Setting Design Attributes on Supply Nets and Logic Nets	336
Modeling Unconnected Pins on Macros	336
Specifying Analog Nets	336
Setting Attributes on Hierarchical Cells	337
Setting Terminal Boundaries	339
Querying for UPF Design and Port Attributes	340
Assigning Supplies to Pad Ports	342
Specifying Retention Strategies	343
Specifying Elements to Include in the Retention Strategy	344
Resolving Retention Strategy Precedence	346
Using the Retention Supply as the Primary Supply	347
Choosing Specific Library Cells With Retention Strategies	348
Zero-Pin Retention Support	350
Inferring Complex Retention Cells	350
Retention Strategy and Clock-Gating Cells	352

Contents

Representing Retention Strategies in the UPF Diagram View	352
Specifying Repeater Strategies	353
Specifying Supplies for Repeaters	355
Deferring Element Definitions in Power Management Strategies	357
Matching Tool and IEEE LRM Defaults	357
Creating Power Switches	358
Representation of Power Switches in the UPF Diagram View	359
Power Models	361
Configuring Power Compiler for Power Models	361
Defining and Applying a Power Model	362
Excluding Designs From Using Power Models	362
Hard and Soft Macros	362
Power State Tables	363
Default Power States	363
Power State Propagation	364
Creating Power State Tables	365
Hierarchical Power State Tables	366
Creating Power State Groups in Hierarchies Having State Propagation Enabled	368
Example	368
Bottom-Up Hierarchical Flow	369
Top-Down Hierarchical Flow	371
Reconciling Voltages in Power State Tables	375
Reporting Power State Tables	378
Visually Analyzing Power State Tables in the UPF Diagram View	379
Support for Well Bias	381
Using a Non-Bias Block in a Bias-Enabled Design	382
Skipping Bias Checks	383
Inserting Power Management Cells	383
Relaxing PVT Library Constraints for Power Management Cells	385
Reviewing the UPF Specifications	385
Commands to Query and Edit Design Objects	386
Reviewing the Power Intent Using the Design Vision GUI	387
Applying Power Intent Changes	392
Examining and Debugging UPF Specifications	392
The analyze_mv_feasibility Command	393
Reporting Resolved Strategies	393

Contents

Reporting Cell Mapping Feasibility	396
Generating HTML Cell Mapping Reports	400
The check_mv_design Command	401
Multivoltage Design Violations in the GUI	402
Generating Design Violation Reports	403
Examining Design Violations in the MV Advisor Violation Browser	405
Exploring the Violations	407
The analyze_mv_design Command	410
Analyzing Multivoltage Design Connections in the GUI	411
Writing the Power Information	413
Preserving the Command Order in the UPF' File	414
Controlling the Line Width in the UPF' File	415
Writing and Reading Verilog Netlists With Power and Ground Information	416
Power and Ground Supply Connection Syntax	417
Supply Sets	418
Power Switches	418
Reading Verilog Netlists With Power and Ground Supply Connections	419
Specifying Design Instances Using SystemVerilog Elements	419
The Golden UPF Flow	420
Reporting Commands for the UPF Flow	422
UPF-Based Hierarchical Multivoltage Flow Methodology	422
Hierarchical UPF Design Methodology	423
Block-Level Implementation	423
Top-Level Implementation	427
Assembling the Design	428
Characterization of Supply Sets and Supply Nets	428
Automatic Inference of Related Supply Nets	429
Top-Level Design Integration	432
Power Domain Merging	432
Legacy Blocks	433
<hr/>	
12. Library Setup for Power Optimization	436
Basic Library Requirements for Multivoltage Designs	436
Power and Ground Pin Syntax	436
Converting Libraries to PG Pin Library Format	436
Using the FRAM View	437
Using Tcl Commands	437
Tcl Commands for Low-Power Library Specification	439

Contents

Macro Cells With Fine-Grained Switches	439
Library Usage in Multicorner-Multimode Designs	440
Link Libraries With Equal Nominal PVT Values	440
Setting the <code>dont_use</code> Attribute on Library Cells	442
Distinct PVT Requirements	442
Automatic Detection of Driving Cell Library	444
Relating the Minimum Library to the Maximum Library	444
Unique Identification of Libraries Based on File Names	445
Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells	446
Using the <code>set_opcond_inference</code> Command	446
Deviating From the Inferred Operating Condition and Its Impact	446
13. Power Optimization in Multicorner-Multimode Designs	450
Optimizing Multicorner-Multimode Designs	450
Optimizing for Leakage Power	450
Optimizing for Dynamic Power Using Low-Power Placement	453
Reporting Commands	453
<code>report_scenarios</code> Command	453
Reporting Examples for Multicorner-Multimode Designs	454
Script Example for Multicorner-Multimode Flow	456

Appendices

A. Lower-Domain Boundary Support	461
Overview of Power Domain Boundaries	461
Applying Isolation and Level-Shifter Strategies	463
Specifying Domain Boundaries With the <code>-applies_to</code> Option	464
Defining Cell Placement With the <code>-location</code> Option	464
B. Integrated Clock-Gating Cell Example	467
Library Description	467
Example Schematics	470
Rising-Edge Latch-Based Integrated Cells	470
Rising-Edge Latch-Free Integrated Cells	472

Contents

Falling Edge Latch-Based Integrated Cells	473
Falling-Edge Latch-Free Integrated Cells	475

C. Attributes for Querying and Filtering	477
Derived Attribute Lists	477
Usage Examples	479

About This User Guide

This user guide describes the Power Compiler tool, its methodology, and its use. The Power Compiler tool assists you in analysis and optimization of your design for power.

The *Power Compiler User Guide* builds on concepts introduced in Design Compiler publications. It is assumed in this user guide that the user has some familiarity with Design Compiler products.

This preface includes the following sections:

- [New in This Release](#)
 - [Related Products, Publications, and Trademarks](#)
 - [Conventions](#)
 - [Customer Support](#)
-

New in This Release

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

Related Products, Publications, and Trademarks

For additional information about the Power Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

<https://solvnetplus.synopsys.com>

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- Design Vision™
- DesignWare® components
- TestMAX™ DFT
- IC Compiler™ II
- Library Compiler™

- Formality®
- PrimePower

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code>
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
Purple	<ul style="list-style-type: none"> • Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as <code>include_enclosing = true false</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code> .
	Indicates a choice among alternatives, such as <code>low medium high</code>
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

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

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

<https://solvnetplus.synopsys.com>

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

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

Contacting Customer Support

To contact Customer Support, go to <https://solvnetplus.synopsys.com>.

Part 1: Power Compiler Concepts

The following topics provide an introduction to the Power Compiler tool:

- [Introduction to the Power Compiler Tool](#)
- [Power Compiler Design Flow](#)
- [Power Modeling and Calculations](#)

1

Introduction to the Power Compiler Tool

The Power Compiler tool is part of the Synopsys Design Compiler synthesis family. The tool performs both RTL and gate-level power optimization and gate-level power analysis. By applying the tool's power reduction techniques, including clock-gating, multivoltage optimization, and leakage power optimization, you can achieve power savings and area and timing optimization in front-end synthesis.

For information about the tool's methodology, see the following topics:

- [Power Compiler Methodology](#)
 - [Power Library Models](#)
 - [Power Analysis](#)
 - [Power Optimization](#)
 - [Getting Started With the Power Compiler Tool](#)
-

Power Compiler Methodology

Low-power designs have become crucial elements for product success. For example, the static power (leakage power) consumption is more critical as the technology becomes more smaller and faster.

The Power Compiler tool provides a complete methodology for low-power design, which consist of the following:

- Power analysis

The tool analyzes the design and computes average power consumption based on the switching activity of the nets.

You can perform power analysis at the register transfer level using RTL simulation or at the gate level using RTL or gate-level simulation.

- Power optimization

The tool optimizes the design for power consumption. It computes average power consumption based on the activity of the nets in the design.

The tool performs the following types of power optimization:

- Leakage power or static power optimization
 - Multivoltage threshold power optimization
 - Power switching
- Dynamic power optimization
 - Clock gating
 - Self-gating
 - Low-power placement
- Multivoltage and multicorner-multimode support

Power Library Models

The power library model analyzes leakage, switching, and internal power. For more information about library modeling and characterization for power, see the Library Compiler documentation.

The Power Compiler gate-level power model supports the following features:

- Composite Current Source (CCS) library support
- Lookup tables based on output pin capacitance and input transition time
- Cells with multiple output pins
- State-dependent and path-dependent internal power
- Leakage power, including state-dependent and path-dependent internal power
- Separate specification of rise and fall power in the internal power group

In addition, you can use CCS power models, which represent the physical circuit properties more closely than other models to the simulated data obtained during characterization with SPICE. It is a current-based power model that contains the following features:

- One library format suitable for a wide range of applications, including power analysis and optimization
- Power analysis with much higher time resolution compared to NLPM models
- Dynamic power characterized by current waveforms stored in the library. The charge can be derived from the current waveform.

- Leakage power modeled as the actual leakage current. The leakage current does not artificially depend on the reference voltage, as is the case with leakage power. This facilitates voltage scaling.
 - Standard-cell and macro modeling
-

Power Analysis

The Power Compiler tool analyzes a design for net switching power, cell internal power, and leakage power. The tool also enables you to perform power analysis of a gate-level design using switching activity from RTL, gate-level simulation, or user annotation.

When analyzing a gate-level design, the Power Compiler tool requires a gate-level netlist and switching activity for the netlist. The tool enables you to capture the switching activity of primary inputs, primary outputs, and outputs of sequential elements during RTL simulation. After you annotate the captured activity on design elements, the tool propagates switching activity through the nonannotated portions of your design.

Using power analysis with switching activity from RTL simulation provides a faster turnaround than analysis using switching activity from gate-level simulation.

If you require more accuracy during the later stages of design development, you can annotate some or all of the nets of your design with switching activity from full-timing gate-level simulation.

The Power Compiler tool supports the following power analysis features:

- Performs gate-level power analysis
- Analyzes net switching power, cell internal power, and leakage power
- Accepts input as user-defined switching activity, switching activity from RTL or gate-level simulation, or a combination
- Propagates switching activity during power analysis to nonannotated nets
- Supports sequential, hierarchical, gated clock, and multiple-clock designs
- Supports RAM and I/O modeling using a detailed state-dependent and path-dependent power model
- Performs power analysis in a single, integrated environment at multiple phases of the design process
- Reports power at any level of hierarchy to enable quick debugging

- Reports capability to validate your testbench
- Supports interfaces to NC-Sim, MTI, VCS-MX, and Verilog-XL simulators for toggle data

Power Optimization

You can optimize your design for power using the following capabilities:

- RTL clock gating
- Gate-level multivoltage and dynamic power optimization

RTL clock gating is a high-level optimization technique that can save a significant amount of power by adding clock gates to registers that are not always enabled and with synchronous load-enable or explicit feedback loops. This greatly reduces the power consumption of the design by reducing switching activity on the clock inputs to registers and eliminating the multiplexers. It also results in a lower area consumption. RTL clock gating optimizes for dynamic power.

When a gate-level power optimization constraint is set in the design, by default, the tool performs optimization to meet the constraints for design rule checking, timing, power and area in that order of priority.

The tool provides the following gate-level power optimization features:

- Push-button user interface to reduce power consumption
- Multivoltage libraries for leakage optimization with short turnaround time
- Simultaneous optimization for timing, power, and area
- Optimization based on circuit activity, capacitance, and transition times
- Power analysis and optimization with the same detailed power library models
- Compatibility with the Synopsys Design Compiler, TestMAX DFT, and Formality tools

Getting Started With the Power Compiler Tool

This section provides information about the basic requirements to use the Power Compiler tool.

Library Requirements

The Power Compiler tool uses logic libraries characterized for power. You can characterize your library with the following power features:

Internal Power

To optimize for dynamic power, the Power Compiler tool requires libraries characterized for internal power. This is the minimum library requirement to characterize for power. This characteristic accounts for short-circuit power consumed internal to gates.

Leakage Power

To optimize for static power, the tool requires libraries characterized for leakage power. This characteristic accounts for the power dissipated while the device is not in use. The tool also supports multivoltage libraries.

State and Path Dependency

To optimize for varying modes of operation, the Power Compiler tool requires libraries characterized for state-dependency. To optimize for varying power consumption based on various input to output paths, the tool requires libraries characterized for path-dependency.

To capture state-dependent and path-dependent switching activity from simulation, library cells must have state- and path-dependent information in the lookup tables for internal power and pin capacitance. The Power Compiler tool uses state-dependent and path-dependent switching activity to compute state-dependent and path-dependent switching power.

If you are developing libraries to use with Synopsys power products, see the Library Compiler documentation. The Power Compiler tool supports nonlinear power models, scalable polynomial equation power models, and composite current source libraries.

Command-Line Interface

The Power Compiler tool is accessible from the Design Compiler command-line interface if you have an appropriate license. See [License Requirements](#).

Using the Design Compiler command-line interface, power optimization takes place during your dc_shell optimization session. For more information about its command-line interface, see the Design Compiler documentation.

The Power Compiler tool also works within the Design Compiler topographical domain shell (dc_shell-topo). Whereas dc_shell uses wide-load models for timing and area power optimizations, dc_shell-topo uses placement timing values instead. For more information, see the Design Compiler documentation.

Note:

Unless otherwise noted, all functionality described in this manual pertains to both dc_shell and dc_shell-topo. Unless otherwise noted, this manual uses "dc_shell" as a generic term that also applies to the Design Compiler topographical domain.

Graphical User Interface

The Power Compiler tool is accessible from Design Vision, the graphical user interface (GUI) for the Synopsys logic synthesis environment. You must have the Design Vision license and other appropriate licenses to perform power analysis and optimizations. For more details, see [License Requirements](#).

Design Vision supports menus and dialog boxes for the frequently used synthesis features. The Power menu in the GUI allows you to specify, modify, and review your power architecture.

For more details on specifying power intent using the GUI, see [Chapter 11, UPF Multivoltage Design Implementation](#). For details about general usage of Design Vision, see the *Design Vision User Guide*.

License Requirements

Power analysis and optimization using the Power Compiler tool requires one of the following combinations of licenses:

- Power-Optimization
- Power-Analysis + Power-Optimization-Upgrade

These licenses also allow you to perform multivoltage power optimization and analysis.

The Power Compiler tool is part of the Design Compiler tool. You must have licenses for the Design Compiler tool in addition to the power licenses.

Design Vision License

You can also perform power analysis and power optimization using the Design Vision GUI. To use Design Vision, you must have the Design-Vision license. To use Design Vision in topographical mode, you must have a Design-Vision license, a DesignWare license, and the DC Ultra package.

How the Licenses Work

When you invoke dc_shell, a power license is checked out only if you use a Power Compiler feature. When the activity is completed, the power license is released.

Synopsys licensing software and the documentation describing it are separate from the tools that use it. You install, configure, and use a single copy of Synopsys Common Licensing (SCL), which provides a single, common licensing base for all Synopsys tools.

For more information, see the *Synopsys Common Licensing Administration Guide*. This guide provides detailed information about SCL installation and configuration, including examples of license key files and troubleshooting guidelines.

Reading and Writing Designs

When using dc_shell, you read designs from disk before working on them, make changes to them, and write them back to the disk.

The tool can read or write a gate-level netlist in any of the formats shown in [Table 1](#).

Table 1 File Formats and Extensions

Format	Default extension	File type	Special license key required
db	.db	Synopsys internal database format	No
ddc	.ddc	Synopsys Design Compiler database format (the default)	No
equation	.eqn	Synopsys equation format	No
LSI	.NET	LSI Logic Corporation netlist format	Yes
MENTOR	.neted	Mentor Graphics® intermediate netlist format	Yes
PLA	.pla	Berkeley (Espresso) PLA format	No
ST	.st	Synopsys state table format	No
TDL	.tdl	Tegas Design Language (TDL) netlist format	Yes
Verilog	.v	Hardware Description Language	Yes
VHDL	.vhd	VHSIC Hardware Description	Yes

Note:

NLPM and CCS are the supported power models in the library.

2

Power Compiler Design Flow

As you create a design, it moves from a high level of abstraction to its final implementation at the gate level. The Power Compiler tool offers analysis and optimization throughout the design cycle, from RTL to the gate level.

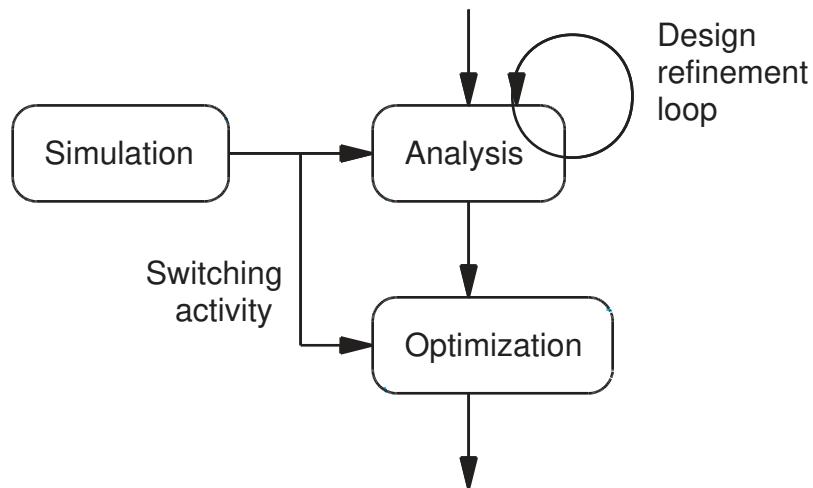
For information about the design flow, see the following topics:

- [Power in the Design Cycle](#)
 - [Power Optimization and Analysis Flow](#)
-

Power in the Design Cycle

At each level of abstraction, use simulation, analysis, and optimization to refine your design before moving to the next lower level of design abstraction. The relationship of these three processes is shown in [Figure 1](#).

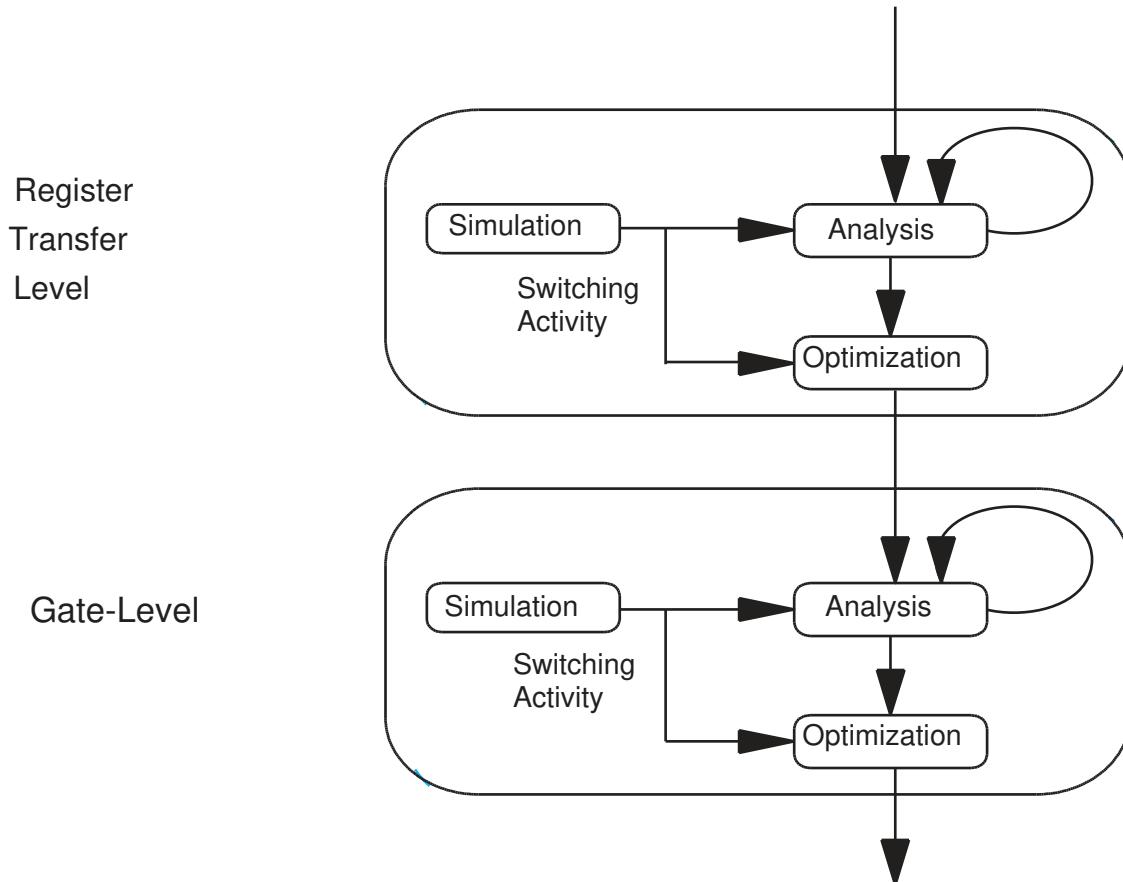
Figure 1 Power Flow at Each Abstraction Level



Simulation, analysis, and optimization occur at each level of abstraction. Design refinement loops occur within each level. Simulation and the resultant switching activity

give analysis and optimization the necessary information to refine the design before going to the next lower level of abstraction. The entire flow is shown in [Figure 2](#).

Figure 2 Power Flow From RTL to Gate-Level

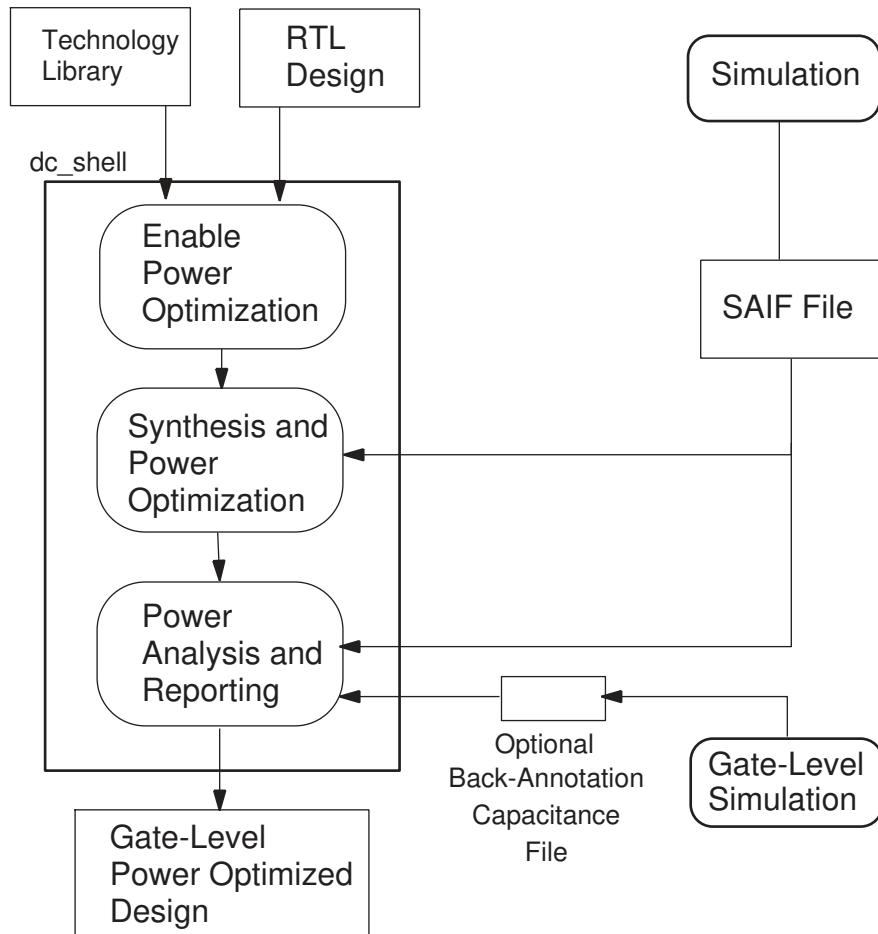


Using the Power Compiler tool, you can analyze and optimize at the RTL and gate levels. The higher the level of design abstraction, the greater the power savings you can achieve.

Power Optimization and Analysis Flow

[Figure 3](#) shows a high-level power optimization and analysis flow.

Figure 3 Power Optimization and Analysis Flow



The power optimization starts with the specified RTL design and logic library and results in a power-optimized gate-level netlist.

During analysis and optimization, the Power Compiler tool uses information in the logic library. To optimize or analyze dynamic power and leakage power, the logic library must be characterized for internal power. To optimize or analyze static power, the logic library must be characterized for leakage power.

You can use the Power Compiler tool to analyze the gate-level netlist produced by the Design Compiler tool or the power-optimized netlist produced by the Power Compiler tool.

Simulation

Most of the steps in the flow occur within the Design Compiler environment, dc_shell. However, [Figure 3](#) shows that the power flow requires a SAIF file, which is generated by simulation.

Simulation generates information about the design's switching activity and creates a Switching Activity Information Format (SAIF) file, which is used for annotation purposes. For information, see [Chapter 4, Generating SAIF Files](#).

During power analysis, the Power Compiler tool uses annotated switching activity to evaluate the power consumption of your design. During power optimization, the tool uses annotated switching activity to make optimization decisions about your design. For more information, see [Chapter 5, Annotating Switching Activity](#).

Enable Power Optimization

The Power Compiler tool provides several techniques for optimizing power, such as clock gating and self-gating. Power optimization achieved at higher levels of abstraction has an increasingly important impact on reduction of power in the final gate-level implementation.

Synthesis and Power Optimization

The Design Compiler and Power Compiler tools work together within the dc_shell environment to synthesize your design to a gate-level netlist optimized for power. Synthesis with power optimization occurs during the Design Compiler compile operation.

In the Synopsys physical guidance flow, the tool can perform low-power placement to reduce the dynamic power consumption of the design. For more details, see [Power Optimization in the Physical Guidance Flow](#).

Power Analysis and Reporting

You can use the Power Compiler tool for analysis of a gate-level design at several points in the flow. [Figure 3](#) shows power analysis after power optimization, which results in a detailed report of your power-optimized netlist.

You can also analyze power before synthesis and power optimization, for example, after annotating the switching activity from a SAIF file to verify that the annotation is correct. Analysis before power optimization provides an optional reference point for comparison with the power-optimized netlist.

3

Power Modeling and Calculations

The Power Compiler tool can analyze both the static and dynamic power consumption of a design.

For information about power modeling, see the following topics:

- [Power Types](#)
 - [Calculating Power](#)
 - [Using CCS Power Libraries](#)
 - [Voltage Scaling](#)
-

Power Types

The power dissipated in a circuit falls into two broad categories:

- Static power
 - Dynamic power
-

Static Power

Static power is the power dissipated by a gate when it is not switching, that is, when it is inactive or static.

Static power is dissipated in several ways. The largest percentage of static power results from source-to-drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called leakage power.

Dynamic Power

Dynamic power is the power dissipated when the circuit is active. A circuit is active anytime the voltage on a net changes due to some stimulus applied to the circuit. Because voltage on an input net can change without necessarily resulting in a logic transition on the

output, dynamic power can be dissipated even when an output net does not change its logic state.

The dynamic power of a circuit is composed of two kinds of power:

- Switching power
- Internal power

Switching Power

The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output.

Because such charging and discharging are the result of the logic transitions at the output of the cell, switching power increases as logic transitions increase. Therefore, the switching power of a cell is a function of both the total load capacitance at the cell output and the rate of logic transitions.

Internal Power

Internal power is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.

To illustrate the cause of short-circuit power, consider the simple gate shown in [Figure 4](#). A rising signal is applied at IN. As the signal transitions from low to high, the N type transistor turns on and the P type transistor turns off. However, for a short time during signal transition, both the P and N type transistors can be on simultaneously. During this time, current I_{sc} flows from V_{dd} to GND, causing the dissipation of short-circuit power (P_{sc}).

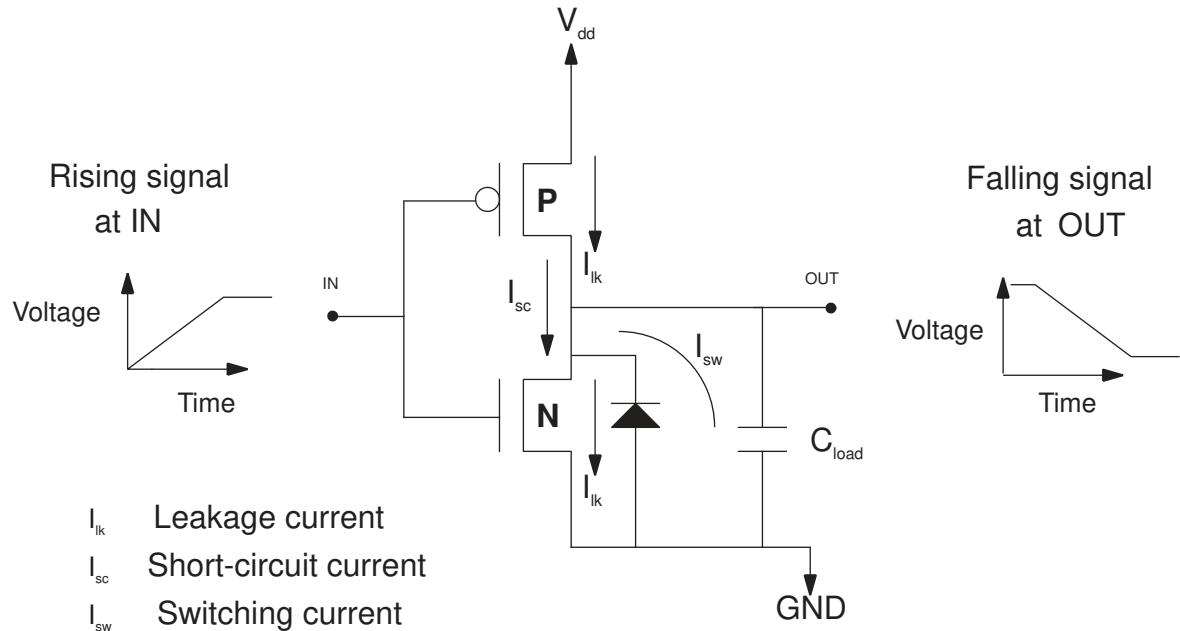
For circuits with fast transition times, short-circuit power can be small. However, for circuits with slow transition times, short-circuit power can account for 30 percent of the total power dissipated by the gate. Short-circuit power is affected by the dimensions of the transistors and the load capacitance at the gate's output.

In most simple library cells, internal power is due mostly to short-circuit power. For more complex cells, the charging and discharging of internal capacitance might be the dominant source of internal power.

Library developers can model internal power by using the internal power library group. For more information about modeling internal power, see the *Library Compiler User Guide*.

[Figure 4](#) shows a simple gate and illustrates where static and dynamic power are dissipated.

Figure 4 Components of Power Dissipation



Calculating Power

Power analysis calculates and reports power based on the equations that accompany this chapter. The Power Compiler tool uses these equations and the information modeled in the specified logic library to evaluate the power of your design. This chapter includes information about library modeling for power where equations for power types appear.

For more information about modeling power in your library, see the *Library Compiler User Guide*.

Note:

The power calculations described in this section only apply to NLPM power calculations.

Leakage Power Calculation

The Power Compiler tool computes the total leakage power of a design by summing the leakage power of the design's library cells, as shown in the following equation:

$$P_{LeakageTotal} = \sum_{\forall cells(i)} P_{CellLeakage_i}$$

Where:

$P_{\text{LeakageTotal}}$ = Total leakage power dissipation of the design

$P_{\text{CellLeakage}_i}$ = Leakage power dissipation of each cell i

Library developers annotate the library cells with appropriate total leakage power dissipated by each library cell. They can provide a single leakage power for all cells in the library by using the `default_cell_leakage_power` attribute or provide leakage power per cell with the `cell_leakage_power` attribute.

If the `cell_leakage_power` attribute is missing or negative, the tool assigns the value of the `default_cell_leakage_power` attribute. If this is not available, the tool assumes a default of 0.

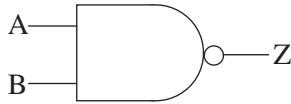
To model state-dependent leakage, use the `leakage_power` attribute. You can also use Boolean expressions to define the conditions for different cell leakage power values.

To calculate cell leakage, the Power Compiler tool determines the units based on the `leakage_power_unit` attribute. It checks for the `leakage_power` attribute first. The leakage value for each state is multiplied by the percentage of the total simulation time at that state and summed to provide the total leakage power per cell.

If the state is not defined in the `leakage_power` attribute, the value of the `cell_leakage_power` attribute is used to obtain the contribution of the leakage power at the undefined state.

[Figure 5](#) shows the leakage power calculation performed on a NAND gate with state-dependent values.

Figure 5 Leakage Power Calculation for a NAND Gate With State Dependent Values



```

library ....
leakage_power_unit : 1nW ;
cell (NAND) ...
cell_leakage_power : 0.5 ;
leakage_power () {
    when : "A&B"
    value : 0.2
  
```

For a total power consumption time of 600, the cell is at the state defined by the condition A&B for 33% of the time. For the remaining 67% of the simulation time, the default is assumed.

Therefore, the total cell leakage value is:

$$(.33 * .2\text{nW}) + (.67 * .5\text{nW}) = .4\text{nW}$$

Multithreshold Voltage Libraries

Static power dissipation has an exponential dependence on the switching threshold of the transistor's voltage. In order to address low-power designs IC foundries offer technologies that enable multiple threshold voltage libraries.

Each type of logic gate is available in two or more different threshold voltage (v_{th}) groups. The threshold voltage determines the speed and the leakage characteristics of the cell. Cells with low-threshold transistors switch quickly but have higher leakage and consume more power. Cells with high threshold transistors have lower leakage and consume less power but switch more slowly.

For leakage power optimization, the Power Compiler tool supports multiple mechanisms for swapping high and low-threshold voltage cells appropriately, based on power and timing requirements.

For more details about using the multithreshold voltage libraries, see [Multiple Threshold Voltage Library Attributes](#).

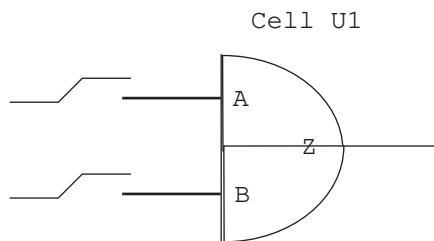
Internal Power Calculation

When computing internal power, power analysis uses information characterized in the logic library. The `internal_power` library group and its associated attributes and groups

define scaling factors and a default for internal power. Library developers can use the internal power table to model internal power on any pin of the library cell.

A cell's internal power is the sum of the internal power of all of the cell's inputs and outputs as modeled in the logic library. [Figure 6](#) shows how Synopsys power tools calculate the internal power for a simple combinational cell, U1 with path-dependent internal power modeling.

Figure 6 Internal Power Model (Combinational)



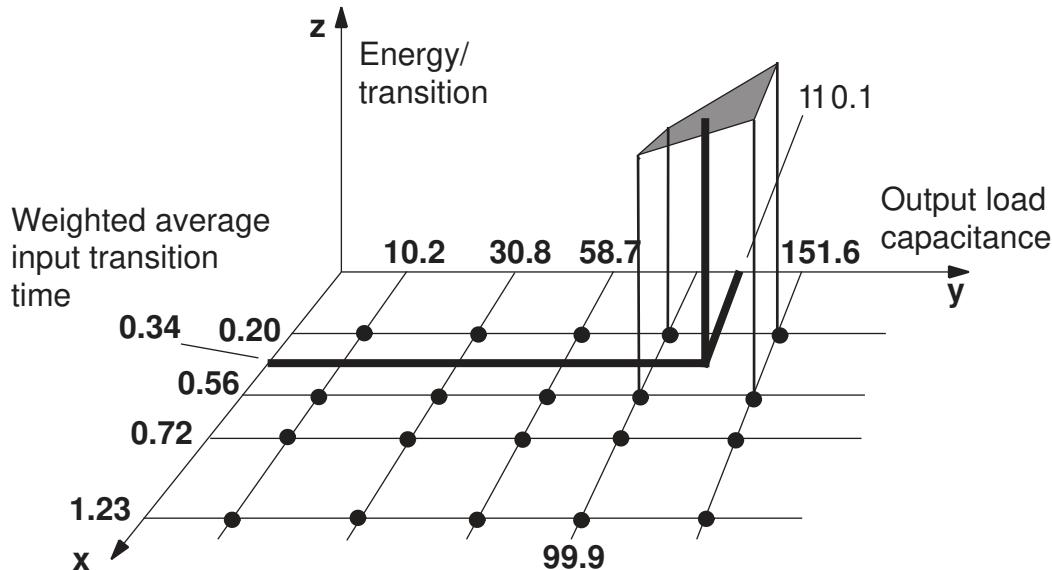
$$\begin{aligned} P_{\text{Int}} &= \sum_{\{i=A,B\}} E_{\{i \rightarrow Z\}} \times \text{PathWeight}_i \times TR_z \\ E_{\{i \rightarrow Z\}} &= f[C_Load, \text{Trans}_i] \\ \sum_{\{i = A,B\}} \text{PathWeight}_i &= 1 \end{aligned}$$

P_{Int}	Total internal power of the cell E
E_z	Internal energy for output Z as a function of input transitions, output load, and voltage
TR_z	Toggle rate of output pin Z, transitions per second
TR_i	Toggle rate of input pin i, transitions per second
Trans_i	Transition time of input i
$\text{WeightAvg}_{(\text{Trans})}$	Weighted average transition time for output Z

The Power Compiler tool calculates the input path weights based on the input toggle rates, transition times, and functionality of the cell. The tool supports NLDM (table-based) models.

NLDM Models

To compute the internal power consumption of NLDM models, the Power Compiler tool uses the weighted average transition time as an index to the internal power associated with the output pin. As an additional index to the power table, the tool uses the output load capacitance. The two indexes enable access to the two-dimensional lookup table for the output, as shown in [Figure 7](#).

Figure 7 Two-Dimensional Lookup Table


For cells in which output pins have equal or opposite logic values, the tool can use a three-dimensional lookup table. The tool indexes the three-dimensional table by using input transition time and both output capacitances of the equal (or opposite) pins. The three-dimensional table is well suited to describing the flip-flop, which has Q and Q-bar outputs of opposite value.

The `internal_power` library group supports one-, two-, or three-dimensional lookup tables. [Table 2](#) shows the types of lookup tables, whether they are appropriate to inputs or outputs, and how they are indexed.

Table 2 *Lookup Tables*

Lookup table	Defined on	Indexed by
One-dimensional	Input	Input transition
	Output	Output load capacitance
Two-dimensional	Output	Input transition and output load capacitance
Three-dimensional	Output	Input transition and output load capacitances of two outputs that have equal or opposite logic values

For more information about modeling internal power and library modeling syntax and methodology, see the *Library Compiler User Guide*.

For various operating conditions, the table model supports scaling factors for the internal power calculation, as follows:

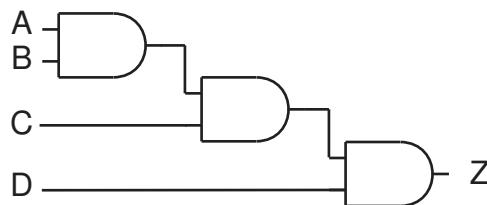
- `k_process_internal_power`
- `k_temp_internal_power`
- `k_volt_internal_power`

These factors do not accurately model the nonlinear effects of the operating conditions. Therefore, most vendors generate separate table-based libraries for different operating conditions.

State and Path Dependency

Cells often consume different amounts of internal power, depending on which input pin transitions or depending on the state of the cell. These are state and path dependent.

To demonstrate path-dependent internal power, consider the following simple library cell, which has several levels of logic and a number of input pins:



Input A and input D can each cause an output transition at Z. However, input D affects only one level of logic, whereas input A affects all three. An output transition at Z consumes more internal power when it results from an input transition at A than when it results from an input transition at D. You can specify multiple lookup tables for outputs, depending on the input transitions.

The Power Compiler tool chooses the appropriate path dependent internal power table for an output by checking the `related_pin` attribute in the library. Based on the percentage of toggles on each input pin, the total power due to transitions on the output pin is calculated by accessing the correct table or equation for each related pin and applying the percentage contribution per input pin.

An example of a cell with state-dependent internal power is a RAM cell. It consumes a different amount of internal power depending on whether it is in read or write mode. You can specify separate tables or equations depending on the state or mode of the cell.

If the toggle rate information is provided for each state defined in the power model, the tool accesses the appropriate information. If only the input or output toggle information is available, the tool averages the tables for the different states to compute the internal power of the cell.

For more information about how the toggle information affects the internal power analysis, see [Performing Power Analysis](#).

Rise and Fall Power

When a signal transitions, the internal power related to the rising transition is different from the internal power related to the falling transition. The Power Compiler tool supports a library model that enables you to designate a separate rising and falling power value, depending on the transition.

Switching Power Calculation

The Power Compiler tool calculates switching power (P_c) as follows:

$$P_c = \frac{V_{dd}^2}{2} \sum_{\forall nets(i)} (C_{Load_i} \times TR_i)$$

Where:

P_c Switching power of the design

TR_i Toggle rate of net i, transitions per second

V_{dd} Supply voltage

C_{Load_i} is the total capacitive load of net i, including parasitic capacitance, gate capacitance, and drain capacitance of all the pins connected to the net i.

The tool obtains C_{Load_i} from the wire load model for the net and from the logic library information for the gates connected to the net. You can also back-annotate capacitance information after physical design.

Dynamic Power Calculation

Because dynamic power is the power dissipated when a circuit is active, the equations for switching power and internal power provide the dynamic power of the design.

Dynamic power = Switching power + Internal power

For more information about the library models, see the *Library Compiler User Guide*.

Dynamic Power Unit Derivation

The unit for switching power and the values in the `internal_power` table is a derived unit. It is derived from the following function:

$$(\text{capacitive_load_unit} * \text{voltage_unit}^2) / \text{time_unit}$$

The function's parameters are defined in the library. The result is scaled to the closest MKS unit: micro, nano, femto, or pico. This dynamic power unit scaling effect needs to be taken into account by library developers when generating energy values for the internal power table.

The following is an example of how the Power Compiler tool derives dynamic power units (if the library has the following attributes):

```
capacitive_load_unit (0.35, ff);
voltage_unit: "1V"
time_unit: "1ns";
```

To obtain the dynamic power unit, complete the following steps:

1. Find the starting value.

```
starting value = capacitive_load_unit*voltage_unit2/
time_unit
starting value = .35e-15*(1^2)/1e-9
starting value = 3.5e-7W
```

The starting value consists of a base unit (1e-7W) and a multiplier (3.5).

2. Select an MKS base unit that converts the multiplier of the starting value found in step 1 to an integer number. For example, select an MKS unit between the range of att [1e-18] and giga [1e+12] watts, which converts the starting value's multiplier into an integer value.

The MKS base unit that meets this requirement in this example is nano [1e-9]. This is because the starting value of 3.5e-7W expressed in nW becomes 350nW. The original multiplier of 3.5 is converted to an integer value (350) by selecting the nW MKS base unit.

```
converted value = 350e-9W
converted value multiplier = 350
base unit = 1e9W = 1nW
```

3. Determine the base unit multiplier by selecting a power of 10 integer (for example, 1, 10, 100, ...) closest in magnitude to the converted value multiplier found in step 2.

```
converted value multiplier = 350 (from step 2)
base unit multiplier = 100
```

4. Combine the base unit multiplier obtained in step 3 and the base unit obtained in step 2 to obtain the dynamic power unit.

```
base unit = 1nW (from step 2)
base unit multiplier = 100 (from step 3)
dynamic power unit = (100) 1nW = 100nW
```

In this example, each cell's dynamic power calculated by the tool is multiplied by 100nW.

Power Calculation for Multirail Cells

The Power Compiler tool supports the power analysis of libraries which contain cells with multiple rails for which power values are defined per voltage rail.

For multivoltage cells which contain separate power tables for each power level, the tool determines the internal and leakage power contribution for each power rail and sums it to report the total power consumption.

For more information about defining per-rail power tables, see the *Library Compiler User Guide*.

The following example shows example cells that contain power tables per rail.

```
cell (AND2_1) {
    area : 1.0000;
    cell_footprint : MV12AND2;
    rail_connection (PV1, VDD1);
    rail_connection (PV2, VDD2);

    pin (a) {
        direction : input;
        capacitance : 0.1;
        input_signal_level : VDD1;
        internal_power () {
            power_level : VDD1;
            power(scalar) { values ( "1.0" ); }
        }
    }
    pin (b) {
        direction : input;
        capacitance : 0.1;
        input_signal_level : VDD1;
        internal_power () {
            power_level : VDD1;
            power(scalar) { values ( "1.0" ); }
        }
    }
    pin (y) {
        direction : output;
        function : "a & b";
        output_signal_level : VDD2;
    }
}
```

```

timing () {
    related_pin : "a";
    timing_sense : positive_unate;
    cell_rise      ( scalar ) { values ( "1.0" ); }
    rise_transition ( scalar ) { values ( "1.0" ); }
    cell_fall      ( scalar ) { values ( "1.0" ); }
    fall_transition ( scalar ) { values ( "1.0" ); }
}
timing () {
    related_pin : "b";
    cell_rise      ( scalar ) { values ( "1.0" ); }
    rise_transition ( scalar ) { values ( "1.0" ); }
    cell_fall      ( scalar ) { values ( "1.0" ); }
    fall_transition ( scalar ) { values ( "1.0" ); }
}
internal_power () {
    power_level : VDD1;
    power(scalar) { values ( "1.0" ); }
}
internal_power () {
    power_level : VDD2;
    power(scalar) { values ( "2.0" ); }
}
leakage_power () {
    power_level : VDD1;
    value : 1.0;
}
leakage_power () {
    power_level : VDD2;
    value : 2.0;
}
cell_leakage_power : 10;
}

```

Using CCS Power Libraries

CCS power libraries contain unified library data for power and rail analysis and optimization, which ensures consistent analysis and simplification of the analysis flow. By capturing current waveforms in the library, you can provide more accurate identification of potential problem areas.

Both CCS and NLPM data can coexist in a cell description in the .lib file. That is, a cell description can have only NLPM data, only CCS data, or both NLPM and CCS data. The tool uses either NLPM data or CCS data for the power calculation.

Use the `power_model_preference nlp | ccs` variable to specify your power model preference when the library contains both NLPM and CCS in it. The default is `nlp`. Using CCS or NLPM power libraries does not change the use model.

For more information about CCS power libraries and how to generate them, see the *Library Compiler User Guide*.

Voltage Scaling

The Power Compiler tool uses the scaling library groups to implement temperature and voltage scaling. For voltage scaling, the libraries in the scaling group must contain CCS and NLPM power models.

To enable the scaling feature and specify the membership of libraries to scaling library groups, use the `define_scaling_lib_group` command. You can specify different scaling library groups for different design objects or subdesigns by using the `set_scaling_lib_group` command. To create an intermediate operating condition, use the `create_operating_conditions` command. Use the `set_operating_conditions` command to set intermediate voltage or temperature conditions. With the specified operating conditions, the tool performs interpolation between the libraries in the library groups to obtain accurate delay information.

For more information about defining and setting the scaling library groups, see the related command man pages.

To perform both voltage and temperature scaling at the same time, use four libraries in the scaling group rather than two, representing the four possible combinations of voltage and temperature extremes: high voltage and high temperature, high voltage and low temperature, low voltage and high temperature, and low voltage and low temperature.

The tool supports power scaling on both single-rail and multirail cells.

Script Examples for Voltage Scaling

[Example 1](#) shows an example script to perform voltage scaling in a multivoltage design.

Example 1 Voltage Scaling in Multivoltage Designs

```
read_verilog rtl.v
current_design top
link
define_scaling_lib_group -name group1{slow_0p81v.db slow_1p2v.db}
set_scaling_lib_group -min group1 -max group1
create_operating_conditions -name scaled_pvt -library slow_0p81.v \
    -process 1 -voltage 1.0 -temperature -40
set_operating_conditions -help
```

[Example 2](#) shows an example script to perform voltage scaling in a multicorner-multimode design.

Example 2 Voltage Scaling in Multicorner-Multimode Designs

```
read_verilog rtl.v
current_design top
link
define_scaling_lib_group -name group1{slow_0p81v.db slow_1p2v.db}
set_scaling_lib_group -min group1 -max group1
create_operating_conditions -name scaled_pvt -library slow_0p81.v \
    -process 1 -voltage 1.0 -temperature -40

create_scenario s1
read_sdc s1.sdc
set_operating_conditions scaled_pvt
set_tlu_plus_files -max_tluplus tlu_file1 -tech2itf_map map_file1
set_scaling_lib_group {group1}
```

Part 2: Power Analysis

The following topics provide information about the power analysis features of the Power Compiler tool:

- [Generating SAIF Files](#)
- [Annotating Switching Activity](#)
- [Performing Power Analysis](#)

4

Generating SAIF Files

The Power Compiler tool requires information about the switching activity of a design to perform power analysis and power optimization. You can use simulation tools to generate switching activity information in the Switching Activity Interchange Format (SAIF).

For information about SAIF files, see the following topics:

- [About Switching Activity](#)
 - [Introduction to SAIF Files](#)
 - [Generating SAIF Files](#)
 - [Verilog Switching Activity Examples](#)
 - [VHDL Switching Activity Example](#)
-

About Switching Activity

The dynamic power component usually accounts for a large percentage of the total power consumption in a combinational circuit. Dynamic power is the sum of the internal power of cells and the switching power. Switching power is the rate of energy usage resulting from the charging and discharging of capacitive loads during transitions between the two logic states, 0 and 1. Switching power depends on the clock rate and also the rate at which toggling occurs between logic states on each net. The toggle rate depends on the data being processed during typical usage of the logic circuit.

The Power Compiler tool models switching activity based on the following:

- Static Probability

The fraction of time that a signal is at the logic 1 state. For example, a static probability of 0.8 means that the signal is in the logic 1 state 80 percent of the time and the logic 0 state 20 percent of the time.

- Toggle Rate

The rate at which a signal changes from 0 to 1 and from 1 to 0, in number of transitions per time unit.

When the switching activity information is available, you should annotate this information on the design objects so that the tool can use the switching activity information during power optimization and analysis.

For more information about annotating switching activity, see [Annotating Switching Activity](#).

Introduction to SAIF Files

The accuracy of power calculations depends on the accuracy of the switching activity data. This data is generated using RTL simulation or gate-level simulation and is stored in a SAIF file. You should use the SAIF file to annotate switching activity information on the design objects before you perform power optimization and analysis.

SAIF is an ASCII format supported by Synopsys to facilitate the interchange of information between various Synopsys tools (see the *IEEE 1801 Standard, Annex J*). Use the `read_saif` command to read the SAIF file and the `write_saif` command to write out the SAIF file.

For more information, see the `read_saif` and the `write_saif` command man pages.

Early in the design cycle, you can use RTL simulation to determine the high-level switching and power characteristics of the design. Later in the design cycle, you can use gate-level simulation to get more detailed switching data to annotate your design. The detailed switching data increases the accuracy of the power optimization and power analysis.

[Table 3](#) summarizes the various methods of generating SAIF files and their accuracies.

Table 3 Comparing Methods of Capturing Switching Activity

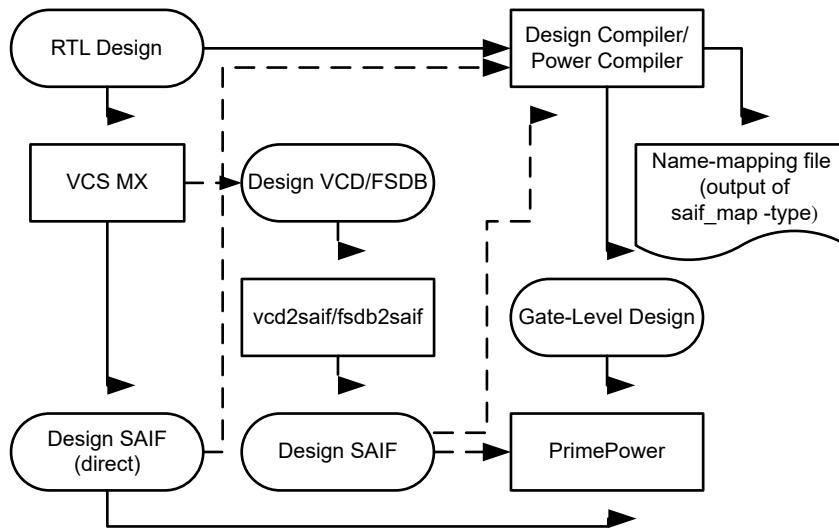
Simulation	Captured	Not captured	Trade-offs
RTL	Synthesis-invariant elements	1. Internal nodes 2. Correlation of non-synthesis-invariant elements 3. Glitching 4. State and path dependencies	Fast runtime at expense of some accuracy
Zero-delay and unit-delay gate-level	1. Synthesis-invariant elements 2. Internal nodes 3. Correlation 4. State dependencies 5. Some path dependencies	1. Some path dependencies 2. Glitching	More accurate than RTL simulation, but significantly higher runtime
Full-timing gate-level	1. All elements of design 2. Correlation 3. State and path dependencies	Highest accuracy, but runtime can be very long	Correlation between primary inputs

Generating SAIF Files

You can generate a SAIF file either from RTL simulation or gate-level simulation. This section discusses both RTL and gate-level simulation using Synopsys VCS. VCS supports Verilog, SystemVerilog, and VHDL formats.

Figure 8 shows two ways of generating a SAIF file. The solid lines indicate the suggested SAIF flow while the dotted lines indicate the alternative method of SAIF flow using various Synopsys tools.

Figure 8 SAIF File Generation and its Usage With Various Synopsys Tools



The following topics describe ways of generating SAIF files:

- [Generating SAIF Files From Simulation](#)
- [Generating SAIF Files From VCD Files](#)
- [Generating SAIF Files From FSDB Output Files](#)

You can read the SAIF file into the Power Compiler tool and generate a mapping file for all the name changes of the nodes. You then read the name-mapping file and the synthesized gate-level netlist in the PrimePower tool to perform averaged power analysis

Generating SAIF Files From Simulation

VCS MX can generate the SAIF file directly from simulation. This direct SAIF file is smaller than VCD or FSDB files. Your input design for simulation can be an RTL or gate-level design. The design can be in Verilog, SystemVerilog, VHDL, or mixed HDL formats. When

your design is in Verilog or SystemVerilog formats, you must specify system tasks to VCS MX using toggle commands. If your design is in VHDL format, use the power command as described in [Generating SAIF Files From VHDL Simulation](#).

For more information about the various supported formats and mixed language formats, see the *VCS MX User Guide*.

When generating the SAIF file during simulation, use the default monitoring policy (see the *VCS MX User Guide* for more information). This monitoring captures the switching activity of only the synthesis-invariant objects such as ports, tristate cells, black box cells, flip-flops, latches, retention registers, and hierarchical cells other than clock-gating cells. Integrated clock-gating cells and latch-based isolation cells are synthesis-dependent objects and therefore not captured.

If the library forward SAIF file contains details of state and path dependencies, the backward SAIF file generated also contains these details. For more information, see [Capturing State- and Path-Dependent Switching Activity](#).

The steps to generate SAIF files from simulation are discussed in the following topics:

- [Generating SAIF Files From SystemVerilog or Verilog Simulations](#)
- [Generating SAIF Files From Gate-Level Simulation](#)
- [VCS MX Toggle Commands](#)
- [Generating SAIF Files From VHDL Simulation](#)

Generating SAIF Files From SystemVerilog or Verilog Simulations

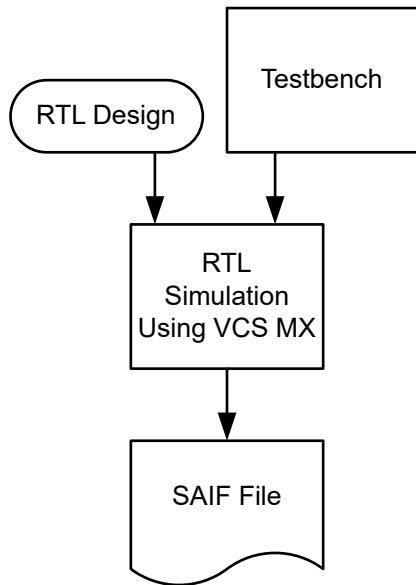
Using VCS MX, you can generate SAIF files from both RTL and gate-level Verilog designs. When your design is in Verilog format, you must specify system tasks to VCS MX. These system tasks are also known as toggle commands. The system tasks specify the module for which switching activity is to be recorded and reported in the SAIF file. They also control the toggle monitoring during simulation.

For details about the toggle commands, see [VCS MX Toggle Commands](#).

Generating SAIF Files From RTL Simulation

[Figure 9](#) presents the methodology to capture switching activity using RTL simulation. RTL simulation captures the switching activity of primary inputs, primary outputs, and other synthesis-invariant elements.

Figure 9 RTL Simulation Using VCS MX



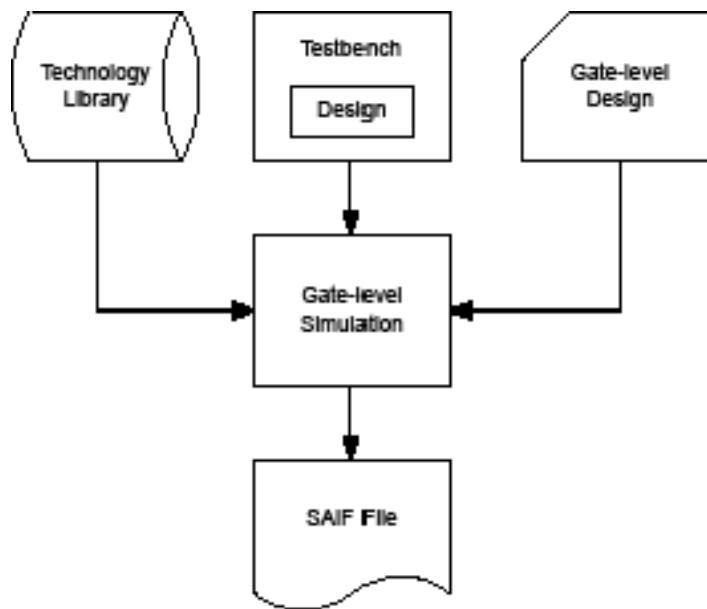
To capture the switching activity using RTL simulation, specify the appropriate testbench and run the simulation.

The SAIF file contains the switching activity information of the synthesis-invariant elements in your design. To use the SAIF file for synthesis in the Power Compiler tool, annotate the switching activity, as described in [Annotating Switching Activity](#).

Generating SAIF Files From Gate-Level Simulation

[Figure 10](#) presents the methodology to capture switching activity using gate-level simulation. Gate-level simulation captures switching activity of pins, ports, and nets in your design.

Figure 10 Gate-Level Simulation Using VCS MX



To capture switching activity using gate-level simulation, specify the appropriate toggle commands in the testbench and run the simulation.

The SAIF file contains information about the switching activity of the pins, ports, and nets in your design. It can represent the pin-switching activity, based on rise and fall values, if your logic library has separate rise and fall power tables.

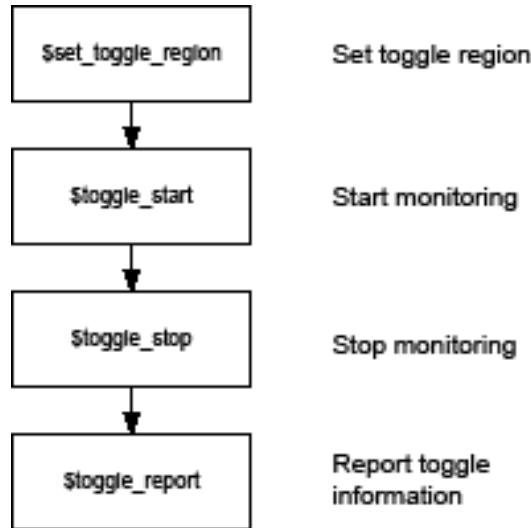
To use the SAIF file for synthesis in the Power Compiler tool, annotate the switching activity as described in [Annotating Switching Activity](#).

VCS MX Toggle Commands

To generate the SAIF file from RTL or gate-level Verilog or SystemVerilog, use toggle commands to specify system tasks to VCS MX. Using the toggle commands, you can specify the subblock for toggle counting and define specific periods for toggle counting during simulation. You can also control the start and stop of toggle counting.

[Figure 11](#) presents an overview of the toggle commands in your testbench file. Each toggle command starts with the \$ symbol. For simplicity, the figure does not show optional commands.

Figure 11 Toggle Command Flow



The system tasks that you specify to VCS MX using the toggle commands are

1. Define the toggle region.

The `$set_toggle_region` command specifies the module instance for which the simulator records the switching activity in the generated SAIF file. The syntax of this command is as follows:

```
$set_toggle_region(instance [, instance]);
```

When you explicitly mention one or more module instances as the toggle region, the simulator registers these objects and monitors them during simulation.

Note:

For gate-level simulation, if the logic library cell pins have rise and fall power values, their switching activity is monitored and reported for rise and fall separately.

2. Begin toggle monitoring.

Use the `$toggle_start` command to instruct the simulator to start monitoring the switching activity. The syntax of this command is as follows:

```
$toggle_start();
```

During simulation, the tool starts monitoring the switching activity of the module instances that are defined in the toggle region.

3. End toggle monitoring.

Use the `$toggle_stop` command to instruct the simulator to stop monitoring the switching activity.

4. Report toggle information in an output file.

Use the `$toggle_report` command to write monitored gate and net switching activity to an output file. You can invoke `$toggle_report` any number of times using different parameters. For more details and examples of SAIF files, see [RTL SAIF File](#).

The syntax for the `$toggle_report` command is as follows:

```
$toggle_report (filename, [synthesis_time_unit], instance_name_string);
```

The values for the various options and parameters are as follows:

- *filename*

This is the name of the switching activity output file.

- *synthesis_time_unit*

This optional parameter is the time unit of your synthesis library, in seconds.

For example, if the time unit in your synthesis library is 10 picoseconds, specify `1.0e-11`. The `$toggle_report` command uses this number to convert simulation time units to synthesis time units. The Power Compiler tool obtains the simulation time unit from simulation. If you don't specify the synthesis time unit parameter, the default is 1 ns (`1.0e-9`).

- *instance_name_string*

This required parameter is the full instance path name of the block from the top of your simulation environment down to the name of the block instance to be reported.

Example

```
$toggle_report ("file.saif", 1.0e-11, "test.DUT");
```

In this example, the file written out is `file.saif`, the synthesis time unit is 10 picoseconds, and the name of the monitored instance is `test.DUT`. The output file format is SAIF, which is the default.

Resetting the Toggle Counter

Use the `$toggle_reset` command to set the toggle counter to 0 for all the nets in the current toggle region. This command starts a new toggle monitoring period in a simulation session.

For example, when using `$toggle_start`, `$toggle_stop` or `$toggle_reset` with `$toggle_report`, you can create SAIF output files for specific periods during simulation. The syntax of this command is as follows:

```
$toggle_reset();
```

Use the `$toggle_reset` command only after you have written out the previous results with the `$toggle_report` command.

Capturing State- and Path-Dependent Switching Activity

By default, the Power Compiler tool estimates the state- and path-dependent power information that is required for power calculations. However, if you want to obtain this information through simulation, you can use the `lib2saif` command before simulation. In this case, given a logic library, you can run the utility to obtain a library SAIF file that contains the directives for generating state- and path-dependent switching information. This file is called the library forward SAIF file. This file becomes an input to gate-level simulation.

The library forward SAIF file contains information from the logic library about cells that have state and path dependencies. It can have rise and fall information if the library has separate rise and fall power tables.

To read the library forward SAIF file into the simulator, use the `$read_lib_saif` command. This command registers the state- and path-dependent information for monitoring during simulation.

The syntax of the `$read_lib_saif` command is as follows:

```
$read_lib_saif(input_file);
```

For gate-level simulation, you must use the `$read_lib_saif` command to register state- and path-dependent cells and, by default, all internal nets in the design. The command registers state-dependent and path-dependent cells by reading the library forward SAIF file. In addition, you must also set the toggle region for monitoring. If you do not use the `$read_lib_saif` command, the simulator registers all internal nets for monitoring by default.

You can use the `$read_lib_saif` command as often as you require during simulation; however, you must use this command before defining the toggle region using the `$set_toggle_region` command. When you define the toggle region, the `$set_toggle_region` command checks for the presence or absence of a previous `$read_lib_saif` command and registers internal nets accordingly.

[Example 3](#) shows an example of a SAIF file generated by the `lib2saif` command.

Example 3 File Generated by lib2saif

```
(SAIFFILE
(SAIFVERSION "2.0" "lib")
```

Chapter 4: Generating SAIF Files

Generating SAIF Files

```
(DIRECTION "forward")
(DESIGN )
(DATE ...)
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "Power Compiler lib2saif") (VERSION ...) (DIVIDER / )
(LIBRARY "example"
(MODULE "AND2"
(PORT
(Z
(IOPATH A IOPATH B)
)
)
)
(MODULE "dff1"
(LEAKAGE
(COND Q
COND !Q
COND_DEFAULT)
)
)
(MODULE "EXOR3"
(PORT
(Z
(COND ((!B * !A) | (B * A)) RISE_FALL (IOPATH C)
COND ((!B * A) | (B * !A)) RISE_FALL (IOPATH C)
COND ((!C * !A) | (C * A)) RISE_FALL (IOPATH B)
COND ((!C * A) | (C * !A)) RISE_FALL (IOPATH B)
COND ((!C * !B) | (C * B)) RISE_FALL (IOPATH A)
COND ((!C * B) | (C * !B)) RISE_FALL (IOPATH A)
COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH C))
)
)
(MODULE "MUX21"
(PORT
(Z
(COND (B * !A) RISE_FALL (IOPATH S)
COND (!B * A) RISE_FALL (IOPATH S)
COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH S))
)
)
(LEAKAGE
(COND (B * S * A)
COND (!B * S * A)
COND (!B * !S * A)
COND (!A * S * B)
COND (!A * !S * B)
COND_DEFAULT)
)
)
(MODULE "NAND2"
(PORT
(Z
```

```

        (IOPATH A IOPATH B)
    )
)
)
(MODULE "OR2"
(PORT
(Z
(IOPATH A IOPATH B)
)
)
)
)

.....
(MODULE "iopad6"
(PORT
(PAD
(COND !TS RISE_FALL (IOPATH DI)
COND_DEFAULT RISE_FALL)
)
(DI
(COND TS RISE_FALL
COND_DEFAULT RISE_FALL)
)
(DO
(IOPATH PAD IOPATH_DEFAULT)
)
)
)
)
)
)
```

Overriding Default Registration of Internal Nets

After you have run the `read_lib_saif` command in the testbench, you can override the default net monitoring behavior using the `$set_gate_level_monitoring` command. This command turns on or turns off the registration of internal nets.

The following is the syntax of the `$set_gate_level_monitoring` command:

```
$set_gate_level_monitoring ("on" | "rtl_on", "mda" | "sv");
```

"on"

This string explicitly registers all internal nets for simulation. Thus, simulation monitors any internal net in the region defined by the `$set_toggle_region` command.

"rtl_on"

The registers in the toggle region are monitored and the nets in the toggle region are not monitored during simulation.

"mda"

Use this argument for Verilog memories and multidimensional arrays.

"sv"

Use this argument for SystemVerilog data objects.

The `$set_gate_level_monitoring` command is optional. If you use it, you must do so before invoking the `$set_toggle_region` command.

Generating SAIF Files From VHDL Simulation

You can use VCS MX to generate SAIF files from RTL or gate-level simulation of VHDL designs. The methodology to generate the SAIF file is similar to the methodology used for Verilog designs, shown in [Figure 9](#) and [Figure 10](#). However, you cannot use the toggle commands to specify the system tasks to the simulator.

For RTL-level VHDL files, variables are not supported by the simulator for monitoring. However, VHDL constructs such as generates, enumerated types, records, and arrays of arrays are supported by VCS MX for simulation.

The use model to generate a SAIF file from VHDL simulation consists of using the `power` command at the VCS MX command line interface, simv. The syntax of the `power` command is as follows:

```
power
  -enable
  -disable
  -reset
  -report file_name synthesis_time_unit scope
  -rtl_saif file_name
  [test_bench_path_name]
  -gate_level on| off | rtl_on
  region_signal_variable
```

- The `-enable` option enables the monitoring of the switching activity.
- The `-disable` option disables the monitoring of the switching activity.
- The `-reset` option resets the toggle counter.
- The `-report` option reports the switching activity to an output SAIF file.
- The `-rtl_saif` option reads the RTL forward-SAIF file.

- You can use `on`, `off`, or `rtl_on` with the `-gate_level` option. [Table 4](#) summarizes the monitoring policy for VHDL simulation.

Table 4 Monitoring Policy for VHDL Simulation

Monitoring policy	Ports	Signals	Variables
<code>on</code>	Yes	Yes	No
<code>off</code>	No	No	No
<code>rtl_on</code>	Yes	Yes	No

- You can specify either the hierarchical path to the signal name or the toggle region and its children to be considered for monitoring.

System Task List for SAIF File Generation From VHDL Simulation

The following example script shows a task list that you specify to the simulator to generate a SAIF file. The design name in the example is `test1`. You can either specify each of these commands at the VCS MX command prompt or run the file that contains these commands.

```
power test1
power -enable
run 10000
power -disable
power -report vhdl.saif 1e-09 test
quit
```

Generating SAIF Files From VCD Files

You can generate SAIF files from VCD files. To generate a SAIF from a VCD file generated by the VCS tool, use the `vcd2saif` utility. Follow these steps to generate the SAIF file and to annotate the switching activity:

1. Run the simulation to generate VCD file.
2. Use the `vcd2saif` utility to convert the VCD file to a SAIF file.
3. Annotate the switching activity within the SAIF file as described in [Annotating Switching Activity](#).

The disadvantage of using this method is that VCD files can be very large, especially for gate-level simulation, requiring more time for processing. Also, the SAIF file generated by the `vcd2saif` utility lacks state-dependent and path-dependent information.

Converting a VCD File to a SAIF File

The `vcd2saif` utility converts the RTL or gate-level VCD file generated by VCS into a SAIF file. This utility has limited capability when the VCD is generated from the SystemVerilog simulation as described in [Limited SystemVerilog Support in the vcd2saif Utility](#).

The `vcd2saif` utility is platform-specific and is located in `install_dir/$ARCH/syn/bin`. The `$ARCH` environment variable represents the specific platform (architecture) of your Synopsys software installation, such as `linux`.

You can use compressed VCD files (.Z) and gzipped VCD files (.gz). In addition, for VPD files, you can use the utility located at `$VCS_HOME/bin/vpd2vcd`, and for FSDB files, you can use the utility located at `$SYNOPSYS/bin/fsdb2vcd`.

The `vcd2saif` utility does not support state-dependent and path-dependent switching activity. For information about each option, use the `vcd2saif -help` command.

Limited SystemVerilog Support in the `vcd2saif` Utility

The `vcd2saif` utility supports only a limited set of SystemVerilog constructs for VCD files generated from SystemVerilog simulation. [Table 5](#) shows the list of SystemVerilog constructs that are supported by the `vcd2saif` utility.

Table 5 SystemVerilog Constructs Supported by the `vcd2saif` Utility

char	int	shortint	longint	bit	byte	logic
shortreal	void	enum	typedef	struct	union	arrays (packed and unpacked)

Generating SAIF Files From FSDB Output Files

There are two ways to generate a SAIF file from an FSDB file:

1. Using the `fsdb2saif` utility
2. Using the `fsdb2vcd` utility and then using the `vcd2saif` utility.

For more information about the FSDB utilities, see the *Verdi3 and Siloti Command Reference Manual*. After generating the SAIF file, annotate the switching activity from the SAIF file as described in [Annotating Switching Activity](#).

Verilog Switching Activity Examples

The following examples demonstrate RTL and gate-level descriptions with Verilog-generated switching activity data.

RTL Example

This Verilog RTL example includes the following elements:

- RTL design description
- RTL testbench
- SAIF output file from simulation

Verilog Design Description

[Example 4](#) shows the description for a state machine called test.

Example 4 RTL Verilog Design Description

```

`timescale 1 ns / 1 ns
module test (data, clock, reset, d_out);

    input [1:0] data;
    input clock;
    input reset;
    output d_out;
    wire d_out;
    wire [1:0] NEXT_STATE;
    reg [1:0] PRES_STATE;

    parameter s0 = 2'b00;
    parameter s5 = 2'b01;
    parameter s10 = 2'b10;
    parameter s15 = 2'b11;

    function [2:0] fsm;
        input [1:0] fsm_data;
        input [1:0] fsm_PRES_STATE;

        reg fsm_d_out;
        reg [1:0] fsm_NEXT_STATE;

        begin
            case (fsm_PRES_STATE)
                s0: //state = s0
                begin
                    if (fsm_data == 2'b10)
                    begin
                        fsm_d_out = 1'b0;

```

```

        fsm_NEXT_STATE = s10;
    end
    else if (fsm_data == 2'b01)
        //...
end

s5: //state = s5
begin
    // ...
end

s10: //state = s10
begin
    // ...
end

s15: //state 15
begin
    // ...
end
endcase

fsm = {fsm_d_out, fsm_NEXT_STATE};
end

endfunction

assign {d_out, NEXT_STATE} = fsm(data, PRES_STATE);

always @ (posedge clock)
begin
    if (reset == 1'b1)
        begin
            PRES_STATE = s0;
        end
    else
        begin
            PRES_STATE= NEXT_STATE;
        end
end
endmodule

```

RTL Testbench

The Verilog testbench in [Example 5](#) simulates the design test described in [Example 4](#). The testbench instantiates the design test as U1.

Example 5 RTL Testbench

```

`timescale 1 ns / 1 ns
module stimulus;

reg clock;

```

```

reg [1:0] data;
reg reset;
wire d_out;
test U1 (data,clock, reset, d_out);

always
begin
    #10 clock = ~clock;
end

initial
begin
$set_toggle_region(stimulus.U1);
$toggle_start();
clock = 1'b0;
data = 2'b00;
reset = 1'b1;
#50 reset = 0;
#25 data = 3; #20 data = 0;
#20 data = 1; #20 data = 2;
// ...
$toggle_stop();
$toggle_report("my_rtl_saif", 1.0e-12, "stimulus");
#80 $finish;
end

```

RTL SAIF File

The RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `$toggle_report` command creates this file.

[Example 6](#) is a SAIF file created for the RTL Verilog description that is also shown in [Example 4](#) and for the testbench shown in [Example 5](#).

Example 6 RTL SAIF File

```

/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets **/
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Fri Feb 5 14:21:20 2021")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS I-2014.03-SP1")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ps)
(DURATION 135000.00)
(INSTANCE stimulus
    INSTANCE U1

```

```

        (NET
            (data\[1\]
                (T0 115000) (T1 20000) (TX 0)
                (TC 2) (IG 0)
            )
            (data\[0\]
                (T0 95000) (T1 40000) (TX 0)
                (TC 3) (IG 0)
            )
            (clock
                (T0 70000) (T1 65000) (TX 0)
                (TC 13) (IG 0)
            )
            (reset
                (T0 85000) (T1 50000) (TX 0)
                (TC 1) (IG 0)
            )
            (d_out
                (T0 0) (T1 0) (TX 135000)
                (TC 0) (IG 0)
            )
            (NEXT_STATE\[1\]
                (T0 0) (T1 0) (TX 135000)
                (TC 0) (IG 0)
            )
            (NEXT_STATE\[0\]
                (T0 0) (T1 0) (TX 135000)
                (TC 0) (IG 0)
            )
        )
    )
}
}

```

Understanding the SAIF File

[Table 6](#) summarizes the definitions for SAIF file terminology.

Table 6 Definitions of SAIF File Terms

T0	Duration of time found in logic 0 state.
T1	Duration of time found in logic 1 state.
TX	Duration of time found in unknown "X" state.
TC	The sum of the rise (0-to-1) and fall (1-to-0) transitions that are captured during monitoring.
IG	Number of 0 - X - 0 and 1 - X - 1 glitches captured during monitoring.
RISE	Rise transitions in a given state.

Table 6 Definitions of SAIF File Terms (Continued)

FALL	Fall transitions in a given state.
------	------------------------------------

Duration refers to the time span between `$toggle_start` and `$toggle_stop` commands in the testbench during simulation. During this time span, ports, pins, and nets are monitored for toggle activity.

Gate-Level Example

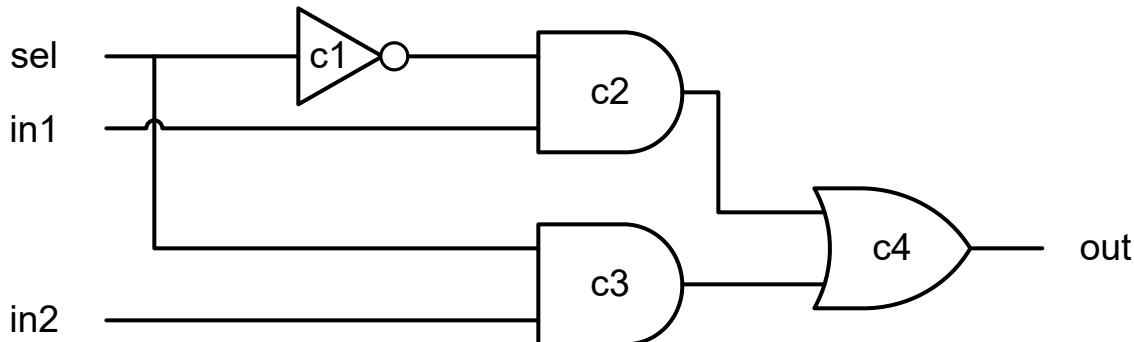
This Verilog gate-level example illustrates the following elements:

- Verilog cell description and schematic
- Verilog testbench
- SAIF output file from simulation

Gate-Level Verilog Module

[Figure 12](#) shows the schematic for a simple multiplexer.

Figure 12 Schematic of Multiplexer Circuit: MUX21



[Example 7](#) is the Verilog module that describes the MUX21 design.

Example 7 Verilog Module of Multiplexer Circuit: MUX21

```

/*'timescale 10ps/ 1ps
*/
module MUX21(out,d1,d2,sel);
    input d1, d2, sel;
    output out;
    IV c1(.Z(sel_),.A(sel));
    AN2 c2(.Z(d1m),.A(d1),.B(sel_));
    AN2 c3(.Z(d2m),.A(d2),.B(sel_));

```

```
    OR2 c4(.Z(out), .A(d1m), .B(d2m));
endmodule
```

Verilog Testbench

The Verilog testbench in [Example 8](#) tests the MUX21 design by simulating it and monitoring the various signals.

Example 8 Verilog Testbench for MUX21

```
/* Begin test.v */
'timescale 1ns/ 10ps
module top;
    reg in1, in2, sel;
    parameter hazrate = 0.99;
    parameter haztime = 0.23;

    MUX21 m1(out,in1,in2,sel);

    initial
    begin
        // start monitoring
        $monitor($time,,, "in1=%b in2=%b sel=%b
out=%b",in1,in2,sel,out);

        // read SAIF file of state or path dependent information
        $read_lib_saif (cell.saif);

        // define the monitoring scope
        $set_toggle_region (m1);

        $toggle_start;

        // test first data line passing 0
        sel = 0;
        in1 = 0;
        in2 = 0;

        // test first data line passing 1
        #10 in1 = 1;

        #10 sel = 1;

        // test second data line passing 1
        #10 in2 = 1;

        $toggle_stop;
        $toggle_report("my_1st", 1.0e-9,"top.m1", hazrate, haztime);

        // exit simulation
        $finish(2);
    end
endmodule
```

The `$set_toggle_region` command sets the monitoring scope in module m1 (the testbench instantiation of MUX21). All subsequent toggle commands affect only registered design objects and designs instantiated in registered objects. Thus, under m1, simulation monitors internal nets and state- and path-dependent cells (in this simple example, however, there are no subdesigns in m1).

The testbench example invokes `$toggle_report` command before exiting the simulation. Make sure that you declare any parameters you use for `$toggle_report` command in your testbench. These parameters appear at the top of the testbench in [Example 8](#).

Gate-Level SAIF File

[Example 9](#) shows a SAIF file generated from gate-level simulation of MUX21.

Example 9 \$toggle_report Output File in SAIF

```
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Fri Oct  6 18:58:58 2000")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-MX Power Compiler")
(VERSION "3.3")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 99999.00)
(INSTANCE tb
  (INSTANCE dut
    (NET
      (n12159
        (T0 99529) (T1 470) (TX 1)
        (TC 46) (IG 0)
      )
      (n12480
        (T0 0) (T1 99998) (TX 0)
        (TC 0) (IG 0)
      )
      (n12117
        (T0 61) (T1 99938) (TX 0)
        (TC 26) (IG 0)
      )
    )
  )
  (INSTANCE U12053
    (PORT
      (Z
        (T0 10) (T1 99989) (TX 0)
        (COND A (RISE)
          (IOPATH B (TC 0) (IG 0)
        )
        COND A (FALL)
        (IOPATH B (TC 0) (IG 0)
      )
    )
  )
)
```

```
)  
COND B (RISE)  
  (IOPATH A (TC 0) (IG 0)  
 )  
COND B (FALL)  
  (IOPATH A (TC 1) (IG 0)  
 )  
COND_DEFAULT (TC 1) (IG 0)  
 )  
 )  
 )
```

VHDL Switching Activity Example

This VHDL RTL example includes the following elements:

- RTL design description
 - RTL testbench
 - SAIF output file from simulation

VHDL Design Description

[Example 10](#) shows the description for a design called ABC.

Example 10 RTL VHDL Design Description

```
library ieee;
use ieee.std_logic_1164.all;
entity ABC is
architecture beh of ABC is
  signal clk: std_logic := '0';
begin
  clk <= not clk after 5 ns;
end beh;
```

RTL Testbench

The RTL testbench in [Example 11](#) simulates the design test described in [Example 10](#). The testbench instantiates the design ABC as ABC_ins.

Example 11 RTL Testbench

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
end entity;
architecture testbench of test is
component ABC is
end component;
begin
ABC_ins: ABC;
end testbench;
```

RTL SAIF File

This RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `power -report` command creates this file.

[Example 12](#) is a SAIF file for the RTL VHDL description that is shown in [Example 10](#).

Example 12 RTL SAIF File

```
/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets **/
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Tue May  5 05:56:35 2009")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-Scirocco-MX Power Compiler")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 10000.00)
(INSTANCE TEST
(INSTANCE ABC_INS
(NET
(CLK
(T0 5000) (T1 5000) (TX 0)
(TC 1999) (IG 0)
)
)
)
)
```

5

Annotating Switching Activity

You can annotate switching activity on designs to generate accurate power calculations.

For information about the different types of switching activity information and how to annotate it on gate-level designs, see the following topics:

- [Types of Switching Activity to Annotate](#)
 - [Annotating Switching Activity Using RTL SAIF Files](#)
 - [Annotating Switching Activity Using Gate-Level SAIF Files](#)
 - [Annotating Inferred Switching Activity](#)
 - [Annotating Switching Activity Using the set_switching_activity Command](#)
 - [Fully Versus Partially Annotating the Design](#)
 - [Analyzing Switching Activity Annotation](#)
 - [Removing the Switching Activity Annotation](#)
 - [Design Objects Without Annotated Switching Activity](#)
-

Types of Switching Activity to Annotate

The power of a design depends on the switching activity of the nets and cell pins. The switching activity is used by the `report_power` command during power calculation.

The following types of switching activity can be annotated on design objects:

- Simple switching activity on design nets, ports, and cell pins. Simple switching activity consists of the static probability and the toggle rate. The static probability is the fraction of the time that the object is at logic 1. The toggle rate is the rate at which the design object switches between logic 0 and logic 1.
- State-dependent toggle rates on input pins of leaf cells. As explained in [Power Modeling and Calculations](#), the internal power characterization of an input pin of a library cell can be state dependent. The input pins of instances of such cells can be annotated with state dependent toggle rates.

- State-dependent and path-dependent toggle rates on output pins of leaf cells. As explained in [Power Modeling and Calculations](#), the internal power characterization of output pins can be state dependent and path dependent. Output pins of cells with state- and path-dependent characterization can be annotated with state- and path-dependent toggle rates.
- State-dependent static probability on leaf cells. Cell leakage power can be characterized using state dependent leakage power tables (see [Power Modeling and Calculations](#)). Such cells can be annotated with state-dependent static probability.

Annotating Switching Activity Using RTL SAIF Files

Optimal power analysis and optimization results occur when switching activities reported in the RTL SAIF file are accurately associated with the correct design objects in the gate-level netlist. For this to occur, the RTL names must map correctly to their gate-level counterparts. During synthesis, however, mapping inaccuracies can occur that can affect your annotation.

To ensure proper name mapping and annotation for RTL SAIF files, do the following:

1. At the beginning of synthesis, specify the `saif_map -start` command.

This command causes the Design Compiler tool to create a name-mapping database during synthesis optimization that the Power Compiler tool then uses for power analysis and optimization.

2. Before compiling, specify the `read_saif -auto_map_names` command to perform RTL SAIF annotation using the name-mapping database.

Using the Name-Mapping Database

You can access the name-mapping database by using the `saif_map` command, which allows you to query, report, modify, save, clear, and load the database. If the object names require modification, use the `read_saif -auto_map_names` command. You can read a regular, uncompressed file or a compressed file in gzip format by using the `-input` option of the `saif_map` command.

After you run the `read_saif -auto_map_names` command, review the name-mapping database using the following commands:

```
read_saif -auto_map_names -input ../sim/rtl.saif  \
    -instance_name tb/dut -verbose
report_saif -hier -rtl_saif -missing
```

You can manually add a mapping entry with the `saif_map -add_name` command as follows:

```
reset_switching_activity
saif_map -add_name "Ax_ins" [get_ports AX_usr_ins]
read_saif -auto_map_names -input ../sim/rtl.saif \
-instance_name tb/dut
```

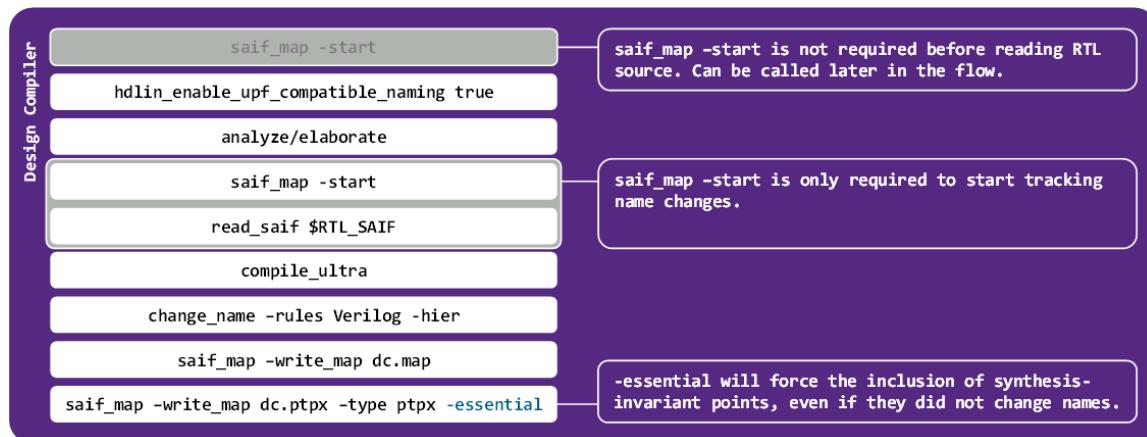
In this example, you manually map the `Ax_ins` RTL SAIF object and the `AX_usr_ins` design object. When you run the `read_saif -auto_map_names` command, the tool performs annotation again using the modified database.

The name-mapping mechanism no longer requires initialization. With this methodology, the `saif_map -start` command is no longer required before reading RTL source files.

Instead, `dclink` callback functions are registered automatically during the `analyze`, `elaborate`, or `read` command. The call-back functions used to track the mapping are registered only when you initiate the `saif_map -start` command. Thus, the name-mapping database can be initialized even without any SAIF files.

[Figure 13](#) shows the flow diagram for the recommended flow.

Figure 13 Recommended flow



The UI is consistent with the Fusion Compiler and IC Compiler II tools, where the `saif_map -start` command needs to be called to start the tracking of the name database.

To turn off this feature, set the following variable to `FALSE`:

```
set pwr_saif_map_auto_start FALSE
saif_map -start
```

If you have not set up a `saif_name` for an object in the SAIF file using the `-set_name` or `-add_name` options, the `saif_map -get_name` command derives a `saif_name` and returns it.

To force the inclusion of all synthesis invariant points in the mapping file regardless of whether or not the objects have undergone a major name change, use the `-essential` option of the `saif_map -write_map -type ptx` command. The `-essential` option only applies to the PrimePower format mapping file in which the following objects are included:

- All primary inputs
- All register output pins
- All pre-existing integrated clock-gating (ICG) output pins
- All macro input and output pins

Integrating the RTL Annotation With the PrimePower tool

The PrimePower tool requires accurate RTL-to-gate name-mapping correspondence to perform accurate power analysis. Use the Power Compiler tool to output the name-mapping files that the PrimePower tool uses for RTL-to-gate name mapping.

After using the `read_saif` command, specify the `saif_map` command as follows to generate a name-mapping file that can be read directly into PrimePower:

```
saif_map -type primepower -write_map file_name
```

The name-mapping output file appears as follows:

```
set_rtl_to_gate_name -rtl{clk_sn} -gate clk_sn
set_rtl_to_gate_name -rtl{rx_top/data_i[9]} \
    -gate rx_top_data_i_reg<9>
...

```

[Example 13](#) shows the recommended flow using RTL SAIF file to ensure optimal power analysis during synthesis and get the proper names for PrimePower.

Example 13 Annotating Switching Activity Using RTL SAIF Files Flow

```
saif_map -start
read_verilog rtl_design.v
link
create_clock clk
read_saif -auto_map_names -input ../sim/rtl.saif \
    -instance_name tb/dut
report_saif -hierarchy -rtl_saif
compile_ultra
report_saif -hierarchy -rtl_saif
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output mapped_design.v
saif_map -type primepower -write_map saifmap.primepower.tcl
```

Annotating Switching Activity Using Gate-Level SAIF Files

You can use either the `read_saif` or `merge_saif` command to annotate switching activity. The `read_saif` command reads a SAIF file and annotates switching activity information on the nets, pins, and ports of the design.

The `merge_saif` command reads a list of SAIF files, computes the toggle rates and static probability, and annotates the switching activity information on the nets, pins, and ports of the design. This command creates a merged-output SAIF file.

Reading SAIF Files Using the `read_saif` Command

To annotate gate-level switching activity onto the gate-level netlist, use the `read_saif` command. For example,

```
dc_shell> read_saif -input myfile.saif -auto_map_names \
    -instance_name T1/DUT/U1
```

In this example, the `read_saif` command annotates the information in the input file named `myfile.saif` onto the current gate-level design, `U1`. The `-instance_name` option identifies the hierarchical location of the current design in the simulation environment.

The input file specified using the `-input` option of the `read_saif` command can be a text file or a compressed gzip file with a `.gzip` extension. For example,

```
dc_shell> read_saif -input myfile.gzip -instance_name T1/DUT/U1
```

A SAIF file is usually generated in the HDL simulation flow, where a simulation testbench instantiates the design being simulated and provides simulation vectors. The generated SAIF file contains the switching activity information organized in a hierarchical fashion, where the hierarchy of the SAIF file reflects the hierarchy of the simulation testbench. If a design is instantiated in the testbench (`tb`) as the instance `i`, then the SAIF file contains the switching activity information for the design under the hierarchy `tb/i`. In this case, specify

the `tb/i` instance name to the `-instance_name` option when reading the SAIF file as follows:

```
dc_shell> read_saif -input des.saif -instance_name tb/i
```

Specifying an invalid instance name results in having all or most of the switching activity stored in the SAIF file not read properly. An error message is printed if none of the information stored in the SAIF file is read by the `read_saif` command.

The SAIF file contains time duration values and specifies a time unit which is usually the time unit used during simulation. When reading the SAIF file, the `read_saif` command automatically converts the SAIF time units to the synthesis time units. The synthesis time units are obtained from the time units of the target or link library. When the synthesis time units cannot be obtained, the `read_saif` command prints a warning message and uses a default time unit of 1 ns. In such cases, you can use the `-scale` and `-unit_base` options to specify the intended synthesis time unit. For example, if a target library with the time unit of 100 ps is used for synthesis and a SAIF file is being read before the library is used (for linking or synthesis), use the options as follows:

```
dc_shell> read_saif ... -scale 100 -unit_base ps
```

Valid arguments for the `-unit_base` option are `s`, `ms`, `us`, `ns`, `ps`, and `fs`. The argument of the `-scale` option can be any floating-point numerical value.

After reading the SAIF file, the `report_lib` command gives the time units specified in a logic library. The `report_power` command gives the synthesis library time units used during power calculations.

Reading SAIF Files Using the `merge_saif` Command

The `merge_saif` command can be used to read switching activity information from multiple SAIF files. Input SAIF files are given individual weights, and a weighted sum of the switching activities is annotated. This command can be used in flows where different SAIF files are generated for different modes of the same design. The switching activity from all the different modes can then be used for power calculations and optimization.

The following example shows how to use the `merge_saif` command. In this example, the design has three modes: standby, slow, and fast; and the SAIF files are `standby.saif`, `slow.saif`, and `fast.saif`. Depending on the expected use of the design, specify the following weight for each SAIF file, with the total weight always equal to 100 percent:

`standby.saif: 80%; slow.saif: 5%; fast.saif: 15%`

The SAIF files are read as shown in the following example:

```
dc_shell> merge_saif -input_list \
    { -input standby.saif -weight 80 \
    -input slow.saif -weight 5 \
    -input fast.saif -weight 15 } \
    -instance_name tb/i
```

The `-output` option of the `merge_saif` command can be used to generate a SAIF file containing the weighted sum of the switching activities. When the output file is specified with the `.gzip` extension, the tool writes a compressed file in gzip format. If the output file is specified with the `.saif` extension, the tool writes an uncompressed SAIF format file.

After the `merge_saif` command reads each individual SAIF file, the tool uses a switching activity propagation mechanism to estimate the switching activity of design nets that are not included in the SAIF file. You can use the following command to generate a gate-level SAIF file with estimated switching activity information from an RTL SAIF file:

```
dc_shell> merge_saif -input_list { -input rtl.saif -weight 100 } \
    -instance_name tb/i -output estimate.saif
```

The `-simple_merge` option can be used to switch off the switching activity propagation mechanism when the information in the SAIF files is being merged.

The syntax of the `merge_saif` command is the same as that of the `read_saif` command with the following exceptions:

- A weighted input file list is specified instead of a single input file.
- The `-simple_merge` and `-output` options can be used with the `merge_saif` command.

Annotating Inferred Switching Activity

The `infer_switching_activity` command detects the drivers of special pins such as asynchronous set, asynchronous clear, synchronous set, and synchronous clear, and suggests values for toggle rate and static probability.

The `infer_switching_activity` command reports the current and proposed static probability and toggle rate, as shown in the following example:

```
dc_shell> infer_switching_activity
Information: Updating design information... (UID-85)
```

Created by `infer_switching_activity` ...

Current Static	Current Toggle	Proposed Static	Proposed Toggle
-------------------	-------------------	--------------------	--------------------

Objects	Type	Probability	Rate	Probability	Rate
U646/Z	driver	None	None	1.0	0.0
U645/ZN	driver	None	None	1.0	0.0

When you specify the `-apply` option, the proposed switching activity is annotated on the drivers listed in the output. After applying the switching activity, the Power Compiler tool uses the applied values to report the power consumption.

Use the `-sci_based` option to report or apply activity on essential points based on the type of special control inputs in the load. The `-sci_based` option can be one of the following values: `sci`, `non_sci`, or `all`.

To specify which scenario to apply annotation or which scenario to report, use the `-scenarios` option.

The `-scenarios` option can be used to specify which set of scenarios to infer switching activity. The `-sci_based` option allows you to customize which objects to infer switching activity. The `-sci_based` option has the following values: `sci`, `non_sci`, or `all`.

Annotating Switching Activity Using the `set_switching_activity` Command

The `set_switching_activity` command annotates switching activity on design objects such as pins, ports, nets, and cells. The types of activity that you can annotate include state- and path-dependent toggle rates and state-dependent static probabilities.

Use the `-static_probability` option to specify the static probability value, which is a floating-point number between 0.0 and 1.0. Static probability is the fraction of time that the signal is at logic 1.

Use the `-toggle_rate` option to specify the toggle rate value, which is a floating point number. Toggle rate is the number of low-to-high or high-to-low transitions made by the signal during a period of time.

The `-toggle_rate` option differs from the toggle rate used for modeling switching activity. The `-toggle_rate` option expresses the sum of the rise and fall transitions that the signal makes during an entire simulation, clock period, or other period you specify. The Power Compiler tool uses the `-toggle_rate` and `-period` (or `-clock`) options to determine the actual toggle rate per unit of time.

The following example specifies that the net `net1` is at logic 1 for 20 percent of the time, and that it transitions between logic values 0 and 1 an average of 10 times in 1000 time

units. The time unit used for the toggle rate is the time unit defined in the target library. The `-period` option is optional and defaults to a value of 1, when it is not specified.

```
dc_shell> set_switching_activity [get_nets net1] \
    -static_probability 0.2 -toggle_rate 10 -period 1000
```

Use the `-state_condition` option to annotate state-dependent toggle rates on pins or state-dependent static probabilities on cells. The state-dependent toggle rates can be annotated only if the library is characterized with state-dependent power tables for internal power, for the pins of the library cell. Similarly, state-dependent static probabilities can be annotated only if the library is characterized with state-dependent power tables for leakage power, for the library cells.

The following example shows how to use the `-state_condition` option to annotate the state-dependent toggle rates on pins. It specifies that the pin ff1/Q toggles 0.01 times when the pin D is at logic 1, and 0.03 times when the pin D is at logic 0.

```
dc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.01 \
    -state_condition "D"
dc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.03 \
    -state_condition "!D"
```

Use the `-rise_ratio` option to specify the ratio of rise transitions to the total transitions for the specified toggle rate. You can also use this option with state-dependent toggle rates to specify the ratio of rise transitions to fall transitions for the specified state. The following example specifies that the xor1/Y pin toggles 0.01 times when the cell is in A state, and that 90 percent of these toggles are rise toggles.

```
dc_shell> set_switching_activity [get_pins xor1/Y] -toggle_rate 0.01 \
    -state_condition "A" -rise_ratio 0.9
```

Use the `-path_sources` option to specify the path-dependent toggle rates. The following example specifies that the and1/Y pin toggles 0.02 times because of a toggle on the input pin A, but never toggles because of a toggle on the B pin. Toggle rates that are both state- and path-dependent are specified by using the `-state_condition` and `-path_sources` options together.

```
dc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.02 \
    -path_sources "A"
dc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.00 \
    -path_sources "B"
```

The state-dependent static probabilities can be annotated using the `-state_condition` option. The following example specifies that the cell named AND1 is at the A & B state for 10 percent of the time, at the A & !B state for 70 percent of the time, and at the !A state for 20 percent of the time.

```
dc_shell> set_switching_activity [get_cells AND1] \
    -static_probability 0.1 -state_condition "A & B"
```

```
dc_shell> set_switching_activity [get_cells AND1] \
           -static_probability 0.7 -state_condition "A & !B"
dc_shell> set_switching_activity [get_cells AND1] \
           -static_probability 0.2 -state_condition "!A"
```

To implicitly select outputs or cells to annotate, use the `-type` option and specify a list of following types of objects:

- Input, output, inout ports of design or input, output, inout pin of hierarchical cells
- Output of registers, output of sequential cells
- Output of black box cells
- Output of tristate cells
- Output of flip-flops clocked by the specified clocks
- Output of clock-gating cells
- Output of memory cells
- Nets

When you use the `set_switching_activity` command to annotate switching activity on all inputs, this includes the clock inputs as well. This results in overriding the switching activity on the clock inputs. To avoid overriding the switching activity on clock inputs, specify all inputs except the clock inputs, as shown in the following example:

```
dc_shell> set_switching_activity [remove_from_collection \
           [all_inputs] clk] \
           -static_probability sp_value -toggle_rate tr_value \
           -period period_value
```

Fully Versus Partially Annotating the Design

For the highest accuracy of power analysis, annotate all the elements in your design. To annotate all design elements, you must use gate-level simulation to monitor all the nodes of the design.

Using gate-level simulation, you can perform the following activities:

- Capture state- and path-dependent switching activity
- Capture switching activity that considers glitching (full-timing gate-level simulation only)

After layout, you can increase accuracy further by annotating wire loads with more accurate net capacitance values. However, if the design layout is performed at the foundry, you might not have access to the post-layout information.

If you annotate some design elements, the Power Compiler tool uses an internal zero-delay simulation to propagate switching activity through nonannotated nets in your design. The tool uses internal simulation if it encounters nonannotated nets during power analysis.

During switching activity propagation, the tool tracks which design elements are user-annotated with the `set_switching_activity` command and which are not. In calculating power, the tool does not overwrite user-annotated switching activity with propagated switching activity.

Power analysis and power optimization both require that you annotate at least the following:

- Primary inputs
- Outputs of synthesis-invariant elements such as black box cells
- Three-state devices
- Sequential elements
- Hierarchical ports

Note:

When performing power analysis on a partially annotated design, it is recommended that you specify a clock before running the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute and propagate switching activity through the design. Use the `create_clock` command to create a clock. If no clock is available, you get a PWR-80 warning message. This does not stop propagation but the estimated switching activity might not be accurate.

Analyzing Switching Activity Annotation

To report the annotated switching activity, use the `report_saif` or the `report_activity` commands. To query the activity on a pin or net, use the `get_switching_activity` command.

This section covers the following topics:

- [Using the report_saif Command to Report Switching Activity](#)
- [Using the report_activity Command to Report Switching Activity](#)
- [Retrieving Switching Activity on a Pin or Net](#)

Using the report_saif Command to Report Switching Activity

The `report_saif` command can be used to display information about the annotated switching activity. The report generated by this command shows the number and percentage of nets, ports, and pins annotated with user-annotated switching activity, default switching activity, and propagated switching activity, respectively. The command does not consider clock-gating cells as synthesis-invariant because these cells can be deleted or inserted during the optimization step. The following example shows the report generated by the command:

```
dc_shell> report_saif -hierarchy
*****
Report : saif
          -hierarchy
Design : des
Version: ...
Date   : ...
*****
-----
Object type      User           Default          Propagated
                  Annotated (%)  Annotated (%)  Activity (%) Total
-----
Nets            251 (99.21%)   1 (0.40%)     1 (0.40%)   253
Ports           59 (98.33%)    1 (1.67%)     0 (0.00%)   60
Pins            251 (99.60%)   0 (0.00%)     1 (0.40%)   252
-----
```

If the `-hierarchy` option is used, the switching activity information is generated for all design objects in the design hierarchy starting from the current instance. If this option is missing, then only design objects in the hierarchical level of the current instance are considered.

If the `-rtl_saif` option is used, switching activity information for RTL-invariant objects is reported. Otherwise switching activity information for all design nets, ports, and pins are reported. You can use the `-rtl_saif` option after reading an RTL SAIF file.

The `-missing` option can be used to display the design objects that do not have user-annotated switching activity information.

Using the report_activity Command to Report Switching Activity

The `report_activity` command displays information about the annotated switching activity for the following activity types:

- Annotated
- Default

- Inferred
- Propagated
- Simulated

To display a report showing the annotation on RTL-invariant objects similar to the report generated by the `read_saif -rtl_saif -hierarchy` command, use the `-rtl` option.

Retrieving Switching Activity on a Pin or Net

To get the activity on a pin, port, cell, or net, use the `get_switching_activity` command.

The following example shows the activity for a repeater output pin:

```
dc_shell> get_switching_activity APSHOLD_15/Z -related_clock
(probability = 0.173116, toggle_rate = 0.167328, type = propagated,
 related_clock = Rclk)
```

In this example, the activity was derived by propagating activity along the logic path rather than being annotated from constraints or from a SAIF file. The related clock (Rclk) is the clock used for deriving default activity for this pin.

Removing the Switching Activity Annotation

Switching activity annotation can be removed from all current design objects using the `reset_switching_activity` command. This command removes all the simple and state-and path-dependent switching activity information.

In the following example, an RTL SAIF file is read before a design is compiled with power constraints and then a more accurate gate-level SAIF file is used to generate power reports:

```
read_saif -map_names -input rtl.back.saif -instance_name tb_rtl/i
compile_ultra
...
reset_switching_activity
read_saif -input gate.back.saif -instance_name tb_gate/i
report_power
```

Note that in this example, the SAIF map is already initialized.

You can selectively remove the switching activity information from individual design objects using the following command:

```
dc_shell> set_switching_activity objects
```

Design Objects Without Annotated Switching Activity

The Power Compiler tool needs switching activity information for all design nets and state- and path-dependent information for all design cells and pins to calculate power. Switching activity that is not user annotated is estimated automatically before power is calculated. This is performed in the following stages:

- The user-annotated and default-annotated switching activities are used to derive the simple static probability and toggle rate information for the rest of the design nets.
- The simple switching activity information (user-annotated or estimated) is used to derive the non-annotated state- and path-dependent switching activity.

Default Switching Activity Values

The following types of nets are automatically annotated with switching activity based on the logic of the design:

- Nets driven by constants: A toggle rate value of 0.0 is used. A static probability value of 0.0 is used for logic 0 constants, while a value of 1.0 is used for logic 1 constants.
- Nets driven by clocks: The toggle rate and static probability are derived from the clock waveform.
- Nets driving or driven by buffers: The switching activity of a nonannotated buffer input or output is set to match the switching activity already determined for the other side of the buffer.
- Nets driving or driven by inverters: The switching activity of the inverter input or output is based on the switching activity already determined for the other side of the inverter. The toggle rate is the same and the static probability is complementary.
- Flip-flop outputs: If a flip-flop cell has both Q and QN output ports and only one of the outputs is annotated, then the other output is assigned the same toggle rate and the complementary static probability.
- Inputs and outputs of black box cells: The switching activity cannot be propagated through a black box. Therefore, the default switching activity is annotated on the outputs of a black box.

The default switching activity depends on the value of the `power_default_static_probability` and `power_default_toggle_rate` variables. The default static probability is 0.5. To specify a different value, set the `power_default_static_probability` variable to the required value.

The default toggle rate is 0.1 multiplied by the related clock frequency specified by the `-clock` option of the `set_switching_activity` command. In other words, the net is

assumed to toggle one time every 10 clock periods on average. If no related clock is specified for a net, the clock with the highest frequency is used. To specify a different toggle rate multiplier, set the `power_default_toggle_rate` variable to the required multiplier value (default 0.1).

Propagating the Switching Activity

For nets that are not user-annotated and not assigned switching activity information by default, the tool uses a zero-delay simulator to propagate switching activity from known nets. Random simulation vectors are generated for the user and default annotated nets depending on the annotated toggle rate and static probability values. The zero-delay simulator uses the functionality of the design cells and the random vectors to obtain the switching activity on nonannotated cell outputs.

The number of simulation steps performed by this mechanism depends on the analysis effort option applied to the `report_power` command. User and default annotated switching activity values are never overwritten by values derived by the propagation mechanism.

However, if a design net is not annotated with both toggle rate and static probability values, then the switching activity on this net cannot be used by the propagation mechanism. For such nets, the nonannotated value is estimated by the propagation mechanism.

Deriving the State- and Path-Dependent Switching Activity

If an RTL SAIF file or a gate-level SAIF file without state- and path-dependent switching information is used to annotate the design switching activity, the Power Compiler tool needs to estimate the required state- and path-dependent switching activity information. After obtaining the simple switching activity (from user annotation, or by switching activity propagation), the tool estimates the state-dependent static probability information for every cell, and the state- and path-dependent toggle rate information for every cell pin. This information is obtained from the switching activities of each cell input and output pins. Although the state- and path-dependent estimation mechanism produces accurate power calculations, for best power results, use the gate-level SAIF files with state- and path-dependent information.

6

Performing Power Analysis

The Power Compiler tool analyzes and reports the power consumption of elements in the design.

For information about power analysis, see the following topics:

- [Overview](#)
 - [Identifying Power and Accuracy](#)
 - [Performing Gate-Level Power Analysis](#)
 - [Analyzing Power With Partially Annotated Designs](#)
 - [Power Correlation](#)
 - [Analyzing the Design For Power Analysis](#)
 - [Characterizing a Design for Power](#)
 - [Reporting the Power Attributes of Library Cells](#)
 - [Using Power Derate Factors](#)
 - [Generating Power Reports](#)
-

Overview

The `report_power` command analyzes and reports the power consumption of the various elements of the design. Before you execute this command, you must capture the switching activity, map the design to gates, and annotate the design.

The tool creates power reports for the following design elements:

- Design
- Modules
- Nets
- Cells or groups of cells of a specific type
- Scenarios, for multicorner-multimode designs

The `report_power` command uses a Power Compiler license. When the command is completed, the license is released. If a license is not available, the command terminates with an error message.

To keep the license after completing the `report_power` command, set the following variable:

```
dc_shell> set_app_var power_keep_license_after_power_commands true
```

Identifying Power and Accuracy

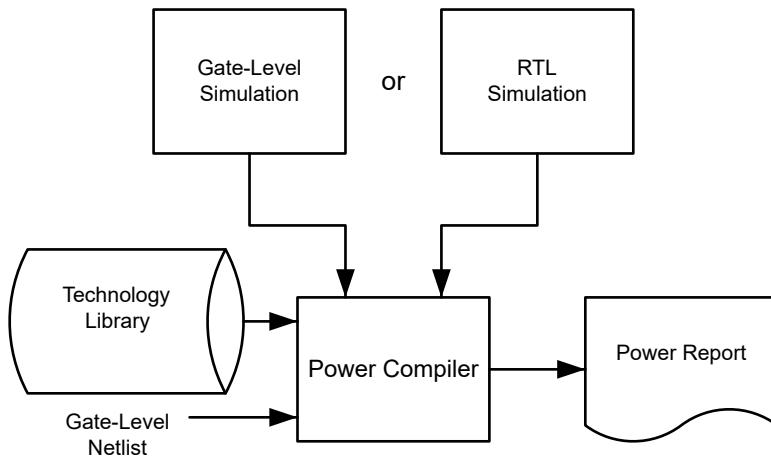
The Power Compiler tool uses several methods to compute the power of the design. The tool considers the type and amount of switching activity annotated on the design and chooses the most accurate method to compute the power. The method used depends on whether you annotate some or all of the elements in your design.

To analyze a gate-level design, the following inputs are required:

- Switching activity
- Logic library
- Gate-level netlist

[Figure 14](#) shows the inputs for the Power Compiler tool.

Figure 14 Inputs to the Power Compiler Tool



For best results, use logic libraries that have been characterized with power information. If the library has only pin capacitance and voltage, but no power information, the Power Compiler tool reports only the switching power of the net. The switching power is a function of the pin capacitance, voltage, and toggle frequency. Use the `report_power` command to display the results.

Factors That Affect the Accuracy of Power Analysis

The following factors can affect the accuracy of power analysis:

- Switching activity annotation
- Delay model
- Correlation
- Complex cells

Switching Activity Annotation

Annotating switching activity relies on the ability to map the names of the synthesis-invariant objects in the RTL source to the equivalent object names in the gate-level netlist. Mapping inconsistencies might cause the SAIF file to be incorrectly or incompletely annotated, which might affect the power analysis results. In turn, the quality of these results affects the results of power optimizations that rely on the annotation. For more information, see [Annotating Switching Activity Using RTL SAIF Files](#).

Clock Frequency Scaling

If a design is synthesized at a frequency that is different from the frequency the simulation is run, the SAIF file generated from the simulation reflects this difference. This causes a mismatch in timing and affects dynamic power analysis.

To correct this problem, the Power Compiler tool allows you to scale the clock frequency, resulting in a more accurate dynamic power analysis.

To enable clock frequency scaling, set the `power_enable_clock_scaling` variable to `true`. Then, run the `set_power_clock_scaling` command to specify clock scaling for power analysis. The command accepts either the `-period` option, which specifies the clock period to which the clock is to be scaled, or the `-ratio` option, which specifies a scaling value to be applied to the toggle rates of pins and nets.

When you use the `set_power_clock_scaling` command, the tool scales only the switching activity applied with the `read_saif` command. The tool does not scale the switching activity applied with the `set_switching_activity` command.

Delay Model

The Power Compiler tool uses a zero-delay model for internal simulation and for propagation of switching activity during power analysis. This zero-delay model assumes that the signal propagates instantly through a gate (with no elapsed time).

The zero-delay model has the advantage of enabling fast and relatively accurate estimation of power dissipation. The zero-delay model does not include the power dissipated due to glitching. If your power analysis must consider glitching, use power

analysis after annotating switching activity from full-timing gate-level simulation. The internal simulation is used only for nodes that do not have user-annotated switching activity.

Switching Activity Correlation

Although the tool propagates switching activity through the design, the logic states of gate inputs might have interdependencies that affect the accuracy of a statistical model. Such interdependency of inputs is called correlation. Correlation affects the accuracy of propagation of toggle rates, which subsequently affects the accuracy of power analysis.

The Power Compiler tool considers correlation within combinational and sequential logic, resulting in more accurate analysis of switching activity for many types of designs. The types of circuits that exhibit high internal correlation are designs with reconvergent fanouts, multipliers, and parity trees. However, the tool has no access to information about correlation external to the design. If correlation exists between the primary inputs of the design, the Power Compiler tool does not recognize the correlation.

The Power Compiler tool considers correlation only within certain memory and CPU thresholds, beyond which correlation is ignored. As the design size increases, the tool might reach its memory limit and cannot fully consider all internal correlation.

As an example of correlation, consider a 4-bit arithmetic logic unit (ALU) that performs five instructions. The data bus is 4 bits wide, and the instruction lines are 3 bits wide. The assumption of uncorrelated inputs is valid for the data bus inputs but fails for the instruction inputs if some instructions are used more often.

If the design has black boxes, such as RAM, ROM, or macro cells, you can annotate switching activity at the outputs of these elements.

Overriding Library Cell Power Characterization

The `set_cell_internal_power` command sets or removes the `power_value` attribute on specified pins. The `power_value` attribute represents the power consumption for a single toggle of the pin. If a cell has at least one such annotated pin, its internal power is calculated by summing the products of the annotated power values and pin toggle rates. If the command is issued without setting the `power_value` attribute, the existing `power_value` attributes are removed from the specified pins. If the `power_value` attribute is specified without a unit, the power unit of the library is used. If the library does not have a defined unit, the tool issues an error message.

Use the `set_cell_internal_power` command to override a cell's library power characterization in situations where that characterization does not apply. This is most common when you manually replace a piece of logic with a single cell and you want the single cell's power consumption to represent the replaced logic. For example, if you replace a clock tree with a single buffer cell, you can set the `power_value` attribute on the output pin of the buffer cell with the value of the power consumption for one clock toggle of the entire clock tree. Although the library cell is characterized, its power

consumption is calculated using the value of the `power_value` attribute set by the `set_cell_internal_power` command.

Performing Gate-Level Power Analysis

After you annotate the design with switching activity, use the `report_power` command to report the power of the design.

To perform power analysis on a partially annotated design, specify a clock with the `create_clock` command before invoking the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute switching activity.

Using the `report_power` Command

The `report_power` command calculates and reports power for a design. When you do not annotate switching activity on the nets, the command performs zero-delay simulation to propagate switching activity for the nets. To compute the switching activity for internal nets, the command uses the switching activity for startpoint nets (if available). The nets that are annotated using the `set_switching_activity` or the `read_saif` command are not overwritten during switching activity propagation.

If you annotate switching activity on all the elements of the design, the Power Compiler tool does not propagate any switching activity through the design. Instead, the tool uses the annotated gate-level switching activity.

Command options modify the report with different sorting modes and with verbose and cumulative options. By default, the tool prints a power summary for the subdesign that contains the specified instance subdesign in the context of the higher-level design.

Power analysis uses net loads during the power calculation. For nets that do not have back-annotated capacitance, the tool estimates the net load from the appropriate wire load model from the logic library.

In topographical mode, the `report_power` command reports the correlated power of the design as the sum of estimated clock tree power and netlist power. For more information, see [Generating Power Reports](#).

The `report_power` command calculates and reports static and dynamic power for the current design. The tool uses user-annotated switching activity to calculate the net switching power, cell internal power, and cell leakage power. When you do not specify any options, the `report_power` command displays the summary of power values only for the current design. If you specify a cell instance, the command reports the summary power values for the specified instance. The command options allow you to specify cells, nets, scenarios, and other conditions.

Some of the options for the `report_power` command are as follows:

`-sort_mode mode`

The `report_power -cell` command uses the `cell_internal_power` value as the default for the `-sort_mode` option. For example, if the logic library does not have any internal power modeling for leaf cells, the `report_power -cell -nworst 10` command reports only the worst ten cells.

When you use the `report_power -net` command, the `net_switching_power` value is the default for the sorting mode. If both the `-net` and `-cell` options are specified and a sort mode is explicitly specified, the selected sort mode is used for both the cell and net reports. If both options are specified without an explicit sort mode, the sort mode is by total dynamic power.

`-histogram`

This option prints a histogram-style report with the number of nets in each power range. Use the `-exclude_leq` and `-exclude_geq` options, to exclude data values less than or greater than specified threshold values. The `-histogram` option must be used with one or both of the `-net` or `-cell` options.

`-groups list_of_cell_group`

This option reports power for the specified power groups. Without this option, the report uses the predefined groups listed in [Table 7](#). The `-groups` option is mutually exclusive with the `-net`, `-hierarchy`, `-levels`, and `-only` options.

[Table 7](#) lists the power groups and the cell types that belong to the group in descending order of priority.

`-hierarchy`

This option enables you to view internal, switching, and leakage power consumed in your design hierarchy, on a block-by-block basis. The hierarchical levels of the design are indicated by indentations.

`-levels level_value`

Use this option only with the `-hierarchy` option. This option enables you to limit the depth of the hierarchy tree displayed in the report. The `level_value` setting should be an integer number greater than or equal to 1. For example, to see the power results for all blocks up to 2 levels from the top, use the following command:

```
dc_shell> report_power -hierarchy -levels 2
```

-scenarios

This option reports the power details for the specified list of scenarios for a multimode design. Inactive scenarios are not reported. When this option is not used, only the current scenario is reported.

Table 7 Groups and Their Cell Types in Descending Order of Priority

Group	Cell types belonging to the group
io_pad	Cells defined in the pad_cell group in the library
memory	Cells defined in the memory group in the library
black_box	Cells that do not have any functional description in the library
clock_network	Cells in the clock network, excluding the io_pad cells
register	Latches and flip-flops driven by the clock network, excluding the io_pad and black_box cells
sequential	Latches and flip-flops clocked by signals that are not in the clock network
combinational	Cells that have a functional description and are not sequential cells

Using the report_power_calculation Command

The Power Compiler tool calculates both dynamic and leakage power. The dynamic power consists of internal power on pins and switching power on nets. Both internal and leakage power might be state-dependent.

The `report_power` command provides a comprehensive report. In addition, the `report_power_calculation` command shows how the reported power numbers are derived from inputs such as the library, simulation data, the netlist, and parasitics. The `report_power_calculation` command does not work on libraries that have built-in security to protect the power table numbers. This restriction does not apply to switching power.

Analyzing Power With Partially Annotated Designs

If you invoke power analysis without annotating any switching activity, the Power Compiler tool uses the following defaults for the primary inputs of your design:

- $P_1 = 0.5$ (the signal is in the 1 state 50 percent of the time)

P_1 is the probability that input P is at logic state 1. For definitions of static probability, P_1 and toggle rate (TR), see [Types of Switching Activity to Annotate](#).

- $TR = 0.1 * f_{clk}$ (the signal switches one time, every 10 clock cycles)

f_{clk} is the frequency of the input signal's related clock, as defined by the `set_switching_activity` command. You can specify the related clock explicitly with its clock name or implicitly with the asterisk symbol (*). In the latter case, the tool infers a related clock automatically. If the input port does not have a related clock, the tool uses the fastest clock in the design.

The defaults for static probability and toggle rate are often reasonable choices for data bus lines. However, the defaults might be unacceptable for some signals, such as a reset or a test-enable signal.

If you do not annotate toggle information for primary inputs, these inputs assume the default toggle value. If the input or logic connected to this input is heavily loaded, the results might be significantly different from what you expect.

To change the defaults for switching activity and static probability, set the following variables:

- `power_default_static_probability`

This variable sets the static probability to use for unannotated nets. The default is 0.5.

- `power_default_toggle_rate`

This variable sets the toggle rate for unannotated nets. The default is 0.1.

- `power_default_toggle_rate_type`

The default is `fastest_clock`, which causes the tool to calculate the default toggle rate by multiplying the fastest clock frequency by the default toggle rate. Set this variable to `absolute` when the design object does not have a specified related clock. In this case, the tool uses the value of the `power_default_toggle_rate` variable.

The following command sets the default static probability to 0.3:

```
dc_shell> set_app_var power_default_static_probability 0.3
```

The following command sets the default toggle rate to 0.4 times the toggle rate of the highest-frequency clock:

```
dc_shell> set_app_var power_default_toggle_rate 0.4
```

Power Correlation

Power correlation refers to the relationship between two power calculations: power after logic synthesis and power after place and route. Power after place and route is the final power, and you might want to know this number early in the process so you can take corrective action if the number exceeds your limits. Power correlation is supported only in Design Compiler topographical mode.

In dc_shell, the power reported after logic synthesis is often significantly different from the final power, and is therefore not a good predictor for final power. This differential is caused by three factors:

- Logic synthesis uses wire load models.
- High-fanout nets are not synthesized.
- Clock trees do not exist in the design at the time of synthesis.

Performing logic synthesis within the Design Compiler topographical domain shell addresses the first two factors because this shell uses a virtual layout, not wire load models, and high-fanout nets are synthesized automatically.

To eliminate the differential caused by the third factor, perform clock-tree estimation within dc_shell-topo.

To improve correlation in cases with abnormal floor plans, use the physical constraints extracted from the floor plan.

Performing Power Correlation

Correlated power refers to the design power that is added to the estimated clock-tree power after logic synthesis in Design Compiler topographical mode. Correlated power is also referred to as estimated total power.

To calculate the correlated power, enable the power prediction feature by using the `set_power_prediction` command.

Specify the clock tree references by using the `-ct_references` option to perform clock-tree estimation, which improves the correlation results.

When the power prediction feature is enabled, the `report_power` command reports the correlated power after the design has been mapped to technology-specific cells. When

the power prediction feature is disabled, the `report_power` command reports only the total power, static power, and dynamic power, without considering the estimated clock-tree power.

The power prediction setting is saved with the design if the design is saved in the .ddc (Synopsys logical database format) binary file format.

Power Correlation Script

The following example script correlates power after you have set up the design environment and applied synthesis constraints:

```
read_verilog
set_power_prediction
compile_ultra
report_power
write -format ddc -output design.ddc
```

In Design Compiler topographical mode, the `report_power` command reports estimated total power, which includes the clock-tree contributions for internal, net-switching, and leakage power.

Analyzing the Design For Power Analysis

Follow these steps to get quick results from gate-level power analysis:

1. Create a SAIF file.

This step requires RTL simulation. For information, see [Generating SAIF Files](#).

2. Compile the design to gates, using various suitable options.
3. Annotate switching activity on primary inputs and other synthesis-invariant elements of the gate-level design.

For information about using SAIF files from RTL simulation to annotate switching activity, see [Generating SAIF Files](#).

4. Use the `report_power` command to analyze your design's power.

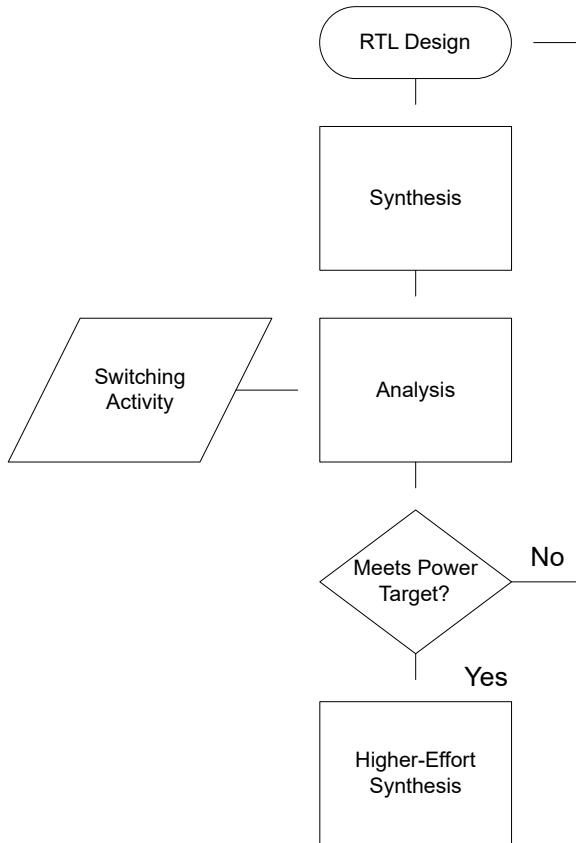
The tool uses an internal zero-delay simulation to propagate switching activity through nonannotated elements of the design.

5. Repeat steps 1 through 4 for other architectures and coding styles.

Quick gate-level power analysis enables you to see the results of changes in your RTL design.

[Figure 15](#) shows the steps that are followed in design exploration using the Power Compiler tool.

Figure 15 Analyzing the Design for Power Analysis



After you refine an RTL design within the iterative loop of design exploration, the design is ready for higher-effort synthesis.

Characterizing a Design for Power

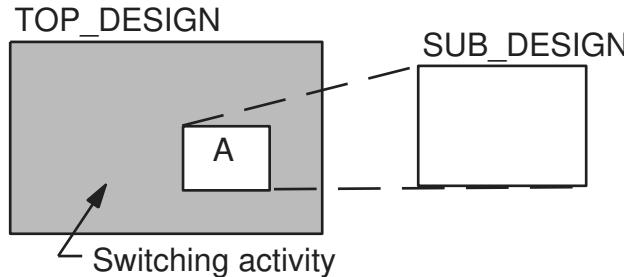
The `-power` option of the `characterize` command is useful in power analysis and optimization. This option characterizes annotated or propagated switching activity from the instance of a subdesign to the nets of the subdesign referenced by the instance. There must be a one-to-one correspondence between the nets in the instance and the nets in the referenced subdesign.

As shown in [Figure 16](#), consider a design hierarchy in which A is a design instance of `SUB_DESIGN` in `TOP_DESIGN`. Instance A references `SUB_DESIGN`. When you invoke

power analysis on TOP_DESIGN, the switching activity propagates on nets that are not already user-annotated.

```
dc_shell> report_power top_design
```

Figure 16 Switching Activity for TOP_DESIGN



The switching activity can be propagated from primary inputs and synthesis-invariant elements. In this example, user-annotated on individual design elements using `set_switching_activity` commands, or both.

However, if you set the current instance to A and characterize for power, the `characterize` command writes the switching activity of instance A onto SUB_DESIGN.

```
dc_shell> current_design TOP_DESIGN
dc_shell> characterize A -power
```

After characterization, you can report the power of SUB_DESIGN with the newly characterized switching activity. If you have a Power Compiler license, you can compile SUB_DESIGN with the newly characterized switching activity.

The `-power` option of the `characterize` command requires a one-to-one correspondence between the nets of the referenced SUB_DESIGN cell and its instance A. If you compile the subdesign before characterizing instance A or make any changes that alter the nets or names of nets, the one-to-one net correspondence is lost and the `characterize` command fails.

After compiling a subdesign and before reanalyzing or compiling TOP_DESIGN, be sure to relink the designs.

Before recompiling the subdesign, follow some or all of the following steps:

- Relink the designs.
- Generate new switching activity for changed designs.
- Annotate or propagate new switching activity on designs.
- Characterize before reanalyzing or recompiling the subdesign.

Reporting the Power Attributes of Library Cells

Use the `report_lib -power` command to report which library cells have power characterization and what type of characterization exists on each library cell. The `report_lib -power` command reports the following information for each cell:

- Leakage power attribute
- Internal power attribute
- Attribute for separate rise and fall power
- Attribute for average rise and fall power
- Toggling pin specified by the internal power table
- When conditions (for state-dependent power)
- The `related_pin` or `related_input` for path-dependent power

Using Power Derate Factors

Power derate factors are scalar multiplicative factors that affect power analysis results in power reporting and power optimization.

Use the `set_power_derate` command to set a power derate value on a list of design objects. You can set derate factors on library cells, leaf cells, hierarchical cells, power groups, or the entire design. By default, the Power Compiler tool uses a factor of 1.0. The order of precedence is as follows, in decreasing order of precedence:

- Leaf cells (either from a setting directly on the cell or from a setting on a power group)
- Hierarchical cells
- Cells without explicitly specified factors but inheriting factors from the hierarchical parent cells
- Library cells
- Entire design

You can use options of the `set_power_derate` command to set factors specifically for leakage power, internal power, or switching power, or to specify a scenario.

If you set a power derate factor more than one time on a specific object, the last value overrides any previously specified values.

To list the power derate factors for a specific object, use the `get_power_derate` command. This command accepts other Tcl commands as arguments, such as the `get_cells` command.

To report power derate factors for a list of objects, use the `report_power_derate` command.

To reset power derate factors to a value of 1.0, use the `reset_power_derate` command.

Generating Power Reports

This section contains examples of reports generated with the `report_power` command.

The `report_power` command in topographical mode reports the correlated power, consisting of estimated clock tree power and netlist power. If the tool cannot perform clock tree estimation, the Power Compiler tool issues a warning that the clock tree estimation could not be performed.

Examples of power reports are as follows:

- [Power Report Summary](#)
- [Net Power Report](#)
- [Cell Power Report](#)
- [Group Report](#)
- [Wire and Pin Switching Power Report](#)
- [Hierarchical Power Report](#)
- [Power Report for Block Abstractions](#)
- [Register Clock-Pin Internal Power Report](#)

Power Report Summary

[Example 14](#) shows a power report summary.

Example 14 Summary Report of the report_power Command

```
dc shell> report_power -analysis_effort high -verbose
*****
Report : power
          -analysis_effort high
          -verbose
Design : DESIGN_1
::
*****
Library(s) Used:
          slow (File: slow.db)

Operating Conditions:
Wire Loading Model Mode: Inactive

Global Operating Voltage = 1.62
Power-specific unit information :
          Voltage Unit = 1V
          Capacitance Units = 1.000000pf
          Time Units = 1ns
          Dynamic Power Units = 1mW      (derived from V,C,T units)
          Leakage Power Units = 1nW

Cell Internal Power Breakdown
-----
Combinational      =   3.0975 mW    (10%)
Sequential         =  22.3222 mW    (72%)
Other              =   0.0000 mW    (0%)

Combinational Count = 13470
Sequential Count   = 2382
Other Count        = 0
Information: Reporting correlated power. (PWR-620)

Cell Internal Power = 27.2572 mW    (76%)
Net Switching Power = 8.6208 mW    (24%)
-----
Total Dynamic Power = 35.8779 mW    (100%)
Cell Leakage Power = 2.6586 uW
```

Chapter 6: Performing Power Analysis

Generating Power Reports

Power Breakdown					
Cell	Cell Internal Power (mW)	Driven Net Switching Power (mW)	Tot Power (mW)	Dynamic Power (% Cell/Tot)	Cell Leakage Power (nW)
Netlist Power	110.6964	117.6087	2.283e+02	(48%)	2.370e+04
Estimated Clock Tree Power	N/A	N/A	N/A	N/A	N/A
Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%) Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	5.6978	7.5225e-04	1.0238e+04	5.9088	(0.00%)
black_box	0.0000	0.7594	0.0000	0.7594	(0.33%)
clock_network	0.2417	1.9775	162.6490	2.2194	{ 0.97%}
register	60.5457	21.1285	3.8973e+03	81.7167	(35.79%)
sequential	6.2925e-03	1.5848e-02	2.7066	2.2143e-02	{ 0.01%}
combinational	43.9830	93.7028	9.4943e+03	137.6785	(60.30%)
Total	110.4746 mW	117.5849 mW	2.3705e+04 nW	228.3049 mW	

Net Power Report

Example 15 shows a net power report sorted by the net switching power and filtered to display only the five nets with the worst switching power.

Example 15 Net Power Report, Sorting, and Display Options

```
dc_shell> report_power -net -flat -sort_mode net_switching_power -nworst 5
*****
Report: power
  -net
  -nworst 5
  -flat
  -sort_mode net_switching_power
Design: DESIGN_1
...
*****
Library(s) Used:
  power_lib.db (File: /remote/libraries/power_lib.db)

Operating Conditions: slow Library: slow
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000pf
  Time Units = 1ns
  Dynamic Power Units = 1mW      (derived from V,C,T units)
  Leakage Power Units = 1nW
```

```

Attributes
-----
a - Switching activity information annotated on net
d - Default switching activity information on net

          Total      Static      Toggle      Switching
Net        Net Load    Prob.       Rate        Power      Attrs
-----
U_TAP_DBG_U_DBG_net5051      0.463     0.374     0.1968     0.1195
U_CORE/U_A7S_pencadd         0.248     0.374     0.1968     0.0641
U_CORE/U_A7S_dataio_net5298  0.247     0.374     0.1968     0.0637
U_CORE/U_MUL8_net5450        0.232     0.374     0.1968     0.0599
U_CORE/U_AREG_net5593         0.194     0.374     0.1968     0.0501
-----
Total (5 nets)                           357.2614 uW

```

Cell Power Report

[Example 16](#) displays a cell power report containing the cumulative cell power report. The cells are sorted by cumulative fanout power values, only the top five are reported.

Example 16 Cell Power Report Containing Cumulative Cell Power

```

dc_shell> report_power -cell -analysis_effort low \
-sort_mode cell_internal_power
*****
Report : power
  -cell
  -analysis_effort low
  -sort mode cell_internal_power
Design : DESIGN_3
...
*****
Library(s) Used:
  slow (File: slow.db)

Operating Conditions: slow Library: slow
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000pf
  Time Units = 1ns
  Dynamic Power Units = 1mW      (derived from V,C,T units)
  Leakage Power Units = 1nW

Information: Reporting correlated power. (PWR-620)

```

```
Attributes
-----
h - Hierarchical cell

Cell          Cell      Driven Net   Tot Dynamic   Cell
Internal     Power    Switching Power    (% Cell/Tot) Leakage
Cell          Power    Power
-----
```

Cell	Internal Power	Driven Power	Tot Power	(% Cell/Tot)	Cell Leakage	Attrs
CLOCK_TREE_EST	1.8375	3.1021	4.940	(37%)	9.9144	
U_CORE	21.7118	N/A	N/A	(N/A)	2226.6487	h
U_TAP_DBG_U_DB	0.0123	N/A	N/A	(N/A)	1.4392	h
U_TAP_DBG_U_SCAN1	0.0106	2.472e-04	1.09e-02	(98%)	0.1458	
...						
Totals (2474 cells)	27.368mW	N/A	N/A (N/A)		2.658uW	

Group Report

[Example 17](#) shows the report generated by the `report_power` command when you use the `-groups` option.

Example 17 Cell Report for Groups

```
dc_shell> report_power -groups "io_pad memory combinational"

*****
Report : power
        -analysis_effort low
Design : RISC_CORE
...
*****

Library(s) Used:
t65pwc_ccs (File: t65pwc_pg.db)
t65pwcd9_ccs (File: t65pwcd9_pg.db)
t65pwcd72_ccs (File: t65pwcd72_pg.db)
t65pwc0d_ccs (File: t65pwc0d_pg.db)

Global Operating Voltage = 1.08
Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW      (derived from V,C,T units)
Leakage Power Units = 1nW
```

Chapter 6: Performing Power Analysis

Generating Power Reports

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	6.0868e-02	7.1321e-02	777.9665	0.1330	(100.00%)	
Total	6.0868e-02 mW	7.1321e-02 mW	777.9665 nW	0.1330 mW		
	Net Switching Power = 71.3213 uW (53.95%)					
	Cell Internal Power = 60.8676 uW (46.05%)					
	<hr/>					
	Total Dynamic Power = 132.1889 uW (100%)					
	Cell Leakage Power = 777.9665 nW					
1						

Wire and Pin Switching Power Report

If you set the `power_report_separate_switching_power` variable to `true`, the group report contains separate columns for wire switching power and pin switching power instead of the total switching power, as shown in [Figure 17](#).

```
dc_shell> set_app_var power_report_separate_switching_power true
dc_shell> report_power
```

Figure 17 Wire and Switching Power Report

Power Group	Internal Power	Wire Switching Power	Pin Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	5.8978	5.0723e-04	2.4463e-04	1.0238e+04	5.9088	(2.59%)	
black_box	0.0000	0.7168	4.2653e-02	0.0000	0.7594	(0.33%)	
clock_network	0.2417	0.7667	1.2108	162.6490	2.2194	(0.97%)	
register	60.5457	17.8876	3.2407	3.8073e+03	81.7167	(35.79%)	
sequential	6.2925e-03	1.4331e-02	1.5165e-03	2.7066	2.2143e-02	(0.01%)	
combinational	43.9830	61.4337	32.2694	9.4943e+03	137.6785	(60.30%)	
Total	110.6746 mW	80.8196 mW	36.7653 mW	2.3705e+04 nW	228.3049 mW		
1							

Hierarchical Power Report

[Example 18](#) shows the results of the `report_power` command using the `-hierarchy` option. This option shows the internal, switching, and leakage power consumed in the design hierarchy on a block-by-block basis.

Example 18 Hierarchical Power Report

```
dc_shell> report_power -hierarchy
*****
Report : power
        -hierarchy
        -analysis_effort low
Design : DESIGN_4
...
*****
Library(s) Used:
    slow (File: slow.db)

Operating Conditions: slow Library: slow
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW      (derived from V,C,T units)
    Leakage Power Units = 1nW

Information: Reporting correlated power. (PWR-620)
-----

```

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
A7S_top	8.683	27.368	2.66e+03	36.054	100.0
CLOCK_TREE_EST	3.102	1.837	9.914	4.940	13.7
U_CORE (A7S_core)	4.318	21.712	2.23e+03	26.032	72.2

Power Report for Block Abstractions

When you use the `create_block_abstraction` command, the power information is saved as attributes on the block abstractions. If you annotate the switching activity, either by using the SAIF file or the `set_switching_activity` command, the switching activity is used to calculate the power information of the block abstractions, as shown in the following example:

```
Current design is 'test11_0
create_block_abstraction
internal power = 62.508907
leakage power = 227754.921875
net switching power = 17.909983, dyn_unit = 1mW, leak_unit = 1nW
```

By default, the `report_power` command reports the total power of the block, using the power information saved as attributes on the model, as shown in the following example:

```
...
Information: u_test11_3 test11_3 is block, internal power = 61.429829 mW
Information: u_test11_2 test11_2 is block, internal power = 60.608749 mW
Information: u_test11_1 test11_1 is block, internal power = 63.291779 mW
Information: u_test11_0 test11_0 is block, internal power = 62.508907 mW
Information: u_test11_core is block, leakage power = 122.281441 uW
Information: u_test11_3 test11_3 is block, leakage power = 222.416214 uW
Information: u_test11_2 test11_2 is block, leakage power = 224.137466 uW
Information: u_test11_1 test11_1 is block, leakage power = 228.666733 uW
Information: u_test11_0 test11_0 is block, leakage power = 227.754929 uW
Information: u_test11_3 test11_3 is block, net switching power =
17.291439mW
Information: u_test11_2 test11_2 is block, net switching power =
17.869442 mW
Information: u_test11_1 test11_1 is block, net switching power =
18.588411 mW
Information: u_test11_0 test11_0 is block, net switching power =
17.909983 mW
```

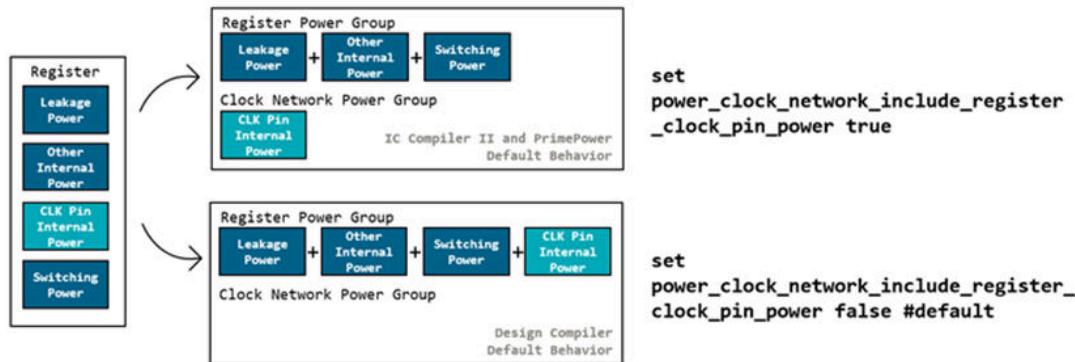
Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%) Attrs
<hr/>					
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)
black_box	271.0595	85.7838	1.0253e+06	357.8686	(18.53%)
clock_network	0.9882	1.5616e+03	117.9128	1.5625e+03	(80.89%)
register	3.9093	0.3665	2.3101e+04	4.2989	(0.22%)
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)
combinational	1.2514	5.6117	5.5759e+04	6.9189	(0.36%)
<hr/>					
Total	277.2083 mW	1.6533e+03 mW	1.1042e+06 nW	1.9316e+03 mW	
1					

For more information about block abstractions, see the *Design Compiler User Guide*.

Register Clock-Pin Internal Power Report

The `report_power` command includes register clock-pin internal power in the clock network power group when the `power_clock_network_include_register_clock_pin_power` variable is set to true.

Enabling this variable aligns the Design Compiler power report with the default IC Compiler II power report:



The following report shows the tool output when the `power_clock_network_include_register_clock_pin_power` variable is set to `false` (the default):

Internal Power Group Attrs	Switching Power	Leakage Power	Total Power	Power (%)
<hr/>				
io_pad	0.0000	0.0000	0.0000	0.0000 (0.00%)
memory	0.0000	0.0000	0.0000	0.0000 (0.00%)
black_box	0.0000	0.0000	0.0000	0.0000 (0.00%)
clock_network	514.1058	7.8403	2.1341e+09	2.6560e+03 (20.35%)
register	882.8593	6.0360	4.6494e+09	5.5383e+03 (42.44%)
sequential	0.0000	0.0000	0.0000	0.0000 (0.00%)
combinational	447.1125	13.9511	4.3956e+09	4.8567e+03 (37.21%)
<hr/>				
Total	1.8441e+03 uW	27.8273 uW	1.1179e+10 pW	1.3051e+04 uW

The following report shows the tool output when the `power_clock_network_include_register_clock_pin_power` variable is set to `true`:

Attributes

i - Including register clock pin internal power

Power Group Attrs	Internal Power	Switching Power	Leakage Power	Total Power	(%)
<hr/>					
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)
clock_network	1.6048e+03	7.8403	2.1341e+09	2.6560e+03	(20.35%)
register	-2.0782e+02	6.0360	4.6494e+09	5.5383e+03	(42.44%)
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)

Chapter 6: Performing Power Analysis
Generating Power Reports

```
combinational      447.1125    13.9511  4.3956e+09    4.8567e+03 (37.21%)
-----
Total           1.8441e+03 uW   27.8273 uW  1.1179e+10 pW   1.3051e+04 uW
```

Part 3: Power Reduction

The following topics provide information about the power reduction techniques available in the Power Compiler tool:

- [Clock Gating](#)
- [Self-Gating](#)
- [Power Optimization](#)
- [Multivoltage Design Concepts](#)
- [UPF Multivoltage Design Implementation](#)
- [Library Setup for Power Optimization](#)
- [Power Optimization in Multicorner-Multimode Designs](#)

7

Clock Gating

Power optimization at higher levels of abstraction has a significant impact on reduction of power in the final gate-level design. Clock gating is an important technique for reducing the power consumption of a design.

For information about clock gating in the Power Compiler tool, see the following topics:

- [Introduction to Clock Gating](#)
- [Using Clock-Gating Conditions](#)
- [Inserting Clock Gates](#)
- [Clock Gating Flows](#)
- [Specifying Clock-Gate Latency](#)
- [Calculating the Clock Tree Delay From Clock-Gating Cell to Registers](#)
- [Specifying Setup and Hold](#)
- [Clock-Gating Styles](#)
- [Modifying the Clock-Gating Structure](#)
- [Integrated Clock-Gating Cells](#)
- [Clock-Gating Naming Conventions](#)
- [Keeping Clock-Gating Information in a Structural Netlist](#)
- [Replacing Clock-Gating Cells](#)
- [Inserting Clock Gates With Safety Registers](#)
- [Clock-Gate Optimization Performed During Compilation](#)
- [Performing Clock-Gating on DesignWare Components](#)
- [Reporting Clock Gates](#)

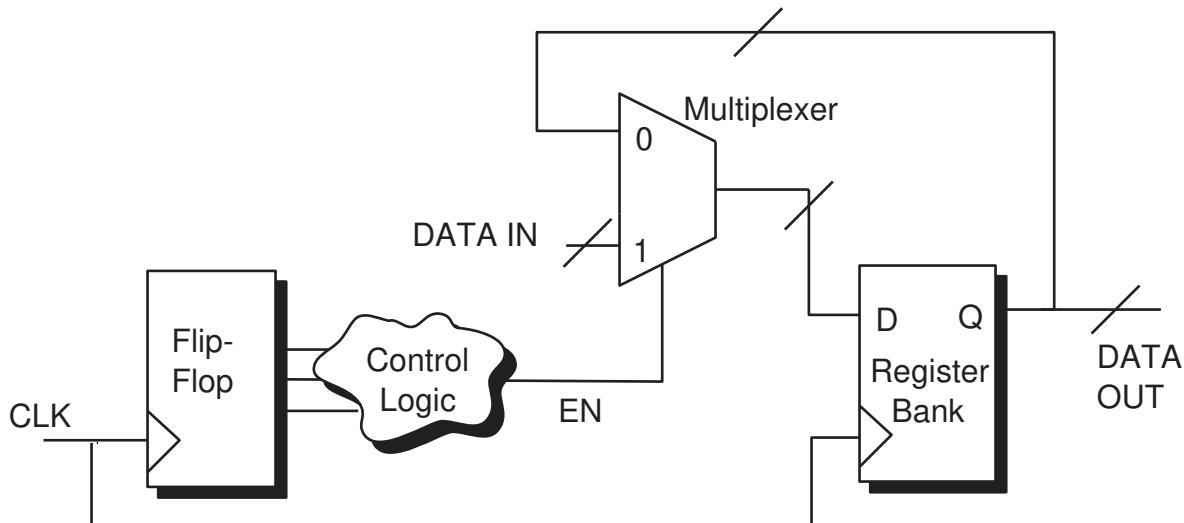
Introduction to Clock Gating

Clock gating applies to synchronous load-enable registers, which are flip-flops that share the same clock and synchronous control signals. Synchronous control signals include synchronous load-enable, synchronous set, synchronous reset, and synchronous toggle.

Synchronous load-enable registers are represented by a register with feedback loop which maintains the same logic value through multiple cycles. Clock gating applied to synchronous load enable registers reduces the power needed when reloading the register banks.

[Figure 18](#) shows a simple register bank implementation using a multiplexer with feedback loop.

Figure 18 Synchronous Load-Enable Register With Multiplexer



When the synchronous load enable signal (EN) is at logic state 0, the register bank is disabled. In this state, the circuit uses the multiplexer to feed the Q output of each storage element in the register bank back to the D input. When the EN signal is at logic state 1, the register is enabled, allowing new values to load at the D input.

Such feedback loops can unnecessarily use power. For example, if the same value is reloaded in the register throughout multiple clock cycles (EN equals 0), the register bank and its clock net consume power while values in the register bank do not change. The multiplexer also consumes power.

Clock gating eliminates the feedback net and multiplexer shown in [Figure 18](#) by inserting a gate in the clock net of the register.

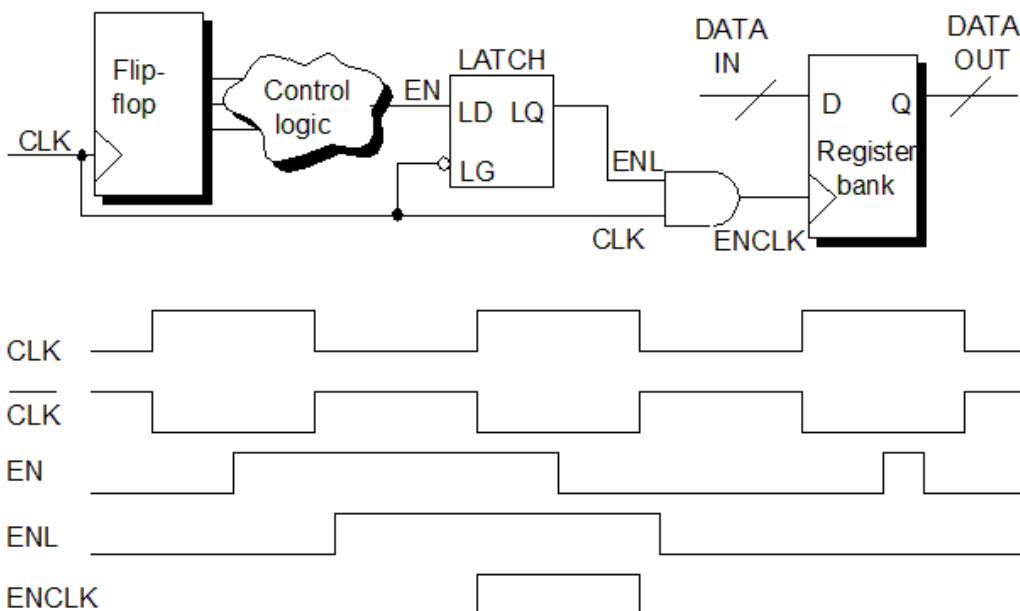
Note:

While applying the clock-gating techniques, the tool considers generated clocks similar to defined clocks.

The clock-gating cell selectively prevents clock edges, thus preventing the gated-clock signal from clocking the gated register.

[Figure 19](#) shows a latch-based clock-gating cell and the waveforms of the signals are shown with respect to the clock signal, CLK.

Figure 19 Latch-Based Clock Gating



The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it derives from the EN signal on the multiplexer shown in [Figure 18](#). The register bank is triggered by the rising edge of the ENCLK signal.

The latch prevents glitches on the EN signal from propagating to the register's clock pin. When the CLK input of the 2-input AND gate is at logic state 1, any glitching of the EN signal could, without the latch, propagate and corrupt the register clock signal. The latch eliminates this possibility because it blocks signal changes when the clock is at logic state 1.

In latch-based clock gating, the AND gate blocks unnecessary clock pulses by maintaining the clock signal's value after the trailing edge. For example, for flip-flops inferred by HDL constructs of rising-edge clocks, the clock gate forces the gated clock to 0 after the falling edge of the clock.

By controlling the clock signal for the register bank, you can eliminate the need for reloading the same value in the register through multiple clock cycles. Clock gating inserts clock-gating circuitry into the register bank's clock network, creating the control to eliminate unnecessary register activity.

Clock gating reduces clock network power dissipation, relaxes datapath timing, and reduces routing congestion by eliminating feedback multiplexer loops. For designs that have large register banks, clock gating can save power and area by reducing the number of gates in the design. However, for smaller register banks, the overhead of adding logic to the clock tree might not compare favorably to the power saved by eliminating a few feedback nets and multiplexers.

Using Clock-Gating Conditions

Before gating the clock signal of a register, the Power Compiler tool checks to see if certain clock-gating conditions are satisfied. The tool inserts a clock gate only if all the clock-gating conditions are satisfied:

- The circuit demonstrates synchronous load-enable functionality.
- The circuit satisfies the setup condition.
- The register bank or group of register banks satisfies the minimum number of bits you specify with the `set_clock_gating_style -minimum_bitwidth` command. The default minimum bitwidth is 3.

After clock gating is complete, the status of clock-gating conditions for gated and ungated register banks appears in the clock-gating report. For information about the clock-gating report, see [Reporting Clock Gates](#).

Clock-Gating Conditions

The register must satisfy the following conditions for the Power Compiler tool to gate the clock signal of the registers:

- Enable condition

If the register bank's synchronous load-enable signal is a constant logic 1, reducible to logic 1, or logic 0, the condition is `false` and the circuit is not gated. If the synchronous load-enable signal is not a constant logic 1 or 0, the condition is `true` and the setup condition is checked. The enable condition is the first condition that the tool checks.

- Setup condition

This condition applies to latch-free clock gating only. The enable signal must come from a register that uses the same clock as the register being gated. The setup condition is checked only if the register satisfies the enable condition.

- Width condition

The width condition is the minimum number of bits for gating registers or groups of registers with equivalent enable signals. The default is 3. You can set the width condition by using the `-minimum_bitwidth` option of the `set_clock_gating_style` command. The width condition is checked only if the register satisfies the enable condition and the setup condition.

Enable Condition

The enable condition of a register or clock gate is a combinational function of nets in the design. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock is passed to the registers in the fanout of the clock gate. The tool uses the enable condition of the registers for clock-gate insertion.

Enable conditions are represented by Boolean expressions for nets. For example:

```
module TEST (en1, en2, en3, in, clk, dataout);
    input en1, en2, en3, clk;
    input [5:0] in;
    output [5:0] dataout;
    reg [5:0] dataout;

    wire enable;

    assign enable = (en1 | en3) & en2;

    always @ (posedge clk) begin
        if( enable )
            dataout <= in;
        else
            dataout <= dataout;
    end
endmodule
```

In this example, the enable condition for the register bank `dataout_reg*` can be expressed as `en1 en2 + en3 en2`.

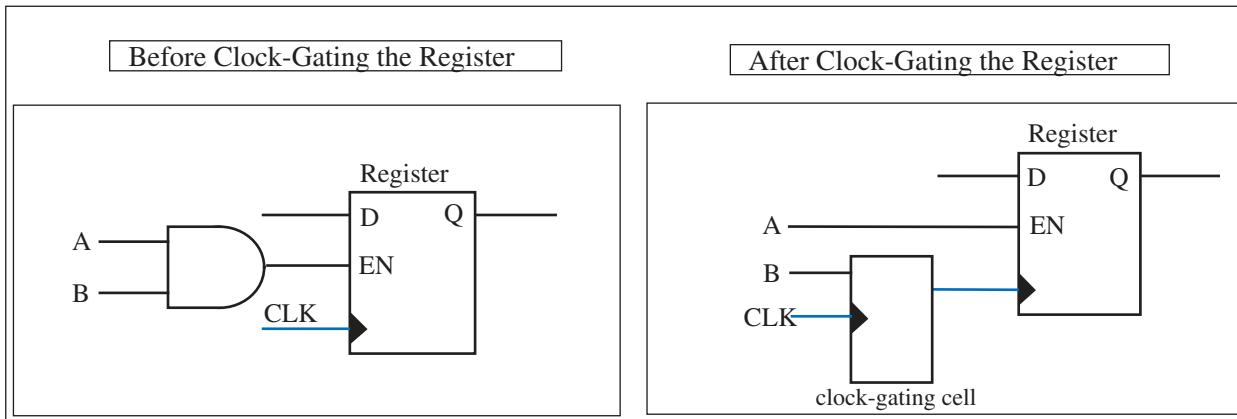
Excluding Specific Signals From the Enable Condition

You can specify signals to be excluded from the enable condition of clock gating. For example, you can specify a late arriving signal to be excluded from the enable condition, to prevent it from becoming a critical path.

The exclusion of a signal from the enable condition depends on the Boolean expression of the enable condition. In [Figure 20](#), the enable signal of the register is an AND function of inputs A and B. To exclude the signal A from the computation of the enable condition of

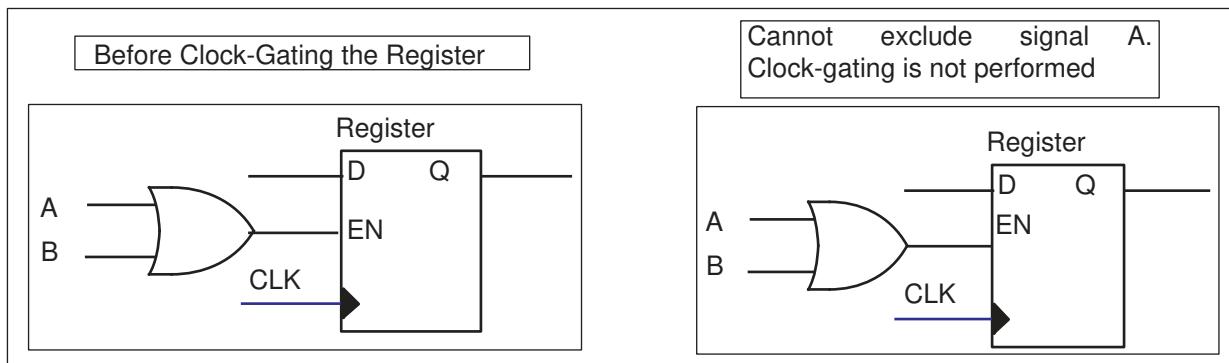
the clock gate, the tool connects input A to the enable pin of the register and input B to the enable pin of the clock-gating cell.

Figure 20 Excluding Signal A From Clock Gating



In [Figure 21](#), the enable signal of the register is an OR function of inputs A and B. The tool does not exclude input A from clock-gating because it is not feasible to gate the register when one of the inputs is at logic 1.

Figure 21 Cannot Exclude Signal A. Clock-Gating is not Performed for the Register



You can exclude a signal from the enable expression of a register, if removing the signal from the enable expression does not result in a constant 0 or a constant 1.

Use the `set_clock_gating_enable -exclude` command to specify the objects whose signals are to be excluded from the enable condition. You can specify objects such as primary input ports, output pins of sequential cells, sequential cells, black box cells, and macro cells to be excluded from the enable condition.

The specified signal is excluded from clock-gating when you run the `compile_ultra -gate_clock` command or any subsequent `compile_ultra -incremental`

`-gate_clock` command. Using the exclusion criteria, the `compile_ultra` command checks the feasibility of excluding the specified signals from clock gating. If exclusion is feasible, the command modifies the enable expression of the clock-gating signal and the enable signal of the register.

If it is not feasible to exclude the specified signal from clock-gating, the tool does not clock-gate the register. If the register is already clock-gated using the signal that is specified for exclusion, the tool removes the clock-gating cell. The `set_clock_gating_objects -force_include` command or the `power_cg_all_registers` variable setting does not prevent the tool from removing the clock-gating cell.

Use the `set_clock_gating_enable -undo` command to remove the exclusion constraint.

The `report_clock_gating -ungated` command reports the details of registers that are not clock-gated, including the reasons for not gating them.

The `write_script` command writes out the exclusion constraint that you specify. You can source the file written by the `write_script` command, in the Design Compiler tool to support ASCII flow or in third-party tools.

Setup Condition

To perform clock gating, the Power Compiler tool requires that the enable signal of the register bank is synchronous with its clock. This is the setup condition.

For latch-based or integrated clock gating, the tool can insert clock gating irrespective of the enable signal's and the clock's clock domains. If the enable signal and the register bank reside in different clock domains, you must ensure that the two clock domains are synchronous and that the setup and hold times for the clock-gating cell meet the timing requirements.

For latch-free clock gating, if any of the following characteristics exist, the setup condition is false and the register bank is not gated:

- If the register bank and its controlling logic (including flip-flops) belong to different clock domains, the setup condition is false.
- If the register bank and its controlling logic (including flip-flops) are driven by different edges of the same clock signals, the setup condition is false.

- If the controlling logic is driven by a combinational path from the input port, the setup condition is false, unless:
 - For primary input ports, you specified a clock with the `set_input_delay` command.
 - You specified `power_cg_derive_related_clock true`, which enables clock propagation of the related clocks from parent hierarchies for inputs on subdesigns. The default is `false`.

These two special cases specify that an input port is synchronous with a given clock; therefore, the setup condition is true.

Specify `power_cg_ignore_setup_condition true` to ignore the setup condition for latch-free clock gating.

Enabling or Disabling Clock Gating on Design Objects

You can enable or disable clock gating on certain design objects by overriding all necessary conditions set by the clock-gating style. The `set_clock_gating_objects` command specifies the design objects on which clock gating should be enabled or disabled during the `compile_ultra -gate_clock` command. If you use the `compile_ultra -gate_clock` command, you must run the `uniquify` command before inserting the clock gates.

The following example includes and excludes the specified registers from clock gating:

```
dc_shell> set_clock_gating_objects \
           -force_include ADDER/out1_reg[*] \
           -exclude ADDER/out2_reg[*]
```

The following example excludes all registers in the subdesign ADDER, except the `out1_reg` bank. The `out1_reg` bank is clock gated according to the specified clock-gating style:

```
dc_shell> set_clock_gating_objects \
           -exclude ADDER \
           -include ADDER/out1_reg[*]
```

The following example sets and then removes the inclusion and exclusion criteria specified by the `-include` and `-exclude` options:

```
dc_shell> set_clock_gating_objects \
           -include ADDER/out1_reg[*] \
           -exclude ADDER/out2_reg[*]

dc_shell> set_clock_gating_objects \
           -undo {ADDER/out1_reg[*] ADDER/out2_reg[*]}
```

Inserting Clock Gates

The Power Compiler tool inserts clock-gating cells to your design if you compile your design using the `-gate_clock` option of the `compile` or `compile_ultra` command.

Using the `compile_ultra -gate_clock` Command

During the compilation process, the Power Compiler tool can insert clock-gates to your design if you use the `-gate_clock` option of the `compile_ultra` command. With the `-gate_clock` option, the `compile_ultra` command can perform clock-gate insertion on the gate-level netlist, RTL netlist, as well as GTECH netlist. By default, when you use the `-gate_clock` option, the tool inserts clock gates only in the same level of hierarchy as the registers gated by the clock gate. For the tool to perform clock gating across the design hierarchy, set the `compile_clock_gating_through_hierarchy` variable to true. For more details about hierarchical clock gating, see [Hierarchical Clock Gating](#).

The `compile_ultra -gate_clock` command can also perform clock gating on DesignWare components. For more details, see [Performing Clock-Gating on DesignWare Components](#).

In Design Compiler topographical mode, when you perform clock gating by using the `compile_ultra -incremental -gate_clock` command, the tool performs incremental placement and gate-level clock gating.

Clock-Gate Insertion in Multivoltage Designs

In a multivoltage design, the different hierarchies of the design can have different operating condition definition and use different target library subsets. While inserting clock-gating cells in a multivoltage design, the Power Compiler tool chooses the appropriate library cells based on the specified clock-gating style as well as the operating conditions that match the operating conditions of the hierarchical cell of the design. If you do not specify a clock-gating style, the tool chooses a suitable clock-gating style. If the tool does not find a library cell that suites the clock-gating style and the operating conditions, a clock-gating cell is not inserted and a warning message is issued.

For more information about clock-gating style, see [Selecting Clock-Gating Styles](#).

Clock Gating Flows

The clock-gating flows supported by the tool are described in the following sections:

- [Inserting Clock Gates in the RTL Design](#)
- [Inserting Clock Gates in Gate-Level Design](#)

Inserting Clock Gates in the RTL Design

To insert clock gating logic in your RTL design and to synthesize the design with the clock-gating logic, follow these steps:

1. Read the RTL design.
2. Use the `compile_ultra -gate_clock` command to compile your design.

During the compilation process the clock gate is inserted on the registers qualified for clock-gating. By default, during clock-gate insertion, the `compile_ultra` command uses the default settings of the `set_clock_gating_style` command, and also honors the setup, hold, and other constraints specified in the logic libraries. To override the setup and hold values specified in the library, use the `set_clock_gating_style` command before compiling your design.

The `compile_ultra` command uses the default settings of the `set_clock_gating_style` command during clock-gate insertion. The default settings of the `set_clock_gating_style` command is suitable for most designs. For more information about the default clock-gating style, see [Default Clock-Gating Style](#).

3. If you are using testability in your design, use the `insert_dft` command to connect the `scan_enable` and the `test_mode` ports or pins of the integrated clock-gating cells.
4. Use the `report_clock_gating` command to report the registers and the clock-gating cells in the design. Use the `report_power` command to get information of the dynamic power used by the design after the clock-gate insertion.

In the following example, clock gating is performed during the compilation process. The default settings of the `set_clock_gating_style` command are used during the clock-gate insertion. The `-scan` option of the `compile_ultra` command enables the examination of your design for scan insertion.

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name CLK
dc_shell> compile_ultra -gate_clock -scan
dc_shell> insert_dft
dc_shell> report_clock_gating
dc_shell> report_power
```

Inserting Clock Gates in Gate-Level Design

To insert clock gating logic in your gate-level netlist and to resynthesize the design with the clock gating logic, follow these steps:

1. Read the gate-level netlist.
2. Use the `compile_ultra -gate_clock -incremental` command to compile your design.

During the compilation process, clock-gating cells are inserted on the registers qualified for clock gating. During this process, by default, the `compile_ultra` command

- Reads the setup and hold constraints that are specified in the logic libraries.
- Propagates these constraints up the hierarchy.

To override the setup and hold values specified in the library, use the `set_clock_gating_style` command before compiling your design. Use the `compile_ultra -gate_clock` command to perform clock-gate insertion on DesignWare components. For more information about clock-gate insertion on DesignWare components, see [Performing Clock-Gating on DesignWare Components](#).

The `compile_ultra -gate_clock` command uses the default settings of the `set_clock_gating_style` command, during the clock-gate insertion. The default settings of the `set_clock_gating_style` command are suitable for most designs. For more information about the default clock-gating style, see [Default Clock-Gating Style](#).

3. If you are using testability in your design, use the `insert_dft` command to connect the `scan_enable` and `test_mode` ports or pins of the integrated clock-gating cells.
4. Use the `report_clock_gating` command to report the registers and the clock gating cells in the design. Use the `report_power` command to get details of the dynamic power used by the design after the clock-gate insertion.

In the following example, clock gating is implemented in the design during the compilation process. The default settings of the `set_clock_gating_style` command are used during the clock-gate insertion.

```
dc_shell> read_ddc design.ddc
dc_shell> compile_ultra -incremental -gate_clock -scan
dc_shell> insert_dft
dc_shell> report_clock_gating
dc_shell> report_power
```

Specifying Clock-Gate Latency

During synthesis, Design Compiler assumes that the clocks are ideal. An ideal clock incurs no delay through the clock network. This assumption is made because real clock-network delays are not known until after clock tree synthesis. In reality clocks are not ideal and there is a non-zero delay through the clock network. For designs with clock gating, the clock-network delay at the registers is different from the clock-network delay at the clock-gating cell. This difference in the clock-network delay at the registers and at the clock-gating cell results in tighter constraints for the setup condition at the enable input of the clock-gating cell.

For Design Compiler to account for the clock network delays during the timing calculation, specify the clock network latency using either the `set_clock_latency` or the `set_clock_gate_latency` command. The `set_clock_gate_latency` command can be used for both, gate-level and RTL designs.

For more information, see the following topics:

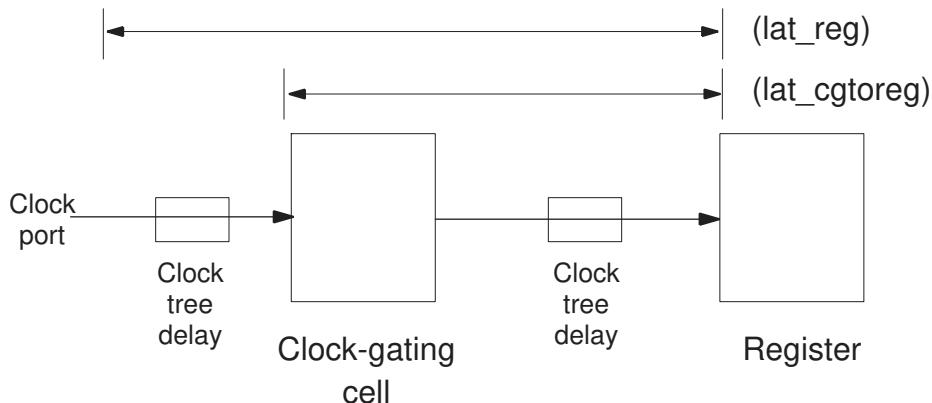
- [The `set_clock_latency` Command](#)
 - [The `set_clock_gate_latency` Command](#)
 - [Applying Clock-Gate Latency](#)
 - [Resetting Clock-Gate Latency](#)
 - [Comparison of the Clock-Gate Latency Specification Commands](#)
-

The `set_clock_latency` Command

Use the `set_clock_latency` command to specify clock network latency for specific clock-gating cells.

As illustrated in [Figure 22](#), `lat_cgtoreg` is the estimated delay from the clock pin of the clock-gating cell to the clock pin of the gated register and `lat_reg` is the estimated clock-network latency to the clock pins of the registers without clock gating.

Figure 22 Clock Latency With Clock-Gating Design



For all clock pins of registers (gated or ungated) in the design that are driven by a specific clock, use the `lat_reg` value for the `set_clock_latency` command. For clock pins of all the clock-gating cells, use the difference between the `lat_reg` and `lat_cgtoreg` values for the `set_clock_latency` command. Because the purpose of setting the latency values is to account for the different clock-network delays between the registers and the clock-gating cell, it is important to get a reasonably accurate value of the difference. The absolute values used are less important, unless you are using these values to account for clock-network delay issues not related to clock gating.

The `set_clock_gate_latency` Command

When you use the `compile_ultra -gate_clock` command, clock gates are inserted during the compilation process. To specify the clock network latency before the clock-gating cells are inserted by the tool, use the `set_clock_gate_latency` command. This command lets you specify the clock network latency for the clock-gating cells as a function of the clock domain, clock-gating stage, and the fanout of the clock-gating cell. The latency that you specify is annotated on the clock-gating cells when they are inserted by the `compile_ultra -gate_clock` command. You can manually annotate the latency values on the existing clock-gating cells in your design using the `apply_clock_gate_latency` command. For more details, see [Applying Clock-Gate Latency](#).

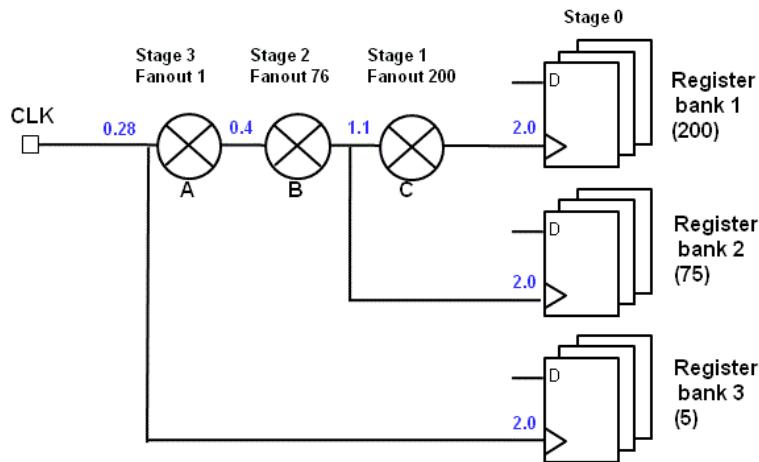
Figure 23 shows the definitions for the clock-gate stages and the fanouts.

The clock-gating cell C drives 200 registers. So the fanout of the cell C is 200. Because C drives registers, and not other clock gating cells, the clock gating stage for the cell C is 1.

The clock-gating cell B drives a set of 75 registers and a clock gating cell C. So the fanout of the clock-gating cells B is 76. The clock-gating stage for the cell B is 2; clock gating stage of cell C plus 1.

Similarly, the clock-gating stage of cell A is 3 and the fanout is 1. The clock-gating stage of all the registers is stage 0.

Figure 23 Clock-Gating Stages and Fanouts



The following example script shows how to specify the latency values for the various clock gate stages and fanouts using the `set_clock_gate_latency` command for the design shown in [Figure 23](#).

```
set_clock_gate_latency -clock CLK -stage 0 \
    -fanout_latency {1-inf 2.0}
set_clock_gate_latency -clock CLK -stage 1 \
    -fanout_latency {1-30 1.8, 31-100 1.5, 101-inf 1.1}
set_clock_gate_latency -clock CLK -stage 2 \
    -fanout_latency {1-5 0.9, 6-20 0.5, 21-100 0.4, 101-inf 0.3}
set_clock_gate_latency -clock CLK -stage 3 \
    -fanout_latency {1-10 0.28, 11-inf 0.11}
```

To specify clock latency value for the clock-gated registers, use the `-stage` option with a value 0. Because you are specifying the latency value for the clock gated registers, the value for the `-fanout_latency` option should be 1-infinity, as shown in the following example:

```
set_clock_gate_latency -clock CLK -stage 0 \
    -fanout_latency {1-inf 1.0}
```

Applying Clock-Gate Latency

The clock latency specified using the `set_clock_gate_latency` command is annotated on the registers during the `compile_ultra -gate_clock` command when the clock-gating cells are inserted. However, if you modify the latency values on the clock gates after the compilation, you must manually apply the latency values on the existing clock-gating cells using the `apply_clock_gate_latency` command.

Note:

After you modify the clock-gate latency using the `set_clock_gate_latency` command, if you compile your design using the `compile_ultra` or `compile_ultra -incremental` command, it is not necessary to use the `apply_clock_gate_latency` command to apply the latency values. The tool annotates the specified value during compilation.

Resetting Clock-Gate Latency

To remove the clock latency information specified on the clock-gating cells, use the `reset_clock_gate_latency` command. This command removes the clock latency values on the specified clocks. If you do not specify the clock, the clock latency values on all the clock-gating cells are removed. This command removes the clock latency on the specified clocks, irrespective of whether the latency values were specified using the `set_clock_latency` or `set_clock_gate_latency` command.

Comparison of the Clock-Gate Latency Specification Commands

Table 8 compares various commands that you can use to specify the clock-gate latency.

Table 8 Comparison of Clock-Gating Latency Specification Commands

<code>set_clock_gate_latency</code>	<code>set_clock_gating style -setup -hold</code>	<code>set_clock_gating check</code>	<code>set_clock_latency</code>
Recommended for use with the <code>compile_ultra -gate_clock</code> command	Default settings are recommended for most designs. Use this command only if the default settings are not suitable for your design	To specify the clock-gate latency on existing clock-gating cells.	To modify clock-gate latency on existing clock-gating cells.
To specify clock-gate latency before the clock gates are inserted by the <code>compile_ultra -gate_clock</code> command		Specification is on the instance. So, specify on each clock-gating cell.	Specification is on the instance. So, specify on each clock-gating cell

Table 8 Comparison of Clock-Gating Latency Specification Commands (Continued)

<code>set_clock_gate_latency</code>	<code>set_clock_gating style -setup -hold</code>	<code>set_clock_gating check</code>	<code>set_clock_latency</code>
To modify the clock-gate latency settings on existing clock-gating cells	To specify the setup and hold values before the clock gates are inserted	Specification overrides the setup and hold values in the library	The latency setting specifies the clock arrival time at the clock-gating cell
The latency setting specifies the clock arrival time at the clock-gating cell	The specification overrides the setup and hold values defined in the library		
Specification is based on clock domain, clock-gating state and fanout	Generic settings for all the clock gates in the design		

Calculating the Clock Tree Delay From Clock-Gating Cell to Registers

If your clock tree synthesis tool does not insert buffers after the clock-gating cell, then the total delay between the clock-gating cell and the registers is equal to the delay of the clock-gating cell (clock pin to clock out signal) plus the wire delay between the clock-gating cell and the registers. If your clock tree synthesis tool inserts buffers after the clock-gating cell, add an estimate of the clock-network delay to the total delay between the clock-gating cell and the registers. You can use an estimate based on the fanout of the clock-gating cell and the driving capacity of typical clock tree buffers or use data from earlier designs.

For most designs, the enable signal arrives early and is not affected by clock-network delay issues. For late arriving enable signals, it is advised to be conservative (high value) in the selection of the delay from the clock-gating cell to the registers. A low value might mean an enable signal which is unable to meet arrival time constraints at the clock-gating cell after the clock tree is inserted. However, a high value might over constrain the enable signal leading to higher area or power and ensures that the enable signal arrives in time at the clock-gating cell.

After placement and clock tree synthesis, you can back-annotate delay information by using the `set_propagated_clock` command for Design Compiler to use real delay data for the clock-network delay. For more information, see the *Design Compiler User Guide*.

Specifying Setup and Hold

During insertion of clock gates, the setup and hold time that you specify defines the margins within which the enable signal (EN) must operate to maintain the integrity of the gated-clock signal.

The setup and hold values for the integrated clock-gating cell are specified in the logic library. The values specified in the logic library are honored by the `compile_ultra -gate_clock` command during the clock-gate insertion. However, you can override these values in the following ways:

- Specifying the `-setup` and `-hold` options of the `set_clock_gating_style` command. By doing so, all the clock gates in the design should have the setup and hold time that you specify.
- For the clock-gating cells already existing in your design, use the `set_clock_gating_check` command to specify your setup and hold time.

You use the `report_timing -to` command to the enable pin of the clock-gating cell to verify that the new values are correct.

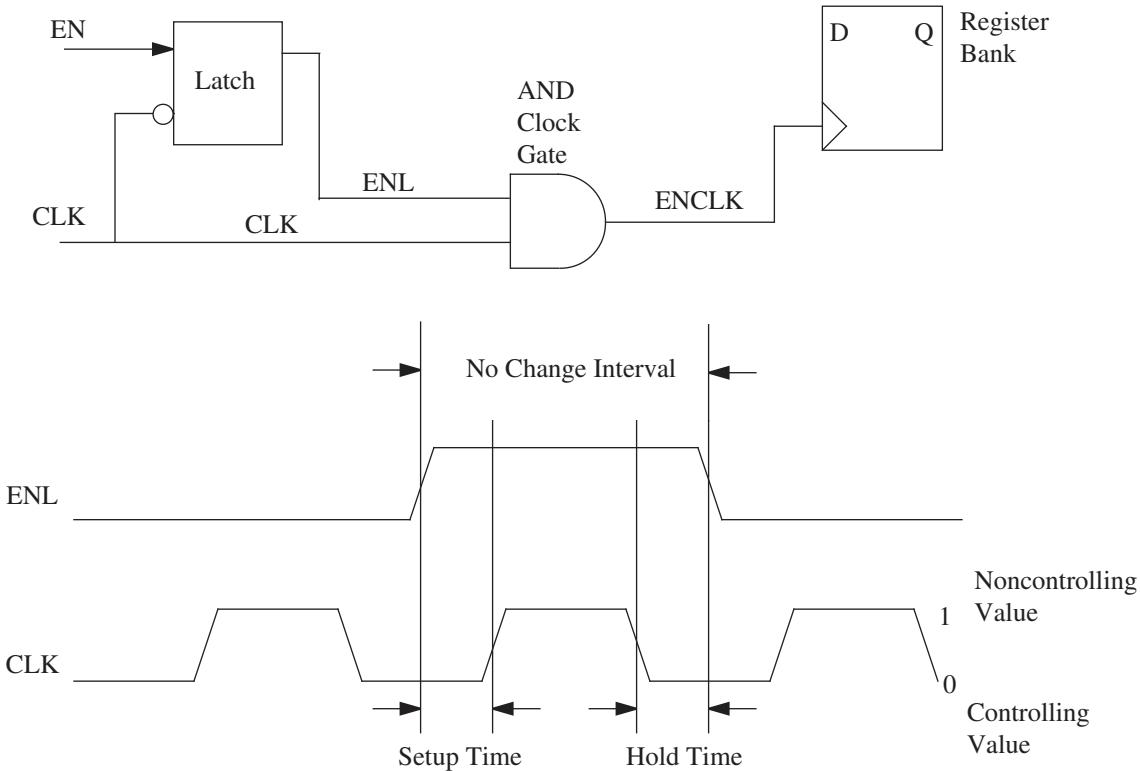
The following example uses the `set_clock_gating_style` command to specify the setup and hold values:

```
set_clock_gating_style \
    -max_fanout 16 -positive_edge_logic integrated \
    -setup 6 -hold 2
compile_ultra -gate_clock
# to validate the user-specified setup or hold time for
# integrated clock gating
report_timing -to clk_gate_out_top_reg/EN
report_timing -to clk_gate_out_top_reg_1/EN
```

The clock gate must not alter the waveform of the clock, other than turning the clock signal on and off. If the enable signal operates outside the chosen margins specified by the `-setup` and `-hold` options, the resulting gated signal might be clipped or corrupted.

[Figure 24](#) and [Figure 25](#) show the relationship of setup and hold time to a clock waveform. [Figure 24](#) shows the relationship with an AND gate as the clock-gating element. [Figure 25](#) shows the relationship with an OR gate as the clock-gating element.

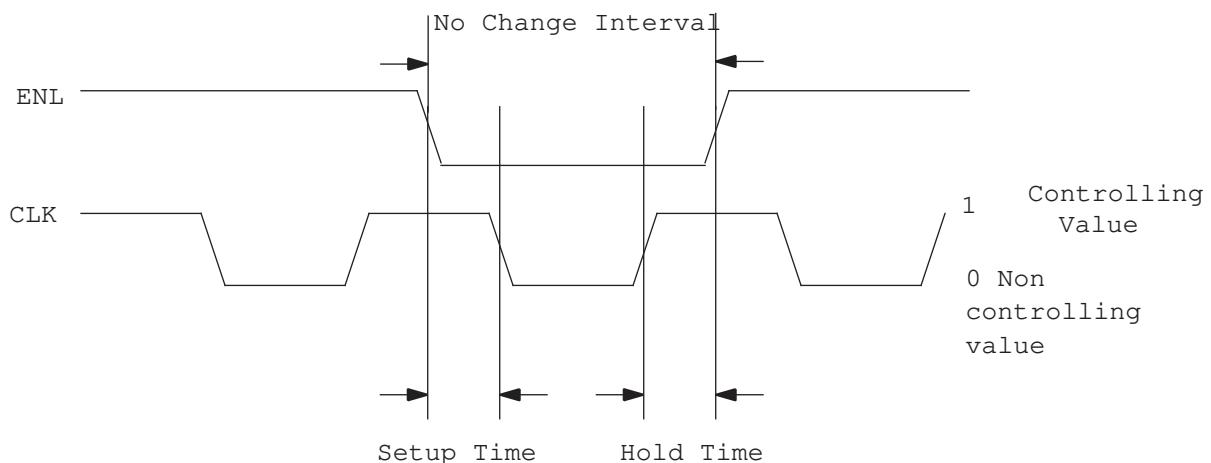
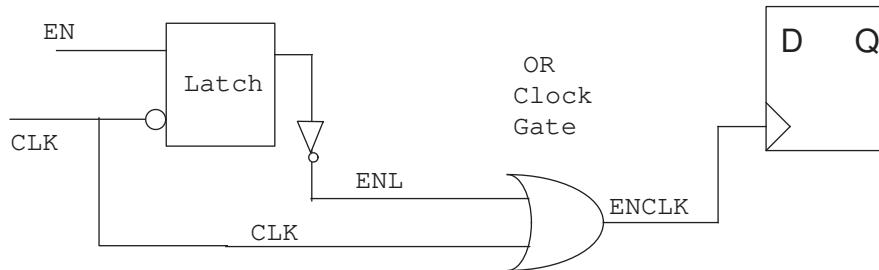
Figure 24 Setup and Hold Time for an AND Clock Gate



Enable after latch (ENL) signal must be stable before the clock input (CLK) makes a transition to a non-controlling value. The hold time ensures that the ENL is stable for the time you specify after the CLK returns to a controlling value. The setup and hold time ensures that the ENL signal is stable for the entire time that the CLK signal has a non-controlling value, which prevents clipping or glitching of the ENCLK clock signal.

You might need to add latency by using the `set_clock_latency` command. Use this command for non-clock-gating registers. For more information, see [Specifying Clock-Gate Latency](#) and the Design Compiler documentation.

Figure 25 Setup and Hold Time for an OR Clock Gate



Note:

When using PrimeTime for static timing analysis, use the `-setup` and `-hold` options of the `set_clock_gating_check` command to change the setup and hold values for the gating check. PrimeTime performs clock-gating checks on all gated clocks using 0.0 as the default for setup and hold.

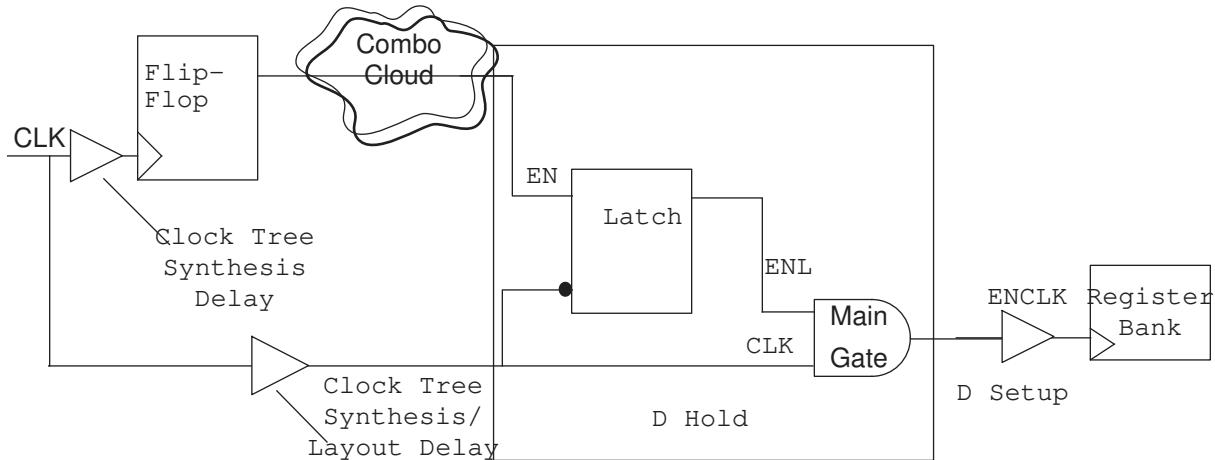
Predicting the Impact of Clock Tree Synthesis

Clock tree synthesis can affect your choice of setup and hold time. However, during clock gating, the clock tree does not exist yet: clock tree synthesis normally occurs much later in the design process than clock gating. Without the clock tree, it can be difficult to precisely predict the impact of clock tree synthesis on the delay of the design. For this reason, you might find it necessary to alter your setup and hold time after clock tree synthesis.

Choosing a Value for Setup

Choose a value for the setup time that estimates the impact of the delay of the clock tree from the clock gate to the gated register bank. In latch-based clock gating, the value for setup mimics the delay of the clock tree from the clock gate to the register bank.

Figure 26 Setup and Hold Time for Clock Tree Synthesis



Your setup time constrains the ENL signal so that after gate-level synthesis, there is still enough timing slack for the addition of the clock tree during clock tree synthesis.

In latch-free clock gating, the value for setup must consider the clock signal duty cycle. For example, in a design using a latch-free clock gate:

1. Estimate the delay of the clock tree between the clock gate and the gated register (as you would for the latch-based clock gate).
2. From the value you estimate in step 1, add the worst-case (largest possible) clock low time (typically half of the clock-cycle time).

This is appropriate for flip-flops triggered on the clock's rising edge. For flip-flops triggered on the clock's falling edge, add the worst-case (largest possible) clock high time.

If the setup time is too small, the ENL signal must be reoptimized after back-annotation from layout to fit the tighter timing constraints. If the value of `-setup` is too large, the ENL signal is too constrained and optimization of combinational control logic results in larger area and power to satisfy the tighter timing constraints.

Choosing a Value for Hold

Latch-based clock gating has the timing requirement that the transition of the ENL signal occur at the 2-input clock gate after the trailing edge (rising edge for falling-edge flip-flop) of the clock signal. This timing requirement is usually satisfied because the addition of a latch because of clock gating, increases the delay on the ENL signal. In rare cases, however, after clock tree synthesis and physical design, additional delay in the clock signal might cause the CLK signal to arrive after the ENL signal. This is due to clock skew between the clock signal driving the clock-gating latch and the clock signal driving the 2-input gate.

If you expect this timing violation, you can set the `-hold` value during clock gating to artificially define a hold constraint on the ENL signal. Gate-level synthesis adds buffers in the ENL signal if they are necessary to satisfy your hold constraint.

If the value of `-hold` is too small, you might have to reoptimize the ENL signal after back-annotation from layout to ensure the integrity of the gated clock signal. If the value of `-hold` is too large, you might find a chain of buffers delaying the ENL signal before the clock gate.

Clock-Gating Styles

The Power Compiler tool inserts the clock-gating cells in the design based on the styles that you specify. When you do not specify a clock-gating style, the tool uses a set of predefined styles for the clock gates. The default settings of the `set_clock_gating_style` command are suitable for most designs.

The following sections discuss in detail, the default clock-gating style and using specific clock-gating styles:

- [Default Clock-Gating Style](#)
- [Selecting Clock-Gating Styles](#)

The `compile_ultra -gate_clock` command prevents clock-gate insertion when the target library does not contain cells for the defined clock-gating style and operating condition and issues the PWR-763 information message. You must redefine the clock-gating style or the operating conditions, based on the clock-gating cells available in the target library.

Default Clock-Gating Style

When you specify the `set_clock_gating_style` command, the default style used by the tool is different from the default style used when you do not specify the command.

When you specify the `set_clock_gating_style` command with only a few options, the tool uses the default specified in [Table 9](#) for the unspecified option.

Table 9 Defaults for Clock-Gating Style

Parameter	Default used when the <code>set_clock_gating_style</code> command is specified without any option
Sequential cell	Latch
Minimum bit-width	3
Setup constraint	Library value
Hold constraint	Library value
Positive edge logic	and
Negative edge logic	or
Control point	none
Control signal	<code>scan_enable</code>
Observation point	false
Observation logic depth	5
Maximum fanout	infinite
Number of stages	1
No sharing	false

When you specify the `set_clock_gating_style` command multiple times, the last setting overrides the previous settings.

When you do not specify a clock-gating style, the Power Compiler tool derives a default clock-gating style based on the specified libraries. The cells are chosen from the library in the following decreasing order of priority:

- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic integrated \ -control_point before -control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic integrated -control_point after \ -control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic integrated -control_point before \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic integrated -control_point after \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic integrated`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic or -control_point after \ -control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic or -control_point before \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic or -control_point after \ -control_signal test_mode`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic or -control_point after \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \ -negative_edge_logic or`
- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic integrated -control_point before \ -control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic integrated -control_point after \ -control_signal scan_enable`

- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic integrated -control_point before \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic integrated -control_point after \ -control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic integrated`
- `set_clock_gating_style -positive_edge_logic and \ -negative_edge_logic or`

The following example inserts clock-gating cells by choosing a suitable default style:

```
read_verilog low.v
compile_ultra -gate_clock
report_clock_gating -style
compile_ultra -incremental
```

Selecting Clock-Gating Styles

Use the `set_clock_gating_style` command to select the clock-gating style. The `compile_ultra -gate_clock` command uses the specified clock-gating style to insert the clock-gating cells. The default settings of the `set_clock_gating_style` command is suitable for most designs. If the default setting does not suit your design, use the `set_clock_gating_style` command to change the default setting.

The clock-gating style that you specify is applied to the entire design. You can also apply the clock-gating style only to specific power domains or hierarchical cells of the design. For more information about specifying clock-gating styles on specific instances, see [Using Instance-Specific Clock-Gating Styles](#).

The following topics describe how to use the `set_clock_gating_style` command:

- [Choosing Gating Logic](#)
- [Choosing an Integrated Clock-Gating Cell](#)
- [Choosing a Configuration for Discrete Gating Logic](#)
- [Choosing a Simple Gating Cell by Name](#)
- [Choosing a Simple Gating Cell and Library by Name](#)
- [Designating Simple Cells Exclusively for Clock Gating](#)
- [Choosing a Specific Latch and Library](#)
- [Choosing a Latch-Free Style](#)

- Improving Testability
- Connecting the Test Ports Throughout the Hierarchy
- Using Instance-Specific Clock-Gating Styles

Choosing Gating Logic

The following options of the `set_clock_gating_style` command specify the type of clock-gating logic or clock-gating cell used for implementing clock gating:

```
-positive_edge_logic [gate_list]
-negative_edge_logic [gate_list]
```

You can specify a configuration of 1-input and 2-input gates (simple gating cells) to use for clock gating, or an integrated clock-gating cell already defined in the target library.

An integrated cell is a dedicated clock-gating cell that combines all of the simple gating logic of a clock gate into one fully characterized cell, possibly with additional logic such as multiple enable inputs, active-low enabling logic, or an inverted gated clock output.

Choosing an Integrated Clock-Gating Cell

You can use the `-positive_edge_logic` and `-negative_edge_logic` options of the `set_clock_gating_style` command to specify the integrated clock-gating cell for clock gating:

```
-positive_edge_logic [gate_list]
-negative_edge_logic [gate_list]
```

The first cell found that meets the clock-gating requirements is used and possibly sized up or down to meet the design rule violations if the library has integrated cells of different sizes. Use the `power_do_not_size_icg_cells` variable to prevent this behavior.

Choosing an Integrated Cell by Functionality

When selecting an integrated cell by functionality, clock gating searches your library for integrated cells having the correct value of the `clock_gating_integrated_cell` attribute.

Use the `set_clock_gating_style` command to specify the functionality of the integrated cell you want clock gating to look for.

The Power Compiler tool uses the first integrated cell it finds in your library that matches the requirements you specify with the `set_clock_gating_style` command. For example, if you enter

```
set_clock_gating_style -negative_edge_logic {integrated}
```

The tool uses the first integrated cell it finds in your logic library that has the `clock_gating_integrated_cell` attribute, as follows:

```
clock_gating_integrated_cell : "latch_nededge";
```

When you do not specify the sequential option, the tool uses the default latch-based gating. For more information about attributes for integrated cells and library syntax, see the Library Compiler documentation.

Choosing an Integrated Cell by Name

Choose an integrated cell by name when you require a specific integrated cell or if you have more than one integrated cell with the same `clock_gating_integrated_cell` attribute. For example,

```
set_clock_gating_style -positive_edge_logic {integrated:my_cell}
```

In this example, clock gating chooses an integrated cell called `my_cell` from the logic library. For more information about attributes for integrated cells and Library syntax, see the Library Compiler documentation.

Specifying a Subset of Integrated Clock Gates

Use the `set_dont_use -power` command to limit clock gate insertion to a specific set of integrated clock gate cells from one or more libraries. This command guarantees that the specified cells is not used for power optimization. For example,

```
set_dont_use -power [get_lib_cells a1.db/icg_a1_*]
set_dont_use -power [get_lib_cells b2.db/icg_b2_*]
set_dont_use -power [get_lib_cells c3.db/icg_c3_*]
set_clock_gating_style -positive_edge_logic {integrated}
compile_ultra -gate_clock
```

In this example, the `set_clock_gating_style` command directs the `compile_ultra -gate_clock` command to use all integrated cells except the cells that have the `dont_use` attribute.

Using Setup and Hold for Integrated Cells

Setup and hold constraints are built into the integrated cell when you create it with Library Compiler, but you can override the values by using either the `set_clock_gating_style` command or the `set_clock_gating_check` command.

If you provide `-setup` and `-hold` values on the command line when using an integrated cell, the values are overridden.

The following example uses an integrated cell to gate rising-edge-triggered registers and uses simple cells to gate falling-edge-triggered registers using latch-free style.

```
set_clock_gating_style -sequential_cell none
-setup setup_value
```

```
-hold hold_value
-positive_edge_logic {integrated}
-negative_edge_logic {inv nor buf}
```

The `setup_value` and `hold_value` apply not only to the integrated cell, but also to the clock gate built for falling-edge-triggered registers using simple cells (INV, NOR, and BUF gates in this example). For more information about integrated clock-gating cells and timing, see the *Library Compiler User Guide*.

Choosing a Configuration for Discrete Gating Logic

The `-positive_edge_logic` and `-negative_edge_logic` options can have up to three string parameters that specify the type of clock gating logic:

- The type of 2-input clock gate (AND, NAND, OR, NOR)
- An inverter or buffer on the clock network before the 2-input clock gate
- An inverter or buffer on the clock network after the 2-input clock gate

The positions of the string parameters determine whether clock gating places a buffer or inverter before or after the 2-input clock gate. For example, if the value of `-positive_edge_logic` is `{and buf}`, clock gating uses an AND gate and places a buffer in the fanout from the AND gate. If the value is `{inv nor}`, clock gating uses a NOR gate and places an inverter in the fanin of the NOR gate. Both of these examples result in AND functionality of the clock gate.

The type of logic that is appropriate for gating your circuit depends on whether the gated register banks are inferred by rising-edge or falling-edge clocks in your HDL code and whether you use latch-based or latch-free clock gating.

If you use latch-free clock gating, you must specify both the `-positive_edge_logic` and `-negative_edge_logic` options.

For proper operation of the gated design, use the `-positive_edge_logic` and `-negative_edge_logic` options of the `set_clock_gating_style` command to choose any combination of gates that provide the appropriate functionality shown in [Table 10](#) and [Table 11](#). [Table 10](#) provides information for the latch-based clock-gating style and [Table 11](#) provides information for the latch-free clock-gating style

Note:

If the Power Compiler tool adds an inverter on the clock line to a rising-edge-triggered register, the Design Compiler tool might infer a falling-edge-triggered register during later synthesis if one is available in your library. If the Power Compiler tool removes an inverter from the clock line to a falling-edge-triggered register, the Design Compiler tool might infer a rising-edge-triggered register if one is available in your library. These actions are normal.

For example, to achieve AND functionality, you can simply use an AND gate. However, AND functionality also results from the combination of an INV and a NOR gate. Any combination of individual gates is allowable if the combination results in the appropriate functionality shown in [Table 10](#) and [Table 11](#).

With the following options, latch-based clock gating uses an AND gate for gating clocks of rising-edge-triggered register banks and an OR gate for gating clocks of falling-edge-triggered register banks. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.

```
-positive_edge_logic {and} -negative_edge_logic {or}
```

With the following options, latch-based clock gating chooses a NOR gate for gating clocks of rising-edge-triggered register banks. Clock gating inserts an inverter in the fanin to the 2-input clock gate and a buffer in the fanout from the 2-input clock gate. This combination results in AND functionality.

```
-positive_edge_logic {inv nor buf} -negative_edge_logic {inv and inv}
```

For falling-edge-triggered register banks in this example, clock gating uses an AND gate to gate the clock. Clock gating inserts inverters in the fanin and fanout of the 2-input clock gate. This combination results in OR functionality. The enable input of the OR gate already has an inverter. This cancels the effect of the additional inverter on the enable input signal. Therefore, only the clock pin of the main gate is inverted.

Table 10 Gating Functionality for Latch-Based Clock Gating

Latch-based clock gating					
Gating logic -pos{} or -neg{}	Rising-edge-triggered registers		Falling-edge-triggered registers		
	Valid	Remarks	Valid	Remarks	
{and}	Yes				
{or}			Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.	
{nand}	Yes	Clock gating adds an inverter to the clock line to the register.			
{nor}			Yes	Clock gating removes the inverter from the clock line to the register.	
{and inv}	Yes	Clock gating adds an inverter to the clock line to the register.			

Table 10 Gating Functionality for Latch-Based Clock Gating (Continued)

		Latch-based clock gating			
		Rising-edge-triggered registers		Falling-edge-triggered registers	
Gating logic -pos{} or -neg{}		Valid	Remarks	Valid	Remarks
{or inv}				Yes	Clock gating removes the inverter from the clock line to the register.
{nand inv}	Yes			Yes	
{nor inv}				Yes	
{inv and}				Yes	Clock gating removes the inverter from the clock line to the register.
{inv or}	Yes	Clock gating adds an inverter to the clock line to the register.			
{inv nand}				Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.
{inv nor}	Yes			Yes	
{inv and inv}				Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.
{inv or inv}	Yes			Yes	
{inv nand inv}				Yes	Clock gating removes the inverter from the clock line to the register.
{inv nor inv}	Yes	Clock gating adds an inverter to the clock line to the register.			

Table 11 Gating Functionality for Latch-Free Clock Gating

		Latch-free clock gating			
		Rising-edge-triggered registers		Falling-edge-triggered registers	
Gating logic -pos{} or -neg{}		Valid	Remarks	Valid	Remarks
{and}				Yes	
{or}	Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.			
{nand}				Yes	Clock gating removes the inverter from the clock line to the register.
{nor}	Yes	Clock gating adds an inverter to the clock line to the register.			
{and inv}				Yes	Clock gating removes the inverter from the clock line to the register.
{or inv}	Yes	Clock gating adds an inverter to the clock line to the register.			
{nand inv}				Yes	
{nor inv}	Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.			
{inv and}	Yes	Clock gating adds an inverter to the clock line to the register.			
{inv or}				Yes	Clock gating removes the inverter from the clock line to the register.
{inv nand}	Yes	The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.			

Table 11 Gating Functionality for Latch-Free Clock Gating (Continued)

		Latch-free clock gating			
		Rising-edge-triggered registers		Falling-edge-triggered registers	
Gating logic -pos{} or -neg{}		Valid	Remarks	Valid	Remarks
{inv nor}				Yes	
{inv and inv}	Yes		The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.		
{inv or inv}				Yes	
{inv nand inv}	Yes		Clock gating adds an inverter to the clock line to the register.		
{inv nor inv}				Yes	Clock gating removes the inverter from the clock line to the register.

Choosing a Simple Gating Cell by Name

The `-positive_edge_logic` and `-negative_edge_logic` options allow you to use a specific clock-gating cell during clock gating. To use a specific gating cell from the target library, specify the cell name after the element type, separated by a colon.

With the following option for rising-edge-triggered register banks, latch-based clock gating chooses the specific AND gate named MYAND2 from the target library. In this example, the tool inserts a buffer in the fanout of the clock gate.

```
-positive_edge_logic {and:MYAND2 buf}
```

Choosing a Simple Gating Cell and Library by Name

In some cases, you might have more than one target library with cell names that are the same. In such cases, you can use a specific cell from a specific library for clock gating.

The `-positive_edge_logic` and `-negative_edge_logic` options allow you to indicate a specific library and cell for clock gating, as follows:

```
target_library = { "CMOS8_MAX.db" "tech_lib1.db" "tech_lib2.db" }
-positive_edge_logic {and:tech_lib1/MYAND2 buf:tech_lib2/ MYBUF2}
```

In this example, clock gating uses a particular AND cell and BUF cell from different logic libraries. The AND cell is MYAND2 from the tech_lib1 library, and the buffer is MYBUF2 from the tech_lib2 library. You must have previously specified these libraries as target libraries by setting the Design Compiler target_library variable.

Designating Simple Cells Exclusively for Clock Gating

During technology mapping, the Design Compiler tool builds clock-gating logic, using the generic representation created by the Power Compiler tool and cells from your library. Unless you are using an integrated cell for gating, there is nothing to prevent the Design Compiler tool from using the same cells for mapping other parts of the design.

You can designate certain cells to be used exclusively or preferentially for gating clocks. Such cells can be the 2-input clock gate, inverters, buffers, or latches used in the latch-based style of clock gating.

To use a specific cell for clock gating and preclude its use in other areas of the design, set the following Library Compiler attributes to true in the library description of the cell:

- `dont_use`

When set to true, this attribute prevents the Design Compiler tool from choosing the cell when mapping the design to technology.

- `is_clock_gating_cell`

This is an attribute of type Boolean for the cell group. When set to true, this attribute identifies the cell for use in clock gating. If `dont_use` and `is_clock_gating_cell` are both set to true, the cell is used only in clock-gating circuitry.

You can set `dont_use` and `is_clock_gating_cell` on

- 2-input clock gates

Examples of 2 clock gates are AND, NAND, OR, and NOR library cells that are used to gate clocks.

- 1-input clock gates

Examples of 1 clock gates are buffer and inverter library cells that are used in the fanin and fanout of the 2 clock gate.

- 2-input D latches

These latches can be active high or low and must have a noninverting output.

To use a cell preferentially in clock gating, set only the `is_clock_gating_cell` attribute to true. Clock gating uses such cells preferentially when inserting clock-gating circuitry. Later, the Design Compiler tool can use them when mapping other parts of the design to the target technology.

For more information about the syntax and use of Library Compiler attributes, see the Library Compiler documentation.

The 2-input clock gate has an enabling input and a clock input that is connected to ENL and CLK signals in [Figure 19](#). If the `clock` attribute is set on one of the pins of the 2-input clock gate, the Power Compiler tool recognizes the remaining input pin as the enable pin. However, library cell syntax allows you to explicitly designate an input pin as the enabling input. In the pin group of the library description for the cell, set the `clock_gate_enable_pin` attribute to `true`. This is an attribute of type Boolean for the pin group.

If the tool finds neither a `clock` attribute nor a `clock_gate_enable_pin` attribute, the tool checks for the existence of setup and hold time on the pins. If setup and hold time are found on a pin, the tool uses that pin as the enable pin. For more information about Library Compiler syntax and cell descriptions, see the Library Compiler documentation.

Choosing a Specific Latch and Library

The `-sequential_cell` option of `set_clock_gating_style` command allows you to select a clock-gating style that uses latches or avoids the use of latches. [Figure 19](#) shows an example of the latch-based clock-gating style. An example of a circuit with the latch-free clock-gating style is shown in [Figure 27](#).

The `-sequential_cell` option allows you to use a specific latch when inserting clock-gating circuitry. To use a specific latch from the target library, specify the name of the latch after the element type, separating the two with a colon (:). For example:

```
-sequential_cell latch:LAH10
```

In the following example, clock gating uses the LAH10 latch from the SPECIFIC_TECHLIB library.

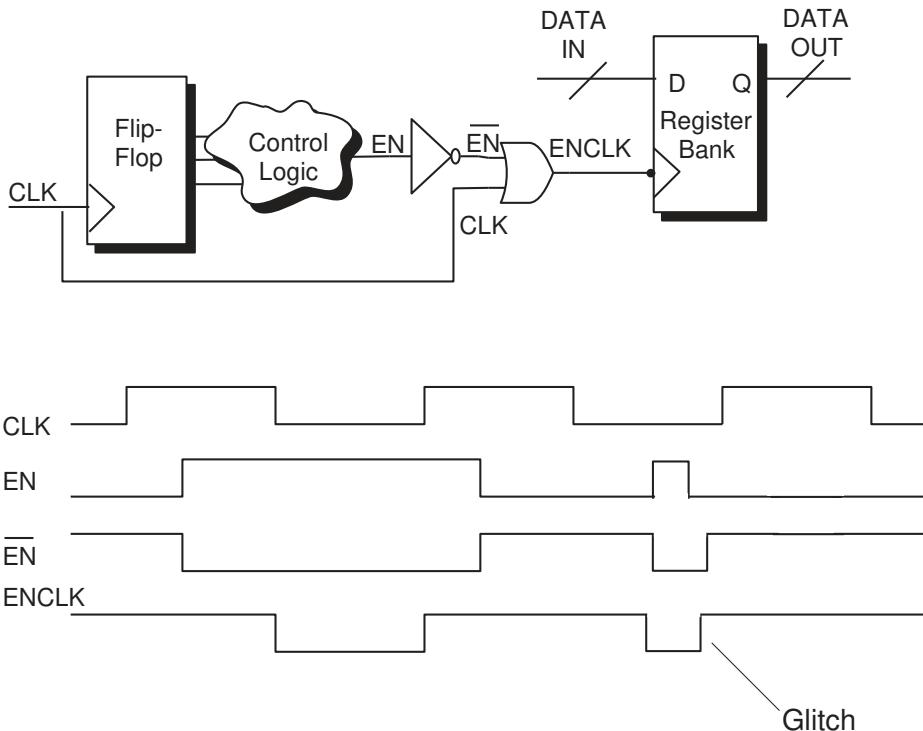
```
-sequential_cell latch:SPECIFIC_TECHLIB/LAH10
```

Choosing a Latch-Free Style

To specify a latch free clock gating style, use the `-sequential_cell none` option of the `set_clock_gating_style` command. For example, in the latch-free style in [Figure 27](#), clock pulses to the register bank are gated by the OR gate and it prevents the trailing clock edge. A latch-free clock gate for rising-edge-triggered logic prevents the falling clock edge.

Eliminating the latch can reduce power dissipation and area slightly. However, the latch-free method has a significant drawback: The EN signal must be stable at its new value before the falling clock edge. If the EN signal is not stable before the falling clock edge, glitches on the EN signal can corrupt the clock signal to the register. Any glitches on the EN signal after the trailing edge of the clock lead to glitching and corruption of the gated clock signal. See [Figure 27](#) for an example of latch-free clock gating.

Figure 27 Latch-Free Clock Gating



Improving Testability

Clock gating introduces multiple clock domains in the design. Introducing multiple clock domains can affect the testability of your design unless you add logic to enhance testability.

In certain scan register styles, a gated register cannot be included in a scan chain, because gating the register's clock makes it uncontrollable for test (assuming there is no dedicated scan clock). Without the register in the scan chain, test controllability is reduced at the register output and test observability is reduced at the register input. If you have many gated registers, this can significantly reduce the fault coverage in your design.

You can improve the testability of your circuit by using the options of the `set_clock_gating_style` command to determine the amount and type of testability logic added during clock gating. Follow these steps to improve testability:

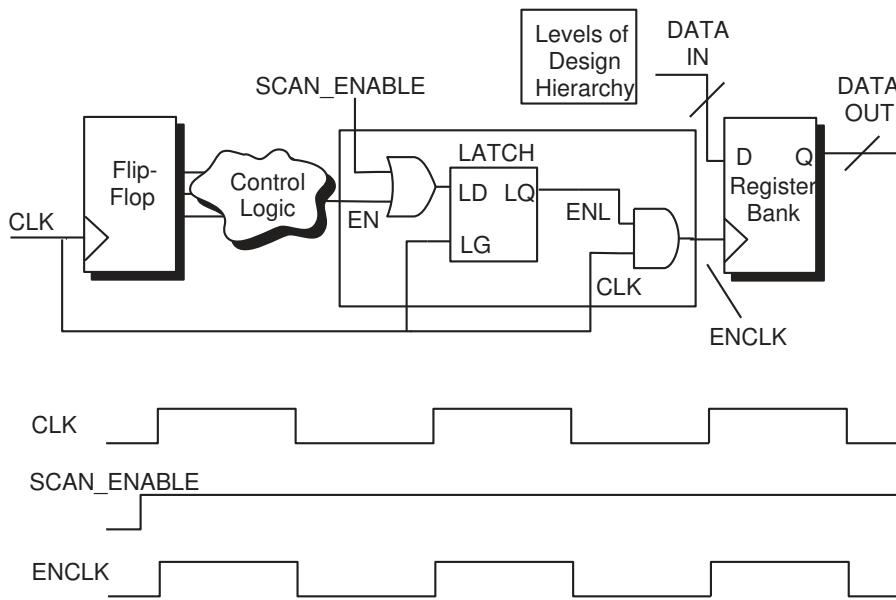
- Add a control point for testing
- Choose `test_mode` or `scan_enable`
- Add observability logic

Inserting a Control Point for Testability

A control point increases the testability of your design by restoring the clock signal to its ungated form during test. The control point is an OR gate that eliminates the function of the clock gate during test, which restores the controllability of the clock signal.

[Figure 28](#) shows a control point (OR gate) connected to the scan_enable port. The control point is before the latch in this example.

Figure 28 Control Point in Gated Clock Circuitry



When the **scan_enable** signal is high, the test signal overrides clock gating, thus making the **ENCLK** and **CLK** signals identical during shift mode. The test solution in [Figure 28](#) has the advantage of achieving testability with the addition of only one OR gate. This configuration has fault coverage comparable to that of a design without clock gating.

The `set_clock_gating_style` command has two options to determine the location and type of the control point for test:

- `-control_point none | before | after`

The default is `none`. The `-control_point` option inserts your control point before or after the clock-gating latch. When using the latch-free clock-gating style, before and after are equivalent.

- `-control_signal test_mode | scan_enable`

The default is `scan_enable`. This option creates a `scan_enable` or `test_mode` test port and connects the port to the control-point OR gate. The TestMAX DFT tool interprets `test_mode` and `scan_enable` in a specific manner. The `-control_signal` option also applies to any observability logic inserted by the `-observation_point` option. You can use the `control_signal` option only if you have used the `-control_point` option.

When creating the control point, the Power Compiler tool creates and names a new test port and assigns appropriate attributes to the port. [Table 12](#) shows variables that the tool checks when naming the new port and when setting attributes on it.

Table 12 Test Port Naming and Attribute Assignment

Setting of <code>-control_signal</code>	Variable that determines test port name	Attributes on test port are the same as those set by
<code>scan_enable</code>	<code>test_scan_enable_port_naming_style</code>	<code>set_dft_signal -type ScanEnable</code>
<code>test_mode</code>	<code>test_mode_port_naming_style</code>	<code>set_attribute test_port_clock_gating</code> <code>set_dft_signal -type TestMode</code>

To connect the test port of the clock-gating design to the test port of your design, use the `insert_dft` command. For more information, see [Connecting the Test Ports Throughout the Hierarchy](#).

Latch-based clock gating requires that the enable signal always arrive after the trailing edge (rising edge for falling-edge signal) of the clock. If you insert the control point before the latch, it is impossible for the control point to violate this requirement. However, your test tool might not support positioning the control point before the clock-gating latch. In such cases, use `-control_point after` to insert the control point after the clock-gating latch.

Note:

If you insert the control point after the latch, the scan_enable signal or test_mode signal must transition after the trailing edge (rising edge for falling-edge signal) of the clock signal during test at the foundry; otherwise glitches in their resulting signal corrupts the clock output.

Scan Enable Versus Test Mode

Scan enable and test mode differ in the following way:

- Scan enable is active only during the scan mode.
- Test mode is active during the entire test (scan mode and parallel mode).

Scan enable typically provides higher fault coverage than test mode. Fault coverage with scan enable is comparable to a circuit without clock gating. However, there can be situations in which you must use test mode. For example, you might need to use test mode if you place the control point before the latch and your test tool does not support this position of the control point with scan enable.

Improving Observability With Test Mode

When using test mode, the EN signal and other signals in the control logic are untestable. If your test methodology requires that you use test_mode, you might need to increase your fault coverage. You can increase fault coverage with test mode by adding observability logic during clock gating.

Note:

When using the -control_signal scan_enable option, increasing observability with observability logic is not necessary.

The set_clock_gating_style command has two options for increasing observability when using the -control_signal test_mode option:

- -observation_point true | false

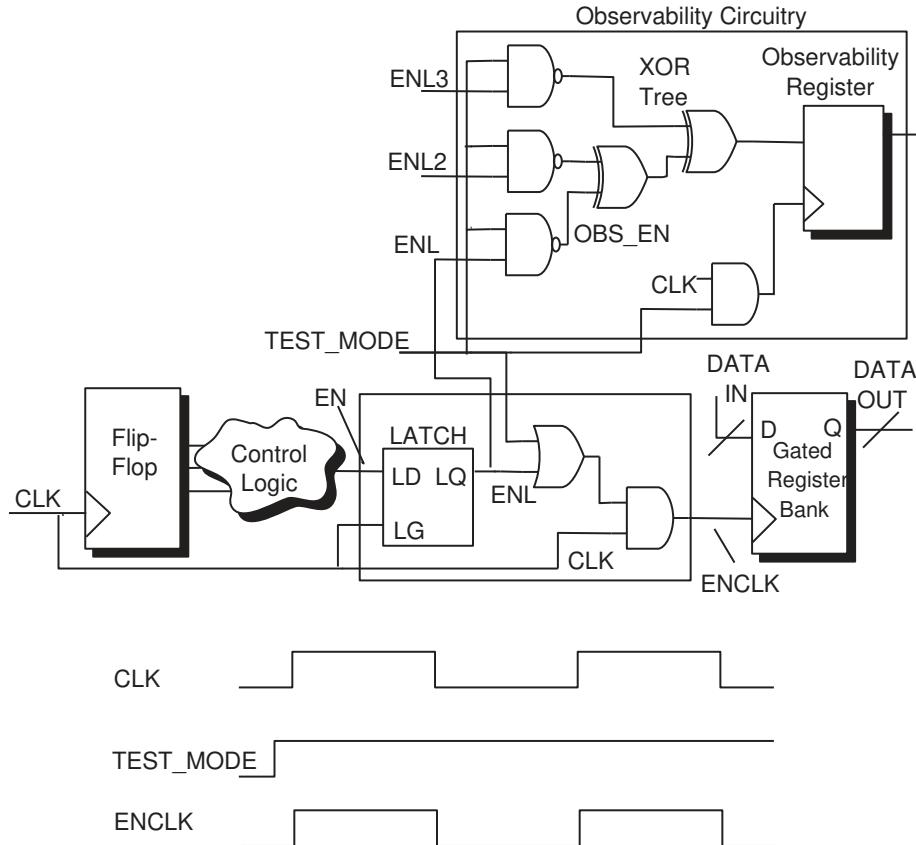
The default is false. When you set this option to true, clock gating adds a cell that contains at least one observability register and an appropriate number of XOR trees (if there is only one signal to be observed, an XOR tree is unnecessary). The scan chain includes the observability register, but the observability register's output is not functionally connected to the circuit.

- -observation_logic_depth depth_value

The default is 5. The value of this option determines the depth of logic of the XOR tree that -observation_point option builds during clock gating. If this value is set to 0, each ENL signal is latched separately and no XOR tree is built. The XOR tree reduces the number of observability registers needed to capture the test signature.

[Figure 29](#) shows a gated clock, including an observability register and an XOR tree.

Figure 29 Gated Clock With High Observability



During test, observability circuitry allows observation of the ENL signal. During normal operation of the circuit, the XOR tree does not consume power, because the NAND gate blocks all ENL signal transitions. This test solution has high testability and is power-efficient, because the XOR tree consumes power only during test and the clock of the observability register is gated.

To connect the test port of the clock-gating design to the test port of your design, see [Connecting the Test Ports Throughout the Hierarchy](#).

Choosing a Depth for Observability Logic

Use the `-observation_logic_depth` option of the `set_clock_gating_style` command to set the logic depth of the XOR tree in the observability cell. The default is 5.

The Power Compiler tool builds one observability cell for each clock-gated design. Each gated register in the design provides a gated enable signal (OBS_EN in [Figure 29](#)) as input to the XOR tree in the observability cell.

If you set the logic depth of your XOR tree too small, clock gating creates more XOR trees (and associated registers) to provide enough XOR inputs to accommodate signals from all the gated registers. Each additional XOR tree adds some overhead for area and power.

If you set the logic depth of your XOR tree too high, clock gating can create one XOR tree with plenty of inputs. However, too large a tree can cause the delay in the observability circuitry to become critical.

Use the following guidelines in choosing or changing the logic depth of your XOR tree. Choose a value that is

- High enough to cause the construction of as few XOR trees as possible
- Low enough to keep the delay in the observability circuitry from becoming critical

Connecting the Test Ports Throughout the Hierarchy

You use the `insert_dft` command to connect the test ports through various level of the design hierarchy.

If you have used the clock-gating feature with the testability options, you must connect the test ports using the `insert_dft` command. After you have compiled all the lower level hierarchies of the design, use the command on the top level of the design.

There are two types of test ports: the `test_mode` port and the `scan_enable` port. A port can be recognized as a test port if it is designated as a `scan_enable` or a `test_mode` port using the `set_dft_signal` command. Alternatively, a port can be designated as a test port by setting the `test_port_clock_gating` attribute on it.

A `scan_enable(test_mode)` port is only connected to other `scan_enable(test_mode)` ports in the design hierarchy. If a `scan_enable(test_mode)` port exists at a particular level of the hierarchy, it is connected to `scan_enable(test_mode)` ports at all higher levels of the hierarchy. If a `scan_enable(test_mode)` port does not exist at a higher level of hierarchy, the `scan_enable(test_mode)` port is created.

The `insert_dft` command connects the test ports on all levels of the design hierarchy to the `test_mode` or `scan_enable` pins of the OR gate in the clock gating logic and the XOR gates in the clock-gating observability logic. If the design does not have a test port at any level of hierarchy, a test port is created. If a test port exists, it is used.

Using the `insert_dft` Command

You use the `insert_dft` command to connect the top-level test ports to the test pins of the clock-gating cells through the design hierarchy. A test port is created if the design does not have a test port at any level of the hierarchy. To identify the test ports, the tool uses the options you specified using the `set_dft_signal` command. The following example shows the usage of the `insert_dft` command to connect to the clock-gating cells. When you specify the value `clock_gating` to the `-usage` option of the `set_dft_signal` command,

during the execution of the `insert_dft` command, the tool connects the specified signal to the test pin of the clock-gating cells.

```
dc_shell> read_ddc design.ddc
dc_shell> set_clock_gating_style -control_signal scan_enable \
           -control_point before
dc_shell> compile_ultra -scan -gate_clock
dc_shell> set_dft_signal -type ScanEnable -port test_se_1
dc_shell> set_dft_signal -type ScanEnable -port test_se_2 \
           -usage clock_gating
dc_shell> create_test_protocol
dc_shell> dft_drc -verbose
dc_shell> preview_dft
dc_shell> insert_dft
```

Using Instance-Specific Clock-Gating Styles

The Power Compiler tool supports setting and removing clock-gating styles on specific design instances and on power domains. You can also enable and disable clock gating by overriding the specified styles. These instance-specific clock-gating styles are honored only by the `compile_ultra -gate_clock` command, as described in the following sections:

- [Specifying Clock-Gating Style on Design Objects](#)
- [Instance-Specific Clock-Gating Style Example](#)
- [Removing the Instance-Specific Clock-Gating Style on Design Objects](#)

Specifying Clock-Gating Style on Design Objects

The clock-gating style specified using the `set_clock_gating_style` command are applied to the entire design by default. To restrict the clock-gating style to specific objects of the design, follow these steps:

1. Set the `power_cg_iscgs_enable` variable to `true`. The default is `false`.
2. Use the `-instances` or the `-power_domains` option of the `set_clock_gating_style` command to restrict the clock-gating styles to be applied to the specified instances or power domains, respectively.

The clock-gating cells are inserted, based on the clock-gating style that you specified.

When you set the `power_cg_iscgs_enable` variable set to `true`, and a specific instance does not have a specified clock-gating style, the tool chooses a clock-gating style in the following decreasing order of priority:

- The style specified on the power domain containing the instance
- The style of the hierarchical cell containing the instance

- The style of the higher level hierarchical cell contains the instance
- When you do not specify the clock-gating style, the tool derives a default clock-gating style based on the specified libraries. For more information, see [Default Clock-Gating Style](#).

Note:

If you set the `power_cg_iscgs_enable` variable to `true`, and do not use the `-instances` or the `-power_domains` option, the clock-gating style is applied only to the current design.

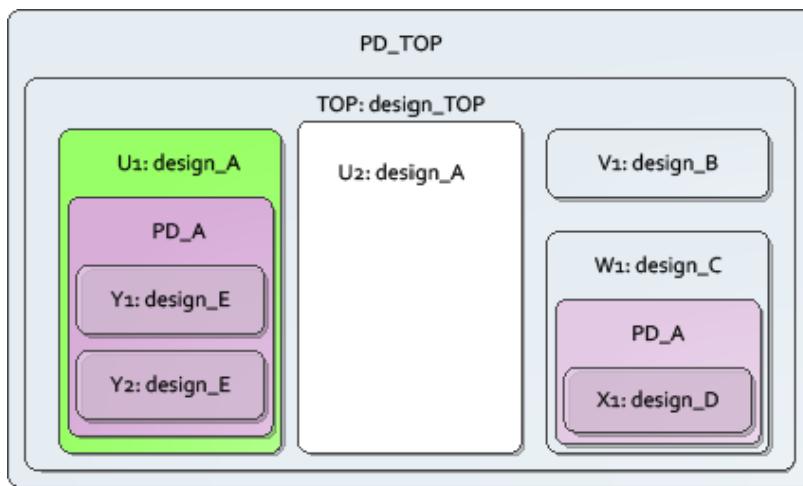
If you use the `-instances` or the `-power_domains` option of the `set_clock_gating_style` command without setting the `power_cg_iscgs_enable` variable to `true`, the tool issues a PWR-815 error message.

Instance-Specific Clock-Gating Style Example

For the design example in [Figure 30](#), the `set_clock_gating_style` command is specified as follows:

```
# Specify the clock gating Style
dc_shell> set_clock_gating_style -designs {design_A}
dc_shell> set_clock_gating_style -instances {Y2}
dc_shell> set_clock_gating_style -instances {U1}
dc_shell> set_clock_gating_style -power_domains {PD_A}
```

Figure 30 Instance Specific Clock-Gating Style Example



The priority rules defined is applied from the bottom of the hierarchy to the top. Also, the Power Compiler tool considers power domains to be more specific than design instances. In the design example of [Figure 30](#),

- The clock-gating style specified for Y2 instance is applied to the Y2 instance. If a clock-gating style is defined for a hierarchical cell inside the instance Y2, the clock-gating style of the hierarchical cell is applied to Y2. This is because, the precedence rule is applied from the bottom of the hierarchy to the top.
- The clock-gating style specified for Y2 instance has higher precedence than clock-gating style defined for PD_A power domain.
- The clock-gating style specified for U1 instance is applied to U1 instance, design design_A design, and PD_A power domain.
 - The clock-gating style specified for the TOP design is applied to W1 and V1 instances.

Removing the Instance-Specific Clock-Gating Style on Design Objects

Use the `remove_clock_gating_style` command to remove the instance-specific clock-gating style that you specified on the design objects. However, this command can be used only when you set the `power_cg_iscgs_enable` variable to `true`.

Modifying the Clock-Gating Structure

While performing RTL clock gating, you can specify the `set_clock_gating_style -max_fanout` command to limit the number of registers that are gated by a single clock-gating element. The results can be multiple clock-gating elements that have the same enable signal and, logically, the same gated-clock signal. All clock-gating cells with the same enable signal belong to the same clock-gating group. All registers gated by a single clock-gating element belong to the same clock-gating subgroup.

The gated registers inserted by the `compile_ultra -gate_clock` command are partitioned into subgroups. These partitions are not based on timing or placement constraints. So the placement tool tries to place the clock-gated registers close to the clock-gating cell, but this might not happen because of other design constraints. The result is a suboptimal partition of gated registers into subgroups.

You can correct this problem by moving clock-gated registers between the clock-gating cells belonging to the same clock-gating group. Because these clock-gating cells are logically equivalent, the rewired circuit is functionally valid.

To rewire or remove clock gating in your design, use the `rewire_clock_gating` or `remove_clock_gating` command.

Changing a Clock-Gated Register to Another Clock-Gating Cell

To selectively rewire a clock-gated register from one clock-gating cell to another logically equivalent clock-gating cell, use the `rewire_clock_gating` command.

However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not perform rewiring of such clock-gating cells.

You can use the `-undo` option to remove any rewiring you specified with the `rewire_clock_gating` command. Based on the options specified, the `-undo` option deletes the directives specified by the previously specified `rewire_clock_gating` command. Use the `-undo` option before you use the `compile -incremental` command. The `compile` command modifies the netlist to rewire the gated registers.

Because rewiring the gated registers alters the clock-gating cell that gates the registers, any path-based timing exception that goes through the old clock-gating cell to a gated register is no longer relevant and is lost.

Removing Clock-Gating Cells From the Design

The Power Compiler tool performs clock gating at the RTL level during the compilation process when you use the `compile_ultra -gate_clock` command. The `remove_clock_gating` command lets you selectively remove the clock gates without having to start at RTL again. The subsequent `compile_ultra` command removes the selected clock-gating cells. As a result you have the ability to use aggressive clock-gating strategies initially and selectively remove clock-gating cells, if needed.

This command removes redundant clock-gating cells that are no longer connected to any clock-gating cells. Any associated test observation logic is also optimized. However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not remove such cells.

All the registers that are not driven by the clock-gated signals are remapped to new sequential cells. This might result in new pin names for the registers. If there are pin-based timing exceptions set on the original register, these exceptions might not transfer properly during the transformation, if the new and the original pin names do not match. Pin-based timing exceptions are specified by using the `set_max_delay`, `set_min_delay`, `set_multicycle_path`, and `set_false_path` commands.

The `remove_clock_gating` command displays a warning if there are pin-based timing exceptions on the register to be ungated. Cell based timing exceptions are not affected because the ungated registers retain their name. It is advisable to use the cell-based timing exceptions with clock-gating registers.

For information, see the Design Compiler documentation.

Rewiring Clock Gating After Retiming

The Power Compiler tool supports the `-balance_fanout` option to the `rewire_clock_gating` command.

This command is used to rebalance the fanout of the clock gates within the design after modifications have been made during retiming. During optimization, the tool automatically balances the register banks based on the minimum and maximum fanout requirements. However, when you run the `compile -ungroup` or `optimize_registers` commands that perform retiming, unconnected registers are removed to improve timing. For clock tree synthesis, ensure that the clock gates have equivalent fanout loads by using the `-balance_fanout` option.

Use the `rewire_clock_gating -balance_fanout` command either after retiming or after compilation to restore a balanced fanout. When you use this command, the tool compares the changed fanout of each equivalent clock-gating cell. The registers are moved around so that each equivalent clock-gating cell now has a balanced set of registers and honors the `-max_fanout` option that you specified originally. Any register banks not meeting the `minimum_bitwidth` requirement are ungated. However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not perform fanout balancing on such cells.

Note:

The command is not intended for use after the `balance_registers` command.

Integrated Clock-Gating Cells

An integrated clock-gating cell integrates the various combinational and sequential elements of a clock gate into a single cell located in the logic library. An integrated clock-gating cell is a cell that you or your library developer creates to use especially for clock gating.

Consider using an integrated clock-gating cell if you are experiencing timing problems, such as clock skew, caused by the placement of clock-gating cells on your clock line.

Use the Library Compiler tool to create an integrated cell for clock gating. For detailed information, see the Library Compiler documentation.

The Library Compiler tool assigns a black box attribute to the complex sequential cells such as integrated clock-gating cells. The Design Compiler tool does not use the integrated cells for general logic synthesis. However, the Power Compiler tool uses these integrated clock-gating cell for clock gating. The selection of the clock-gating cell is determined either by the default or the values specified with the `set_clock_gating_style` command. Each integrated clock-gating cell in the library must contain the Library Compiler `clock_gating_integrated_cell` attribute. This attribute

can be set to either the string `generic` or to one of 26 strings that represent specific clock-gating types. The `generic` setting causes the Library Compiler tool to infer the `clock_gating_integrated_cell` attribute from the functionality of the clock-gating cell. Using one of the 26 standard strings specifies the functionality explicitly according to established conventions. For more details, see [Appendix B, Integrated Clock-Gating Cell Example](#).

Integrated Clock-Gating Cell Attributes

The `clock_gating_integrated_cell` attribute should be set to one of 26 function-specific strings, such as `latch_posedge_postcontrol`. Each string is a concatenation of up to four strings that describe the cell's functionality. The library developer specifies the attribute when the integrated cell is created. When you set the `clock_gating_integrated_cell` attribute to `generic`, the Power Compiler tool infers the value from the Library Compiler attribute.

For more information, see the *Library Compiler User Guide*.

The `clock_gating_integrated_cell` attribute can have any one of 26 different values. [Table 13](#) contains a short list of example values and their meanings.

Table 13 Examples of Values for Integrated Clock-Gating Cell

Value of <code>clock_gating_integrated_cell</code> Integrated cell must contain	
<code>latch_negedge</code>	Latch-based gating logic Logic appropriate for gating falling-edge-triggered registers
<code>latch_posedge_postcontrol</code>	Latch-based gating logic Logic appropriate for gating rising-edge-triggered registers Test control logic located after the latch
<code>latch_negedge_precontrol</code>	Latch-based gating logic Logic appropriate for gating falling-edge-triggered registers Test control logic located before the latch
<code>none_posedge_control_obs</code>	Latch-free gating logic Logic appropriate for gating rising-edge-triggered registers Test control logic (no latch) Observability port

For more examples, see [Appendix B, Integrated Clock-Gating Cell Example](#).

The `set_clock_gating_style` command determines the integrated cell that the Power Compiler tool uses for clock gating. The tool searches the library for the integrated cell that has the attribute value corresponding to the options you specify with the `set_clock_gating_style` command.

For example, consider the following command:

```
set_clock_gating_style -sequential_cell latch \
    -positive_edge_logic {integrated} -control_point before \
    -control_signal test_mode -observation_point true
```

The `-sequential_cell latch` and `-control_point before` options result in a latch-based style, and the tool searches for an integrated clock-gating cell with `control` as the third string parameter of the `clock_gating_integrated_cell` attribute.

The tool selects a latch-based positive-edge integrated clock-gating cell because you specified the `-positive_edge_logic {integrated}` option. If your library does not contain a positive-edge integrated clock-gating cell, the tool chooses a nonintegrated clock-gating cell that meets the current clock-gating style. If the `power_cg_permit_opposite_edge_icg` is `true`, the tool can choose a negative-edge integrated clock-gating cell with an inverter to achieve the positive-edge trigger.

If more than one integrated cell has the correct attribute value, the Power Compiler tool chooses the first integrated cell that it finds in the target library. If you have a preference, specify the integrated cell by name.

The Power Compiler tool does not check the function of the integrated cell to ensure that it complies with the value of the `clock_gating_integrated_cell` attribute. The correct functionality is checked by the Library Compiler tool when the integrated cell is initially created. The Power Compiler tool searches for an integrated clock-gating cell that contains the specified attribute value.

Pin Attributes

The Power Compiler tool requires certain Library Compiler attributes on the pins of your integrated clock-gating cell. [Table 14](#) lists the required pin attributes for pin names that pertain to clock gating. Some pins, such as the pins for test and observability are optional; however, if a pin is present, it must have the corresponding attribute listed in [Table 14](#).

Table 14 Pin Attributes for Integrated Clock-Gating Cells

Integrated cell pin name	Input or output	Required Library Compiler attribute
clock	Input	<code>clock_gate_clock_pin</code>
enable	Input	<code>clock_gate_enable_pin</code>

Table 14 Pin Attributes for Integrated Clock-Gating Cells (Continued)

Integrated cell pin name	Input or output	Required Library Compiler attribute
test_mode or scan_enable	Input	clock_gate_test_pin
enable_clock	Output	clock_gate_out_pin
observability	Output	clock_gate_obs_pin

Other tools used in your synthesis and verification flow might require additional pin attributes that are not specific to clock gating and are not listed in [Table 14](#).

For more information about Library Compiler attributes and library syntax, see the Library Compiler documentation.

Timing Considerations

Clock gating requires certain timing arcs on your integrated clock-gating cell.

For latch-based clock gating,

- Define setup and hold arcs on the enable pin with respect to the clock pin.
For the latch-based gating style, these arcs are defined with respect to the controlling edge of the clock that is driving the latch.
- Define combinational arcs from the clock and enable inputs to the output.

For latch-free clock gating,

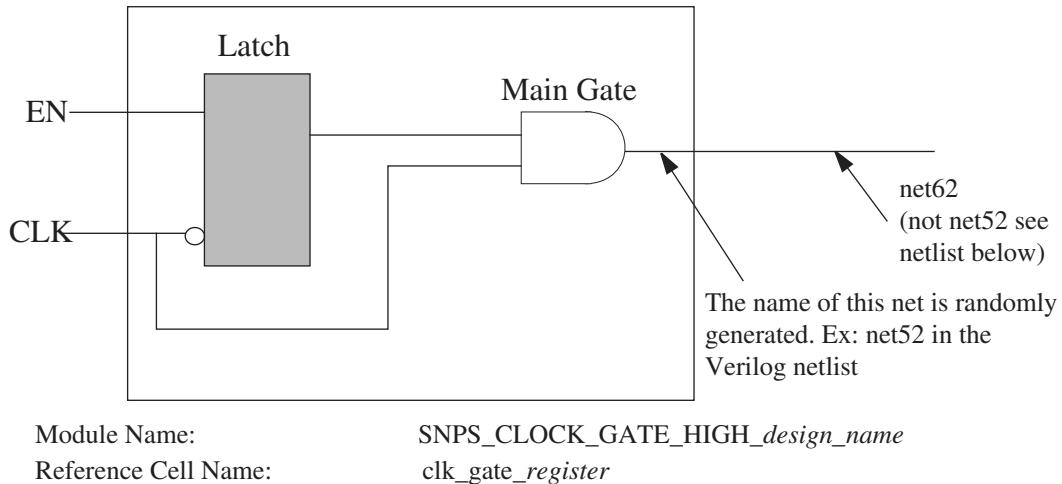
- Define no-change arcs on the enable pin with respect to the clock pin.
For the integrated latch-free gating style, these arcs must be no-change arcs, because they are defined with respect to different clock edges.
- Define combinational arcs from the clock and enable inputs to the output.

For more detailed information about timing your integrated cell, see the Library Compiler documentation.

Clock-Gating Naming Conventions

Clock-gating creates subdesigns containing clock-gating logic. Default naming conventions are shown in [Figure 31](#).

Figure 31 Default Naming Conventions



The Verilog netlist looks as follows:

```
module SNPS_CLOCK_GATE_HIGH_ff_03 ( CLK, EN, ENCLK );
    input CLK, EN;
    wire net50, net52, net53, net56;
    assign net50 = CLK;
    assign net50 = CLK;
    assign ENCLK = net52;
    assign net53 = EN;

    L_CSLDP1NQW latch ( .D(net53), .ENN(net50),
    .Q(net56) );
    L_CSAN2 main_gate ( .A(net56), .B(net50), .Z(net52) );
endmodule

module ff_03 ( q, d, clk, e, clr );
    output [2:0] q;
    output [2:0] q;
    input [2:0] d;
    input clk, e, clr;
    wire N0, net62;

    L_CSF2QP \q_reg[2] ( .D(d[2]), .CP(net62), .RN(clr),
    .Q(q[2]) );
    L_CSF2QP \q_reg[1] ( .D(d[1]), .CP(net62), .RN(clr),
    .Q(q[1]) );
    L_CSF2QP \q_reg[0] ( .D(d[0]), .CP(net62), .RN(clr),
```

```
.Q(q[0]) );
  SNPS_CLOCK_GATE_HIGH_ff_03 clk_gate_q_reg (.CLK(clk),
.EN(N0),
.ENCLK(net62));
  L_CSIV1 U5 (.A(e), .Z(N0));
endmodule
```

The module_name(SNPS_CLOCK_GATE_...), reference cell_name(clk_gate..) and the gated_clock enable net name(net62) could be changed according to your preferences.

Set the power_cg_module_naming_style, power_cg_cell_naming_style, and power_cg_gated_clock_net_naming_style variables before running the compile_ultra -gate_clock command.

Use the variables either in .synopsys_setup(dc file or before clock-gate insertion. The details of the implementation are as follows:

Usage: set power_cg_module_naming_style
"prefix_%e_%l_midfix_%p_%t_%d_suffix"
where,
prefix/midfix/suffix are just examples of any constant strings that can be specified.
%e - edge type (HIGH/LOW)
%l - library name of integrated clock gating cell library or concatenated target_library names
%p - immediate parent module name
%t - top module (current design) name
%d - index added if there is a name clash

Usage: set power_cg_cell_naming_style
"prefix_%c_%n_midfix_%r_%R_%d_suffix"
where,
%c - clock
%n - immediate enable signal name
%r - first gated reg bank name
%R - all gated reg banks sorted alphabetically
%d - index for splitting or name clash resolution

Usage: set power_cg_gated_clock_net_naming_style
"prefix_%c_%e_%g_%d_suffix"
%c - original clock
%e - immediate enable signal name
%g - clock gate (instance) name
%d - index for splitting or name clash resolution

Note:

If %d is not specified, the tool assumes a %d at the end.

Example Script for Naming Style

```

set power_cg_module_naming_style Synopsys_%e_mid_%t
set power_cg_cell_naming_style cg_%c_%n_mid_%R
set power_cg_gated_clock_net_naming_style gclk_%c_%n

define_design_lib WORK -path ./work_writable
set target_library cstarlib_lvt.db
set link_library { cstarlib_lvt.db }

set_clock_gating_style -sequential_cell latch -max_fanout 3 \
    -minimum_bitwidth 1
analyze -format verilog -library WORK ff_03.v
elaborate ff_03
compile_ultra -gate_clock
uniquify
create_clock -name "clk" -period 5 \
    -waveform {"0" "2.5" } { "clk" }
compile_ultra
current_design ff_03
write -format verilog -output 3.ff_03.vg -hierarchy

```

Example Script of Output Netlist

```

module Synopsys_HIGH_mid_ff_03_0 ( CLK, EN, ENCLK );
    input CLK;
    input EN;
    output ENCLK;
    wire net15, net12, net11, net9;
    assign net12 = EN;
    assign ENCLK = net11;
    assign net9 = CLK;

    L_CSAN2 main_gate ( .A(net15), .B(net9), .Z(net11) );
    L_CSLDP1NQW latch ( .D(net12), .ENN(net9), .Q(net15) );
endmodule

module ff_03 ( q, d, clk, e, clr );
    output [2:0] q;
    input [2:0] d;
    input clk;
    input e;
    input clr;
    wire N1, gclk_clk_N1_0;

    Synopsys_HIGH_mid_ff_03_0 cg_clk_N1_mid_q_reg_0 (
        .CLK(clk), .EN(N1),
        .ENCLK(gclk_clk_N1_0) );
    L_CSFDP2QP \q_reg[2] (~.D(d[2])), .CP(gclk_clk_N1_0),
    .RN(clr), .Q(q[2])
);

```

```

    L_CSF2QP \q_reg[1]  (.D(d[1]), .CP(gclk_clk_N1_0),
    .RN(clr), .Q(q[1])
);
    L_CSF2QP \q_reg[0]  (.D(d[0]), .CP(gclk_clk_N1_0),
    .RN(clr), .Q(q[0])
);
    L_CSIV1 U3 ( .A(e), .Z(N1) );
endmodule

```

Keeping Clock-Gating Information in a Structural Netlist

The Power Compiler tool applies several clock-gating attributes to the design and to the clock-gating cells and gated registers in the design. Commands such as `report_clock_gating`, `rewrite_clock_gating`, `remove_clock_gating` and several placement optimization algorithms depend on these attributes for proper operation.

The `power_cg_flatten` variable specifies whether to flatten the clock-gating cells when you use commands that perform ungrouping, such as `ungroup`, `compile -ungroup_all`, or `balance_registers`. By default, the variable is set to `false` and the clock-gating cells are not flattened. This is recommended for most situations because ungrouping the discrete clock gates could cause problems.

You can write a clock-gated structural netlist in ASCII format after synthesis. Reading back the structural netlist in ASCII format causes the clock-gating attributes to be lost, possibly preventing clock-gating and optimization from operating properly.

The tool can automatically retrieve the clock-gating attributes and identify the clock-gating cells when you read the ASCII netlist. For more information, see [Identifying and Preserving Clock-Gating Cells](#).

Identifying and Preserving Clock-Gating Cells

The clock-gate identification feature helps the tool to recognize the clock-gating cells that it inserted in the netlist in the previous run or the clock-gating cells that you instantiated in the ASCII netlist.

Identification of Clock-Gating Cells

The Power Compiler tool can identify clock-gating cells, including the hierarchical integrated clock-gating cells that exist in an ASCII netlist or the discrete hierarchical clock-gating cells inserted in a previous run of the tool.

To identify the clock-gating cells inserted by the tool and annotate the related attributes, either set the `power_cg_auto_identify` variable to `true` before reading in the design or use the `identify_clock_gating` command without specifying any options.

The following example shows how to use the `power_cg_auto_identify` variable to identify and report the identified clock-gating cells.

```
set power_cg_auto_identify true
read_verilog my_design.v
current_design my_design_top
report_clock_gating
```

Explicit Identification of Clock-Gating Cells

If the `identify_clock_gating` command used without options cannot identify specific clock-gating cells, specify the `-gating_elements` option to explicitly identify clock-gating cells. The tool sets the `pwr_cg_preservation_type` attribute on the specified cell.

To explicitly identify a clock-gating cell, the cell must have at least two input pins and one output pin; one of the input pins must be a clock pin.

Explicit identification provides you the flexibility to identify the clock-gating cells that differ from the configuration expected by the Power Compiler tool for automatic identification. However the explicitly identified clock-gating cells have additional optimization restrictions.

When a cell could be automatically identified but was explicitly identified, the tool sets the `pwr_cg_preservation_type` attribute to `preserve` on the cell, and you can remove the attribute. When a cell is identified explicitly and could not be automatically identified, the tool sets the `pwr_cg_preservation_type` attribute to `unmodifiable_read_only` and you cannot remove the attribute.

For more information on the `pwr_cg_preservation_type` attribute, see [Preserving the Identified Clock-Gating Cells](#).

The following example shows how to identify a specific clock-gating element using the `-gating_elements` option of the `identify_clock_gating` command.

```
dc_shell> read_verilog design.v
dc_shell> current_design top
dc_shell> link
# Defining the clock is not a prerequisite for clock-gate identification
# Identifies all the clock-gating cells inserted by the tool
dc_shell> identify_clock_gating

# Identifies the specified clock-gating cell
dc_shell> identify_clock_gating -gating_elements CG_1
dc_shell> report_clock_gating
```

For more details, see [Usage Flow With the `identify_clock_gating` Command](#).

Preserving the Identified Clock-Gating Cells

To preserve the identified clock-gating cells, use the `set_preserve_clock_gate` command. The command sets the `pwr_cg_preservation_type` attribute to one of the following values on the specified clock-gating cells:

- `preserve`
- `dont_modify_fanout`
- `dont_modify_enable`
- `unmodifiable`

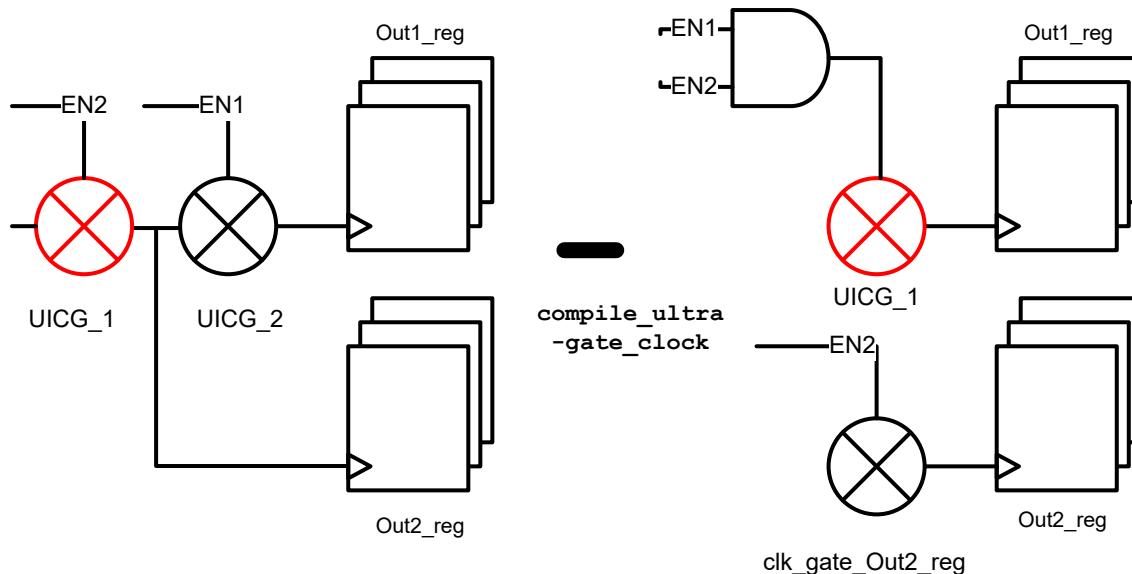
Note:

When you specify a non clock-gating cell with the `set_preserve_clock_gate` command, the Power Compiler tool ignores the command without any warning message.

The following example and [Figure 32](#) illustrate clock-gating optimization when the `pwr_cg_preservation_type` attribute is set to `preserve`:

```
set power_cg_reconfig_stages true
set_clock_gating_style -num_stages 1
identify_clock_gating
set_preserve_clock_gate [get_cells UICG_1]
```

Figure 32 Optimization When pwr_cg_preservation_type is Set to Preserve



When the `power_cg_reconfig_stages` attribute is set to `true` and a clock gate is specified with a maximum stage of 1, the tool collapses existing multilevel clock gate instances into a single stage.

In [Figure 32](#), the `UICG_1` instance is marked to be preserved with the `set_preserve_clock_gate` command. The tool can still modify the enable logic and the fanout when no additional options are specified. After optimization, the clock gate instances restrict the maximum number of stages allowed. When the `set_preserve_clock_gate` command is used, the `UICG_1` name is preserved. Otherwise, the tool chooses new names for resulting instances.

The following table describes the behavior of the `set_preserve_clock_gate` command and the `pwr_cg_preservation_type` attribute values:

Table 15 Behavior of the `set_preserve_clock_gate` Command and the `pwr_cg_preservation_type` Attribute

<code>set_preserve_clock_gate</code>	<code>pwr_cg_preservation_type</code>	Behavior
No options given	<code>preserve</code>	Preserve the specified clock-gating cell; allow all clock-gating optimizations if the cell and its name are conserved after <code>compile_ultra -gate_clock</code> command.
<code>-dont_modify_fanout</code>	<code>dont_modify_fanout</code>	Preserve the specified clock-gating cell and prevent any clock-gating optimization in the direct fanout of the specified clock-gating cell.
<code>-dont_modify_enable</code>	<code>dont_modify_enable</code>	Preserve the specified clock-gating cell and prevent any clock-gating optimization in the clock-gating enable logic.
<code>-dont_modify_fanout</code> <code>-dont_modify_enable</code>	<code>unmodifiable</code>	Prevent any further clock-gating optimization on the clock-gating fanout and clock-gating enable logic of the specified clock-gating cell.

Note:

The `-dont_modify_fanout` and `-dont_modify_enable` options restrict only clock-gating optimizations. Any other compile optimization is still allowed, such as remapping, buffering, boundary retiming, and so on.

When you use the `set_preserve_clock_gate` command for a clock-gating cell,

- If the `-dont_modify_fanout` option is not used, the tool optimizes the cell if it does not drive any load or does not meet the minimum bitwidth.
- The `remove_clock_gating` command does not remove these cells

During clock-gate merging, the tool preserves the cell that has the `pwr_cg_preservation_type` attribute set as shown in [Figure 43](#).

To remove the `pwr_cg_preservation_type` attribute, use the `remove_attribute` command. However, you cannot remove it if the `pwr_cg_preservation_type` attribute is set to `unmodifiable_read_only` during explicit clock-gating identification.

Identified Clock-Gating Cells and `dont_touch`

When you use the `set_dont_touch` command on identified clock-gating cells, the cells are affected in the following ways:

- No rewiring of the clock gate
- No removal of the clock gate
- No merging or splitting of the clock gates

The `dont_touch` setting also affects the fanout of the clock-gating cells in the following ways:

- No further addition of clock-gating cells
- No fanout balancing of flip-flops that were gated
- No addition or removal of loads on the fanout of the clock-gating cell

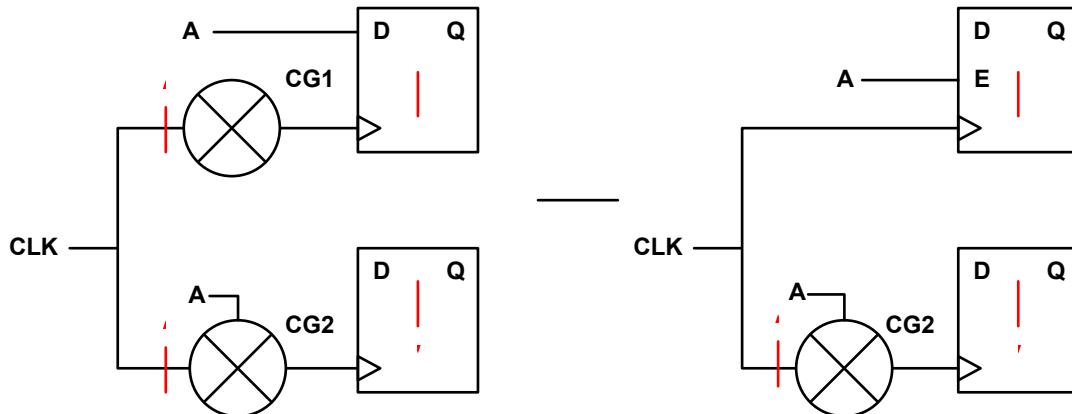
Handling Clock-Gating Edge Conflicts

Instantiated clock-gating cells that drive registers with a different activation edge are identified and optimized like any other clock-gating cell. The clock gating style for these cells are honored if they do not invert the clock signal. For example, suppose you have the following code:

```
set_clock_gating_style -minimum_bitwidth 3
identify_clock_gating
compile_ultra -gate_clock
```

With the example shown in [Figure 33](#), the tool removes the clock-gating cell named CG1 because its activation edge is the same as the register it gates. However, it does not remove the positive-edge clock-gating cell named CG2 because it gates a register with a negative-edge activation edge.

Figure 33 Optimization of Same-Edge Activation Clock-Gating Cell



Comparison of Clock-Gate Identification Methods

The advantages and disadvantages of the methods of clock-gate identification are summarized in [Table 16](#).

Table 16 Identifying Clock-Gated Designs

Command Used	Advantages	Disadvantages
<code>write_script</code>	Clock-gating attributes are written using the <code>set_attribute</code> and <code>set_preserve_clock_gate</code> commands to save the current settings. This method uses familiar commands and procedure.	Netlist changes are not supported.
<code>identify_clock_gating</code>	Netlist changes performed outside of Design Compiler are supported.	You must run this command at the right place. Some attributes such as <code>max_fanout</code> might be lost unless the <code>set_clock_gating_style</code> command is used.

Usage Flow With the `write_script` Command

Follow these steps to retrieve the clock-gating information in the ASCII netlist using the `write_script` command.

1. Set up the environment, read in the RTL design, and insert the clock-gating logic.
2. Compile the design with the required constraints.
3. Run the `change_names` command to conform to the specified rules.

4. Write out the netlist.
5. Save current attributes and settings by using the `write_script -hierarchy` command. Use the `-output` option of the command to write the output to a file. This command writes out all the attributes set by the `set_attribute` command.
6. Exit the Design Compiler session. Make sure you do not make any changes to the netlist before quitting.
7. Read in the design netlist.
8. Source the file written by the `write_script` command. This sets all the required attributes on the design, including the clock-gating cells, for proper execution throughout the flow.

If you do not need clock-gating information, use the `-no_cg` option of the `write_script` command. This results in a smaller script file.

The following example script shows the output file created by the `write_script` command.

```
#####
# Created by write_script -format dctcl on February 6, 2020 11:22 am
#####
# Set the current_design #
current_design module4

set_local_link_library {CORELIB8DLL.db}
set_attribute -type int [current_design] power_cg_max_fanout 2048
set_attribute -type boolean [get_cells clk_gate_out1_reg] \
clock_gating_logic true
set_attribute -type boolean [get_cells clk_gate_out1_reg] \
hpower_inv_cg_cell false
set_attribute -type integer [get_cells {out1_reg[0]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[1]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[2]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[3]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[4]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[5]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells clk_gate_out1_reg] \
power_cg_gating_group 0
set_size_only [get_cells latch] true
```

Usage Flow With the `identify_clock_gating` Command

This section describes the steps you follow to retrieve the clock-gating information using the `identify_clock_gating` command.

After you have saved the design that has the clock-gating information, follow these steps to retrieve the clock-gating information:

1. Read in the structural netlist that already has clock-gating cells inserted.
2. Set the `set_clock_gating_style` command. This ensures that the settings are the same as before saving the design. Otherwise, a few attributes such as `max_fanout` are not retained.
3. Use the `identify_clock_gating` command without any options to identify all clock-gating elements. This step traverses the design, searches appropriately for the clock-gating structure recognized by the power Compiler tool, and annotates the attributes needed for later operations.

Your design now contains all the clock-gating information. You can verify this using the `report_clock_gating` command.

Note:

Identify the clock-gating elements before optimizing the design so that the enable logic of the clock-gating elements can be optimized by the `compile_ultra -gate_clock` command.

Replacing Clock-Gating Cells

The Power Compiler tool detects clock-gating circuitry at the block or module level. At the module level, the clock-gating circuit can be either an instantiated or inferred logic. The tool replaces this logic with an integrated clock-gating cell or discrete cells according to the attributes that the `set_clock_gating_style` command specified. This cell replacement is performed using the `replace_clock_gates` command. This feature allows you to use the integrated clock-gating cell that is recognized by the `report_clock_gating`, `remove_clock_gating`, and `rewire_clock_gating` commands, for further operations.

Follow these steps to perform module-level replacement of clock-gating cells:

1. Set clock-gating directives and styles (optional).

The default settings of the `set_clock_gating_style` command is suitable for most designs. You can choose a value for the clock-gating conditions and a clock-gating style that is compatible with the clock-gating cell that is being replaced using the `set_clock_gating_style` command. Note that the `replace_clock_gates` command only operates on clock gate composition when using the style settings. It does not use the `-num_stages` option.

2. Read the RTL design.

3. Define the clock ports.

The clock port must be identified using the `create_clock` command before performing the replacement operation.

4. Replace manually-instantiated clock-gating cells.

To replace manually-inserted clock gates with tool-inserted clock gates, use the `replace_clock_gates` command. To perform the replacement hierarchically, use the `-global` option. If you have not specified a clock-gating styles, the tool uses the default clock-gating style.

Note:

This command replaces only combinational logic. It does not replace observability logic.

5. Compile the design.

The replaced clock-gating logic is unmapped. Use the `compile_ultra` command to compile your design.

6. Report the registers.

Use the `report_clock_gating` command to get the list of cells as shown in the following example:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
dc_shell> report_power
```

In the following example, replacement is performed on a gating cell that is driving registers in a black box cell:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> set_replace_clock_gates -rising_edge_clock RAM/clk
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
```

In the following example, replacement is performed only on selected gating cells:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> set_replace_clock_gates -exclude_cells {SUB/C10}
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
```

[Example 19](#) shows a clock-gate replacement report.

Example 19 Clock-Gate Replacement Report

```
Current clock gating style.....
Sequential cell: none
Minimum register bank size: 3
Minimum bank size for enhanced clock gating: 6
Maximum fanout: 2048
Setup time for clock gate: 1.300000
Hold time for clock gate: 0.000000
Clock gating circuitry (positive edge): or
Clock gating circuitry (negative edge): and
Note: inverter between clock gating circuitry
      and (negative edge) register clock pin.
Control point insertion: none
Control signal for control point: scan_enable
Observation point insertion: false
Observation logic depth: 5
Maximum number of stages: 5
1
replace_clock_gates -global
  Loading target library 'ssc_core_typ'
  Loading design 'regs'
Information: Performing clock-gating on design regs

Clock Gate Replacement Report
=====
| Clock |           | Include | Clock | Edge |           | Setup | Gate |
| Root  | Cell Name | Exclude | Fanin | Type | Func. | Cond. | Repl. |
=====
| clk   | C7        | -       | 1     | fall  | and   | yes   | yes  |
=====

Summary:
          number  percentage
Replaced cells (total):           1        100
Cell not replaced because
  Cell was excluded:             0        0
  Multiple clock inputs:         0        0
  Mixed or unknown clock edge type: 0        0
  No compatible clock gate available: 0        0
  Setup condition violated:      0        0
Total:                           1        100
```

Chapter 7: Clock Gating
Inserting Clock Gates With Safety Registers

Clock Gate Insertion Report

Gated Group	Flip-Flop Name	Include Exclude	Bits	Enable Cond.	Setup Cond.	Width Cond.	Clock Gated
GATED REGISTERS							
cg0	q2_reg[3]	-	1	yes	yes	yes	yes
	q2_reg[2]	-	1				
	q2_reg[1]	-	1				
	q2_reg[0]	-	1				
cg1	q3_reg[3]	-	4	yes	yes	yes	yes
UNGATED REGISTERS							
	si_reg	-	1	no	??	??	no
	ti_reg	-	1	no	??	??	no
	q4_reg[0]	-	1	no	??	??	no

Summary:

Flip-Flops

	Banks number	Bit-Width number
	percentage	percentage
Clock gated (total):	3	30
Clock not gated because		
Bank was excluded:	0	0
Bank width too small:	0	0
Enable condition not met:	7	70
Setup condition violated:	0	0
Total:	10	100

Clock gates in design

	number	percentage
Replaced clock gates:	1	16
Inserted clock gates:	3	50
Factored clock gates:	2	33
Total:	6	100

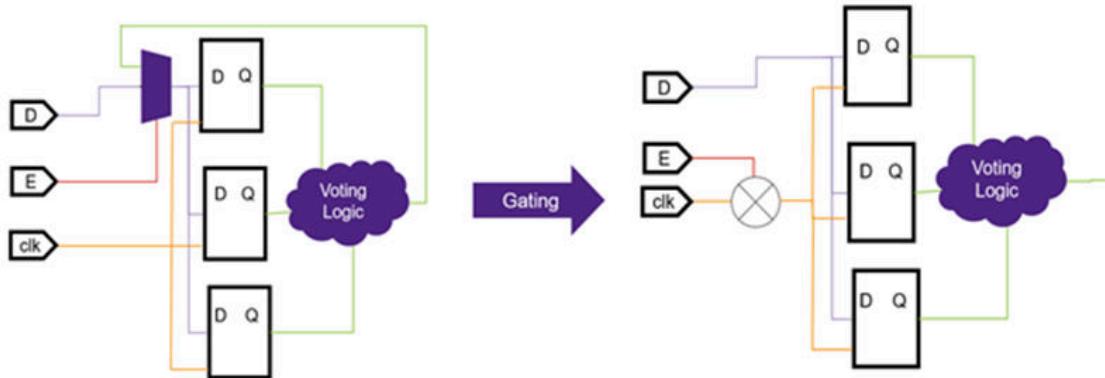
Multistage clock gating information

Number of multistage clock gates:	2
Average multistage fanout:	2.0
Number of gated cells:	16
Maximum number of clock gate stages:	3
Average number of clock gate stages:	2.2

Inserting Clock Gates With Safety Registers

You can gate triple modular redundancy (TMR) registers using the same clock gate, for all safety registers, without touching or modifying the voting logic.

Figure 34 Gating of TMR Register



The Design Compiler NXT tool automatically detects whether safety registers are used and inserts clock gates accordingly. The tool always ensures that the safety registers within the same safety group share the same clock and that the voting logic remains untouched.

Self gating is not supported for safety registers. When the tool cannot gate safety registers, the `report_clock_gating -ungated` command shows the new Safety register not supported reason.

For example, the following figure shows the ungated reasons report issued by the `report_clock_gating -ungated` command:

Ungated Register Report		
Ungated Register	Reason	What Next?
A_out_reg[0]	Safety register not supported	-
A_out_reg[0]_tmr1	Safety register not supported	-
A_out_reg[0]_tmr2	Safety register not supported	-

Clock-Gate Optimization Performed During Compilation

To further increase the power saving of your design, the Power Compiler tool uses certain techniques during compilation to reduce the number of clock-gating cells in the design. These techniques are described in the following sections:

- [Hierarchical Clock Gating](#)
 - [Enhanced Register-Based Clock Gating](#)
 - [Multistage Clock Gating](#)
 - [Clock Gate Merging](#)
 - [Placement-Aware Clock Gating in Design Compiler Graphical](#)
 - [Clock Gating Multibit Registers](#)
-

Hierarchical Clock Gating

Clock-gating techniques in the Power Compiler tool extract common enable conditions that are shared across the registers within the same block.

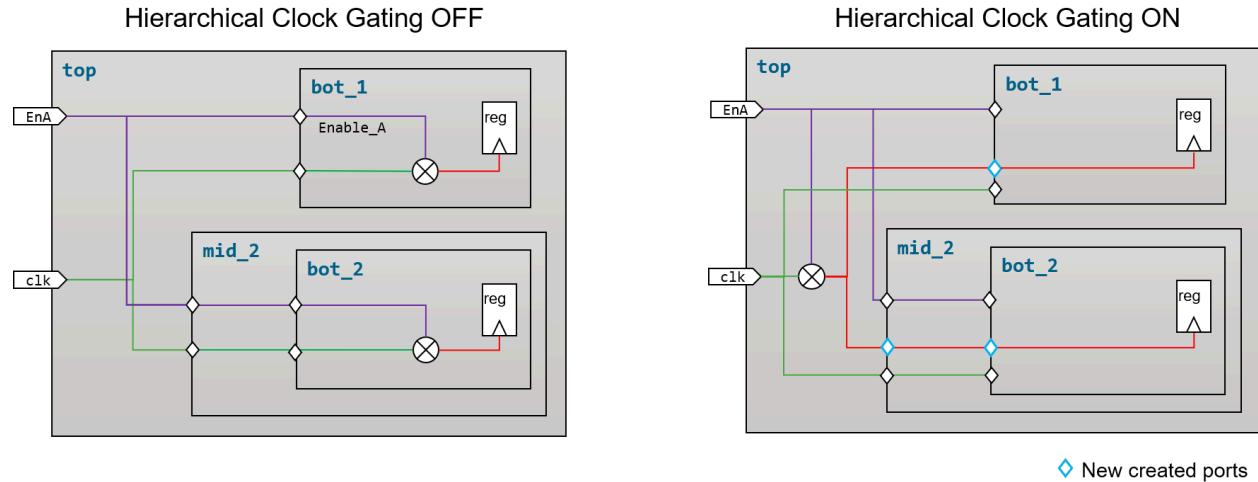
In hierarchical clock gating, during the clock-gate insertion, the tool extracts the common enables shared across registers in different levels of hierarchy in the design. This technique looks for globally shared enables while inserting clock-gating cells. This increases the clock-gating opportunities and also reduces the number of clock gates inserted. This technique, combined with proper placement, improves the power savings.

Note:

During hierarchical clock gating, the tool honors the boundary optimization settings. If you disable boundary optimization, the tool does not perform hierarchical clock-gate insertion.

The Power Compiler tool inserts hierarchical clock-gating cells at various levels of design hierarchy. As a result, additional ports are created for the clock-gated enable signal as shown in [Figure 35](#). These additional ports are added to the subdesigns. The Formality tool verifies the designs successfully when the designs being compared have the same number of primary ports.

Figure 35 Ports Added During Hierarchical Clock Gating



The Power Compiler tool can perform hierarchical clock gating on RTL netlists as well as gate-level netlists. To perform hierarchical clock gating by using the `compile_ultra -gate_clock` command, you must set the `compile_clock_gating_through_hierarchy` variable to `true` before compiling your design.

The following example shows hierarchical clock gating using the `compile_ultra` command:

```
# Set your target library and link library
# Set the clock-gating style (optional)
# Following command is optional. Use for global clock gating
dc_shell> set compile_clock_gating_through_hierarchy true

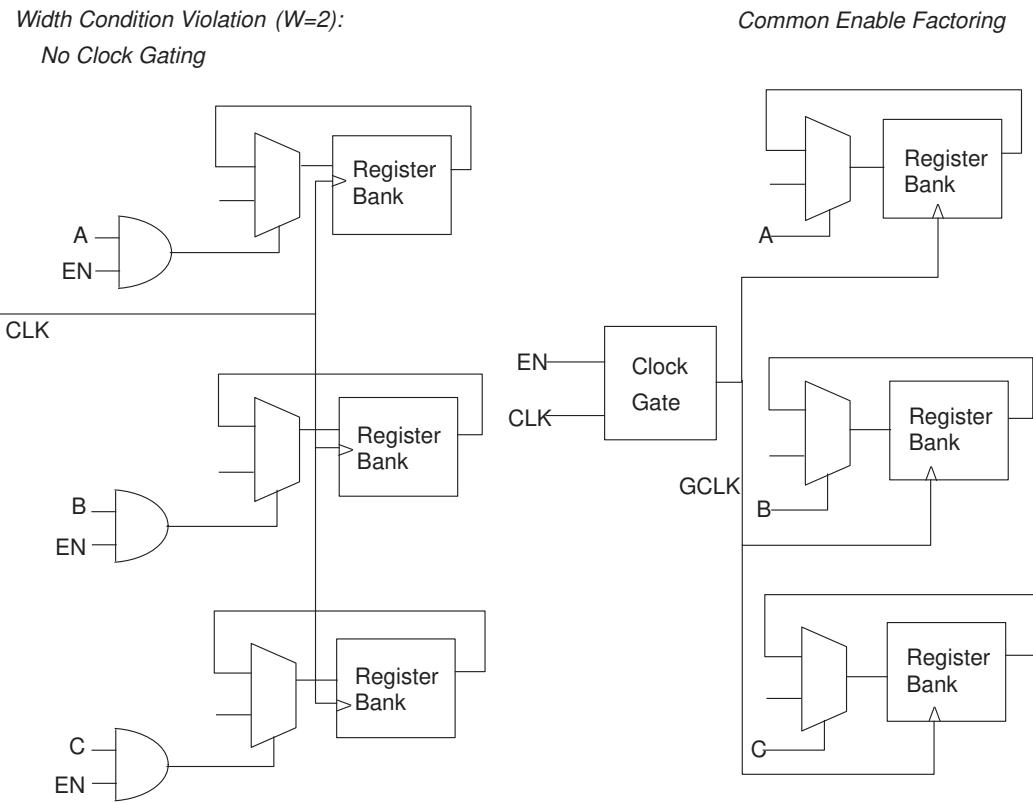
# Read your design
dc_shell> create_clock -name clk -period 10
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating -verbose -gating_elements -gated
dc_shell> report_power
```

Enhanced Register-Based Clock Gating

The regular register-based clock gating requires certain conditions in order for successful implementation. One of these conditions is the minimum bit-width of the register bank to be gated. If the minimum bit-width is less than 3, which is the default, there is no clock-gating opportunity. This width constraint ensures that the overhead of using the clock-gating cell does not overcome the power savings.

The Power Compiler tool can factor out the common enable signal EN shared between three register banks and insert one clock-gating cell for these register banks, which would normally not be clock gated due to the width condition. The result is shown in Figure 36.

Figure 36 Design With Common Enable Signal



The default total minimum bit-width of registers for enhanced clock gating to be implemented is twice that of regular clock gating. Because the default for regular register clock gating is 3, for the enhanced clock gating the register width should be at least $2 * 3$, which is 6.

Enhanced clock gating is the default behavior when you use the `compile_ultra -gate_clock` command.

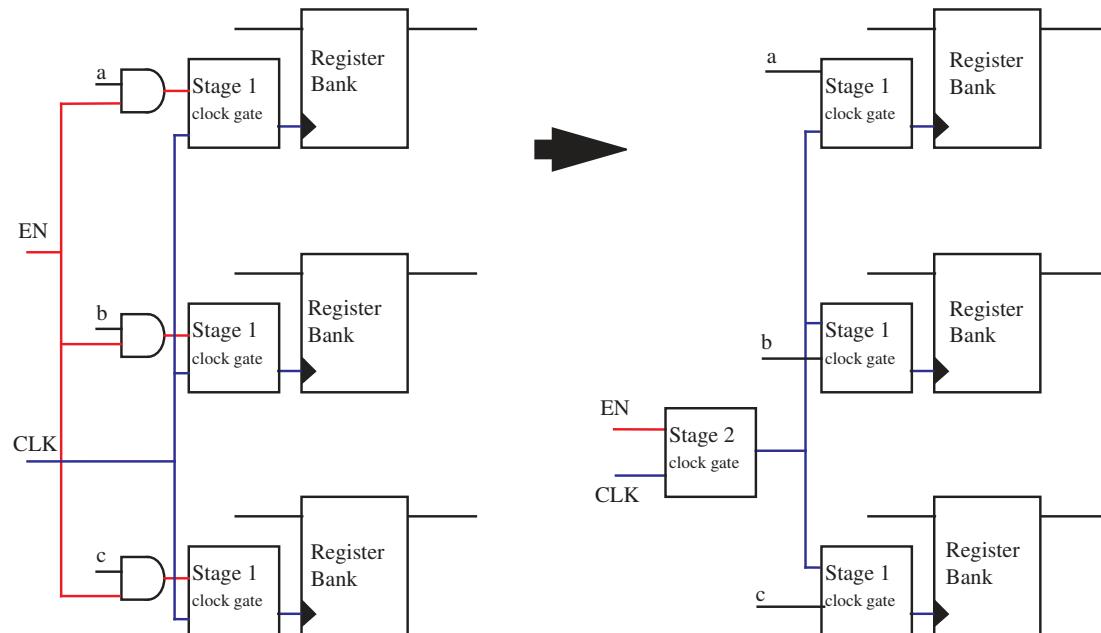
In the following example, automated clock gating with enhanced clock gating is implemented if the clock-gating conditions are met.

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
```

Multistage Clock Gating

When a clock-gating cell drives another or a row of clock-gating cells, it is called multistage clock gating. For additional power savings, the tool identifies common enables and factoring using another clock-gating cell as shown in [Figure 37](#).

Figure 37 Multistage Clock Gating With `set_clock_gating_style -num_stages 2`



To perform multistage clock gating, you should set the maximum number of stages for multistage clock gating by using the `-num_stages` option of the `set_clock_gating_style` command. The default of the `-num_stages` option is 1. After setting the maximum number of stages, use the `compile_ultra -gate_clock` command to perform multistage clock gating.

However, the `compile_ultra` command performs the following additional clock-gate optimization steps during multistage clock gating:

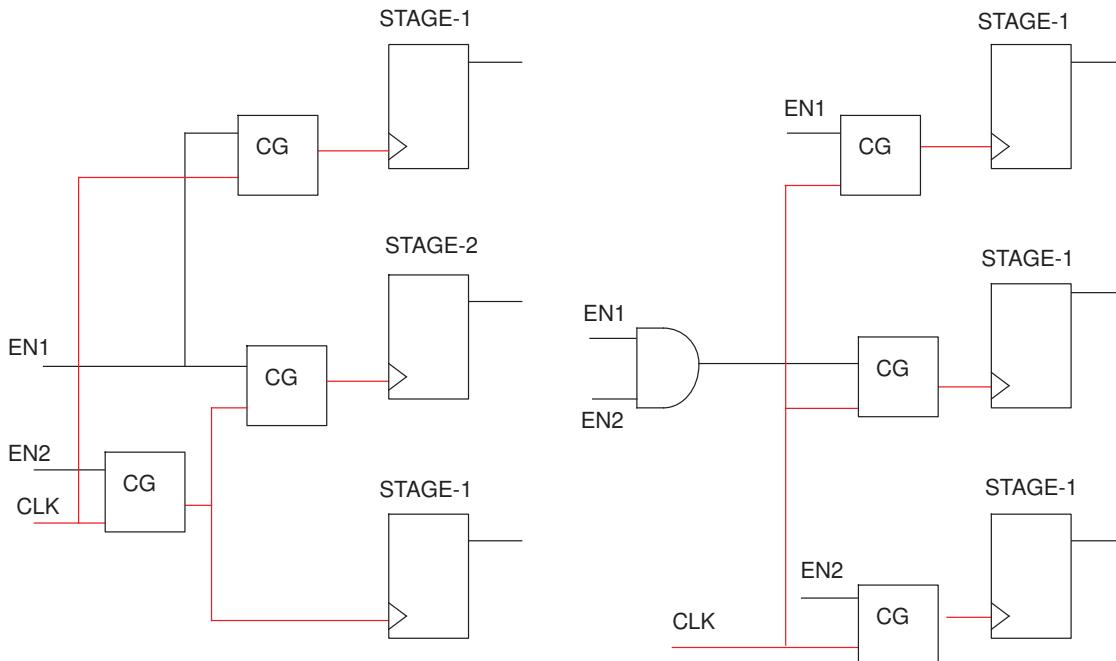
- Reconfiguring the number of clock-gating stages

If you set the `power_cg_reconfig_stages` variable to `true`, the tool reconfigures the number of clock-gating stages. The reconfiguration complies with the value of the `-num_stages` option of the `set_clock_gating_style` command. This is done only on the clock gates inserted by the tool or identified by the tool.

- Balancing the number of clock-gating stages

If you set the `power_cg_balance_stages` variable to `true`, the tool balances the number of the existing clock-gating stages across various register banks. Balanced clock-gate stages ensure uniform clock latency across register banks. [Figure 38](#) shows the transformation for balancing the clock-gating stages.

Figure 38 Balancing the Number of Stages



Multistage Clock-Gating Flow

Follow these steps to build a multistage clock-gating structure for a design that does not have clock-gating cells:

- Set clock-gating styles and directives.

Use the `set_clock_gating_style` command to specify the clock gating stages and other clock gating conditions. You can set the number of stages for multistage clock gating as shown in the following example:

```
set_clock_gating_style -num_stages 5
```

- Read your design.

Read in the design using a read command.

3. Perform multistage clock gating.

Use the `compile_ultra -gate_clock` command.

4. Report the gate elements registers.

Use the `report_clock_gating` command to get the list of cells and the `report_power` command to see the design power after the multistage clock gating.

The following is an example script to perform multistage clock gating using the `compile_ultra -gate_clock` command:

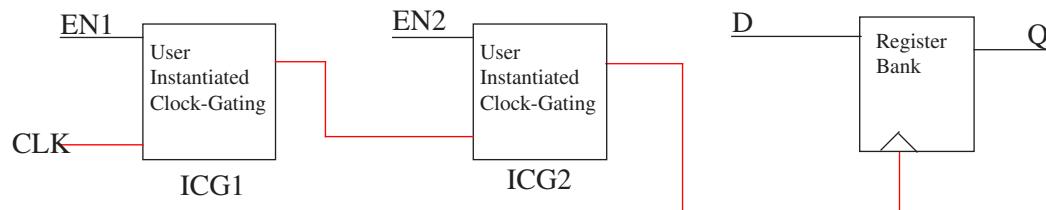
```
# set the target library and the link library

dc_shell> set_clock_gating_style -num_stages 5
dc_shell> read_verilog design.v
dc_shell> create_clock -name clk -period 10
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating -verbose -gating_elements \
           -gated -multi_stage
dc_shell> report_power
```

Clock Gate Merging

When your design has multiple clock-gating cells as shown in [Figure 39](#), the Power Compiler tool can merge two clock-gating cells, into one clock-gating cell.

Figure 39 Two Integrated Clock-Gating Cells

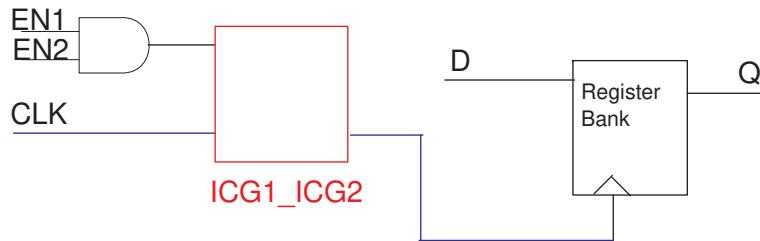


[Example 20](#) shows how to insert, identify, and merge ICG1 and ICG2 clock-gating cells shown in [Figure 40](#):

Example 20 Example to Insert, Identify, and Merge Clock-Gating Cells

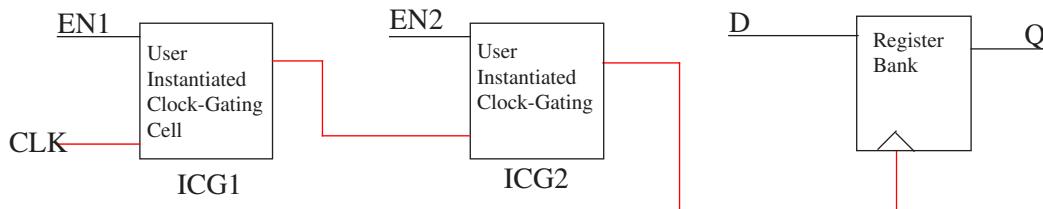
```
dc_shell> set power_cg_auto_identify true
dc_shell> set power_cg_reconfig_stages true
dc_shell> set_clock_gating_style -positive_edge_logic {integrated} \
           -num_stages 1
dc_shell> compile_ultra -gate_clock
```

Figure 40 Two Integrated Clock-Gating Cells Merged by AND Operation of the Enable Inputs



In [Figure 41](#), one of the clock-gating cells in [Figure 39](#) is set for preservation.

Figure 41 One Integrated Clock-Gating Cell With Preservation

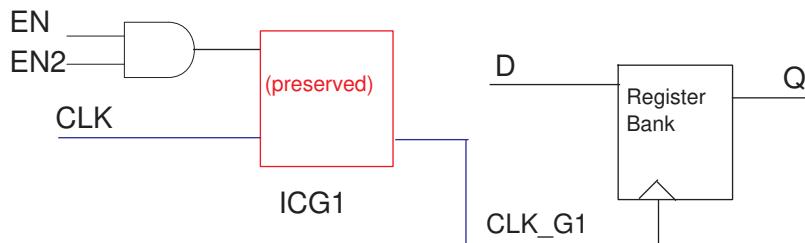


[Example 21](#) shows how to insert, identify, preserve the ICG1 cell, and merge clock-gating cells as show in [Figure 42](#). The preserved clock gate retains its name.

Example 21 Example to Insert, Identify, Preserve, and Merge Clock-Gating Cells

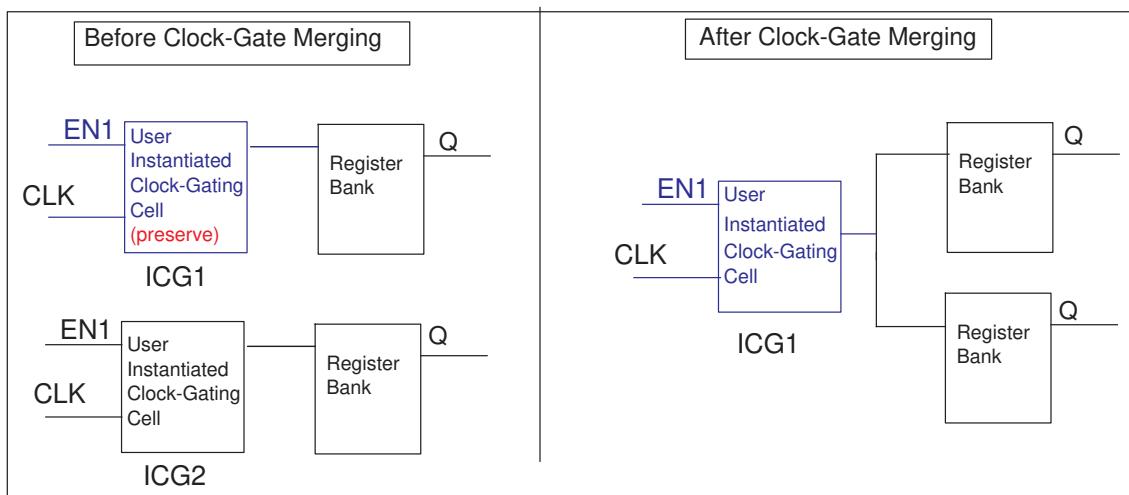
```
dc_shell> set power_cg_reconfig_stages true
dc_shell> set_clock_gating_style -positive_edge_logic {integrated} \
           -num_stages 1
dc_shell> identify_clock_gating
dc_shell> set_preserve_clock_gate [get_cells ICG1]
dc_shell> compile_ultra -gate_clock
```

Figure 42 Merged Clock-Gating Cells With Preserved Cell Name Remaining



In Figure 43, one of the clock gates to be merged has the `pwr_cg_preservation_type` attribute set, and the other does not. The tool merges the clock gates by preserving the cell that has the `pwr_cg_preservation_type` attribute set.

Figure 43 Merging Clock Gates When One of the Identified Clock-Gating Cell Does Not Have the `pwr_cg_preservation_type` Attribute Set



Placement-Aware Clock Gating in Design Compiler Graphical

In the Synopsys physical guidance flow, the Design Compiler Graphical tool can restructure the connection between the integrated clock-gating cell and the registers that it drives so that the clock-gating cell and the registers can be placed close to each other. This restructuring is used by the IC Compiler II tool during placement optimization, which improves the overall timing and area of the design.

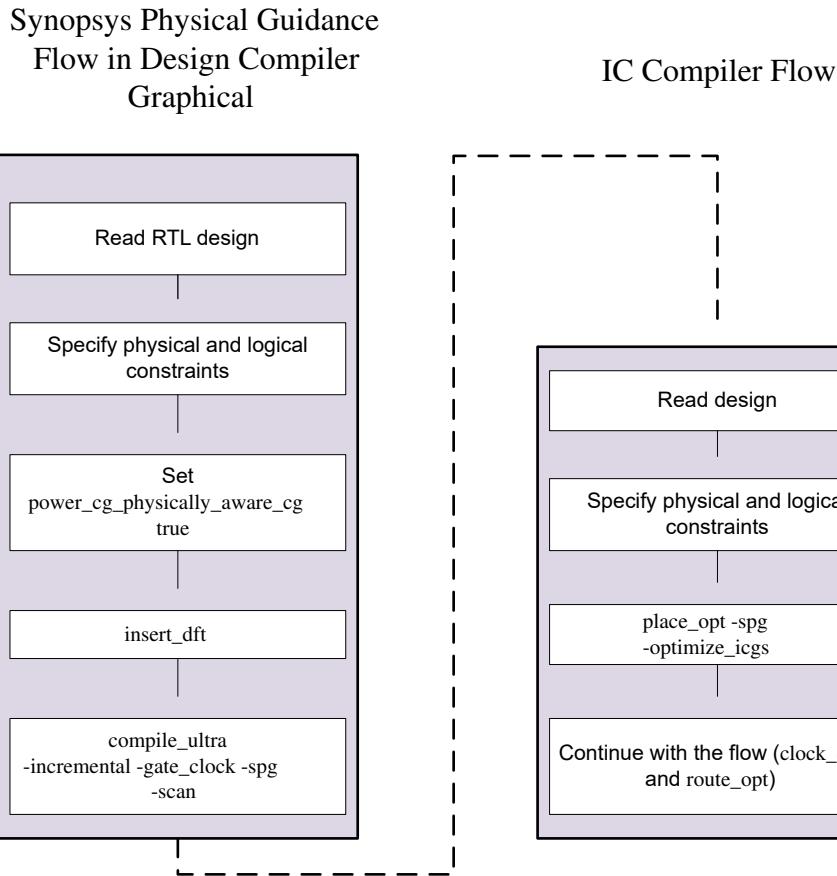
To enable the restructuring of the integrated clock-gating cell and the registers, set the `power_cg_physically_aware_cg` variable to `true`. The default is `false`. When the `power_cg_physically_aware_cg` variable is set to `true`, the annotation of clock-gate latency values is disabled during compile. Any new, tool-inserted clock gates do not have any latency annotation. This also means any latency values set using the `set_clock_latency` command are unused, and the `apply_clock_gate_latency` command has no effect. However, any previously annotated clock-gate latency values on clock-gating cells are left unchanged.

[Figure 44](#) shows the physical guidance flow for placement-aware clock-gating in the Design Compiler Graphical and IC Compiler II tools.

Note:

When placement-aware clock gating is enabled, clock-gating identification is performed during the `compile_ultra` command to obtain a better correlation with the IC Compiler II tool. This is independent of the value of the `power_cg_auto_identify` variable.

Figure 44 Physical Guidance Flow for Placement-Aware Clock Gating



Clock Gating Multibit Registers

The Power Compiler tool supports insertion of clock-gating cells on multibit registers. The enable pins for all the registers must be the same for clock gating to occur. All the clock-gating commands are supported for multibit registers.

To enable clock gating on multibit registers, set the `hdlin_infer_multibit` variable to `default_all`. For more information, see [SolvNetPlus article 000025387, “Multibit Register Synthesis and Physical Implementation Application Note.”](#)

To set the maximum fanout value for the clock-gating cells, use the `set_clock_gating_style -max_fanout` command. When calculating the `max_fanout` value for multibit registers, use the required register count as the fanout value.

For example, if you specify `set_clock_gating_style -max_fanout 4`, the `compile_ultra -gate_clock` command inserts a clock-gating cell that can drive up to

4 registers whether they are 4 single-bit or 4 multibit registers or a combination of the two types of registers.

It is important to note that while `max_fanout` is calculated as the actual load on the clock-gating cell, the `min_bitwidth` value is the minimum number of bits to either gate or ungate whether it is a multibit register or not. For example, if you set the `min_bitwidth` value to 3, you can clock gate four single-bit registers if they share the same clock and enable lines. The multibit mapping feature converts the four single-bit registers into a single 4-bit multibit register. Both configurations—the four single bit registers or the single 4-bit multibit register—satisfy the minimum bitwidth setting of three.

The `report_clock_gating` command generates a report that includes details of the decomposition of multibit registers.

Note:

The Power Compiler tool does not support self-gating on multibit registers.

Performing Clock-Gating on DesignWare Components

The Power Compiler tool provides the ability to perform clock gating on DesignWare components instead of treating them as black box cells. The `compile_ultra -gate_clock` command performs clock gating on DesignWare components, by default.

The following example script performs clock gating on DesignWare components:

```
set target_library [list my_lib.db cg_integ_pos.db]
set synthetic_library dw.foundation.sldb
set link_library [list "*" my_lib.db
dw.foundation.sldb cg_integ_pos.db]
set_clock_gating_style -minimum_bitwidth 1 -sequential_cell latch \
    -positive_edge_logic {integrated:CGLP} # Optional
read_verilog cpurd_fifo.v
write -format verilog -hierarchy -output elab.v
compile_ultra -gate_clock
insert_dft
write -format verilog -hierarchy -output comp.v
```

You can view the DesignWare clock-gated registers using the `report_clock_gating -gated` command. The DesignWare clock gates are designated by a (*) in the report.

Reporting Clock Gates

The `report_clock_gating` command reports the clock-gating cells and the registers with and without clock-gating signals in the current design. To see the dynamic power savings due to clock-gate insertion, use the `report_power` command.

For more information, see the following topics:

- [The report_clock_gating Command](#)

The report_clock_gating Command

The Power Compiler tool provides detailed reporting of clock gates with the `report_clock_gating` command. As a prerequisite, clock gates must be identified by using one of the following methods:

- Set the `power_cg_auto_identify` variable to `true` before reading in the design.
- Execute the `identify_clock_gating` command without specifying any options.

If you use the `report_clock_gating` command without any options, the report includes a summary table of the clock-gating elements and a table that shows the origins of the clock gates.

An example of the default report is shown in [Example 22](#).

Example 22 Default Clock-Gating Report

```
dc_shell> report_clock_gating
*****
Report : clock_gating
Design : low_design
Version: ...
Date   : ...
*****
Clock-Gating Summary
+-----+
| Number of clock gating elements | 2 |
| Number of gated registers     | 8 (80.00%) |
| Number of ungated registers   | 2 (20.00%) |
| Total number of registers     | 10 |
+-----+
```

Clock-Gating Report by Origin		
	Actual (%)	Count
Number of tool-inserted clock gating elements	1 (50.00%)	1
Number of pre-existing clock gating elements	1 (50.00%)	1
Number of gated registers	8 (80.00%)	8
Number of tool-inserted gated registers	4 (40.00%)	4
Number of pre-existing gated registers	8 (80.00%)	8
Number of ungated registers	2 (20.00%)	2
Number of registers		10

If the design contains multibit registers, the default report also contains information about the number and percentage of gated and ungated bits with respect to both tool-inserted and previously existing clock gates, as shown in [Example 23](#). The Actual Count column represents the count of the cells in the design. The Single-bit Equivalent column represents the register count if every multibit register is converted to an equivalent number of single-bit registers.

Example 23 Clock-Gating Report for a Design With Clock-Gated Multibit Registers

Clock Gating Multibit Decomposition			
	Actual	Single-bit	
	Count	Equivalent	
Number of Gated Registers			
1-bit	0	0	
4-bit	1	4	
Total	1	4	
Number of Ungated Registers			
1-bit	1	1	
4-bit	0	0	
Total	1	1	
Total Number of Registers			
1-bit	1	1	
4-bit	1	4	
Total	2	5	

The percentages in the report are calculated with respect to the total number of registers or bits. In a multistage design, registers and bits can be simultaneously gated by both

tool-inserted and previously existing clock gates. For this reason, some of the reported percentages might not add up to 100 percent.

When you use the `-gating_elements` option, the report includes the label `(d)` next to the name of a clock-gating cell to indicate that the cell or its parent hierarchical cell has been marked by the `set_dont_touch` command. This means that this clock-gating cell is not modified or removed. [Example 24](#) shows an example report using the `-gating_elements` option. The report includes a similar section for each clock gating bank.

Example 24 Clock-Gating Report Using the `-gating_elements` Option

```
-----
          Clock-Gating Cell Report
-----
Clock Gating Bank : clk_gate_out1_reg (ss_hvt_0v70_125c: 0.7)
-----
STYLE = latch, MIN = 2, MAX = unlimited, HOLD = 0.00, OBS_DEPTH = 5

INPUTS :
  clk_gate_out1_reg/CLK = clk
  clk_gate_out1_reg/EN = N6
  clk_gate_out1_reg/TE = test_se
OUTPUTS :
  clk_gate_out1_reg/ENCLK = net107
-----
```

[Example 25](#) shows an example report using the `-ungated`, `-gated`, `-gating_elements`, and `-verbose` options. Tables display all the ungated and gated registers in the current design. The `-ungated` option shows the specific registers that are not clock-gated and the reasons for not gating them.

Example 25 Clock-Gating Report With Gated and Ungated Elements

```
-----
          Clock Gating Cell Report
-----
Clock Gating Bank : clk_gate_C7
-----
STYLE = none, MIN = 3, MAX = 2048, HOLD = 0.0, SETUP = 1.3, OBS_DEPTH = 5
TEST INFORMATION :
  OBS_POINT = NO, CTRL_SIGNAL = scan_enable, CTRL_POINT = none
INPUTS :
  clk_gate_C7/CLK = clk
  clk_gate_C7/EN = xi
OUTPUTS :
  clk_gate_C7/ENCLK = xclk
RELATED REGISTERS :
  q4_reg[3]
  q4_reg[2]
  q4_reg[1]
  q4_reg[0]
-----
```

Gated Register Report		
Clock Gating Bank	Gated Register	
clk_gate_C7	q4_reg[0] q4_reg[1] q4_reg[2] q4_reg[3]	
clk_gate_q3_reg	q3_reg[0] q3_reg[1] q3_reg[2] q3_reg[3]	
Ungated Register Report		
Ungated Register	Reason	What Next ?
q1_reg	Min bitwidth not met	
q2_reg	Min bitwidth not met	
q5_reg	Min bitwidth not met	

Clock Gating Summary		
Number of clock gating elements	0	
Number of gated registers	0 (0.00%)	
Number of ungated registers	4 (100.00%)	
Total number of registers	4	

[Example 26](#) shows a report generated with the `-multi_stage` and `-no_hier` options for a hierarchical multistage clock gated design. A multistage clock gate is a clock-gating cell that is driving another clock-gating cell.

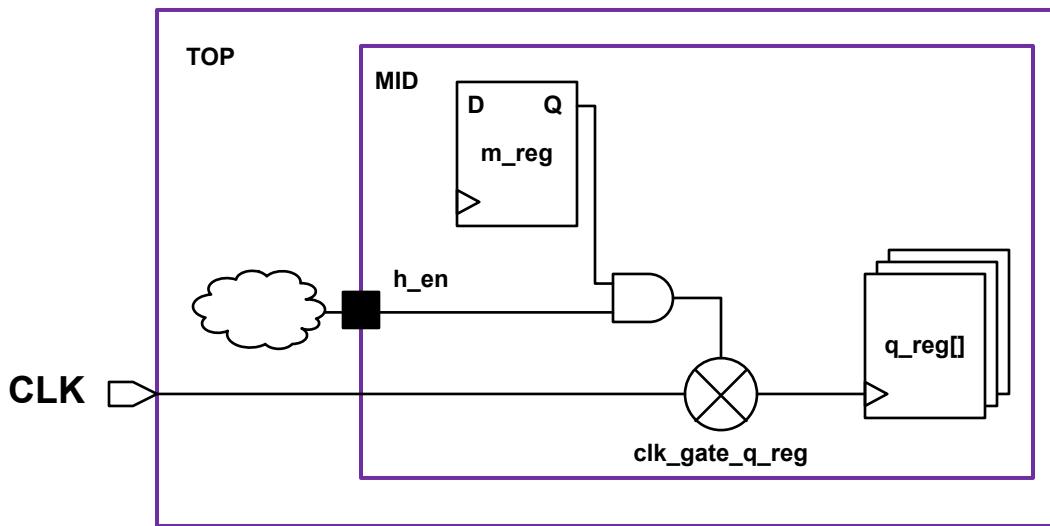
Example 26 Clock-Gating Report Using the `-no_hierarchy` and `-multi_stage` Options

Clock Gating Summary

Number of clock gating elements	6
Number of gated registers	16 (72.73%)
Number of ungated registers	6 (27.27%)
Total number of registers	22
Number of multi-stage clock gates	2
Average multi-stage fanout	2.0
Number of gated cells	16
Maximum number of stages	3
Average number of stages	2.2

Figure 45 shows the enable conditions of a clock-gating cell. The enable condition for the `clk_gate_q_reg` clock-gating cell is represented by the RTL invariant object names during synthesis. In this example, these objects are the sequential output pins (the Q pin coming from the `m_reg` register) and hierarchical input pins (the `h_en1` pin). These objects are put into one basic logical expression representing when the clock signal is disabled.

Figure 45 Example of Enable Conditions for Clock-Gating Cells



The `report_clock_gating -enable_conditions` command prints a report, as shown in Example 27.

Example 27 Example of Clock Gating Report for Enable Conditions

```
*****
Report : clock_gating
    -enable_conditions
Design : top
Version: ...
Date   : ...
*****
```

Enable Conditions Report

Clock Gating Bank		Gated Register
MID/clk_gate_q_reg		MID/q_reg

Enable condition:
MID/h_en & MID/m_reg/Q

The `all_clock_gates` command includes the `-origin origin_spec` option that specifies the origins of the cells returned. Accepted values include `pre_existing` and `tool_inserted`. The `-origin` option can also be combined with other existing options.

- `pre_existing`: Use this argument to return the origin of all the preexisting clock gates. The tool displays an information message when the `pre_existing` argument is used without initially running the `identify_clock_gates` command.

[Figure 46](#) and [Figure 47](#) show the `all_clock_gates` command output with and without initially running the `identify_clock_gates` command:

Figure 46 Output of the `all_clock_gates` command after initially running the `identify_clock_gates` command

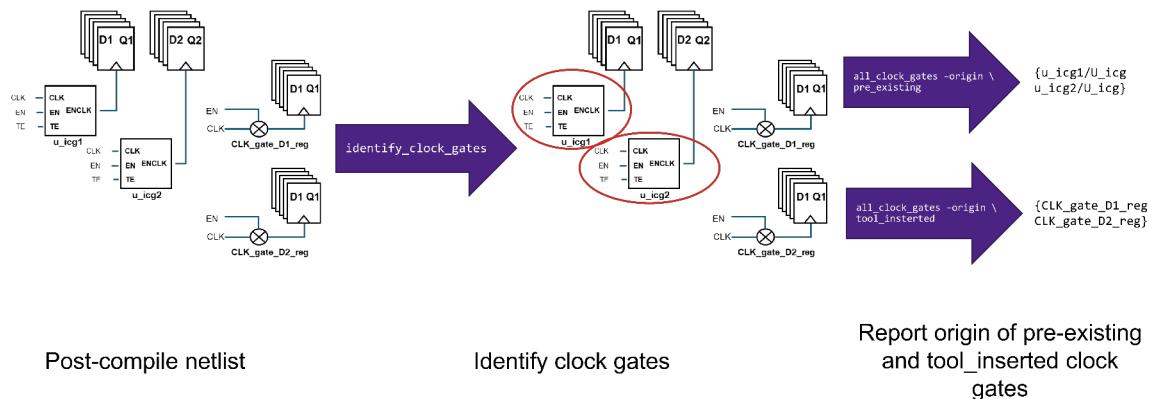
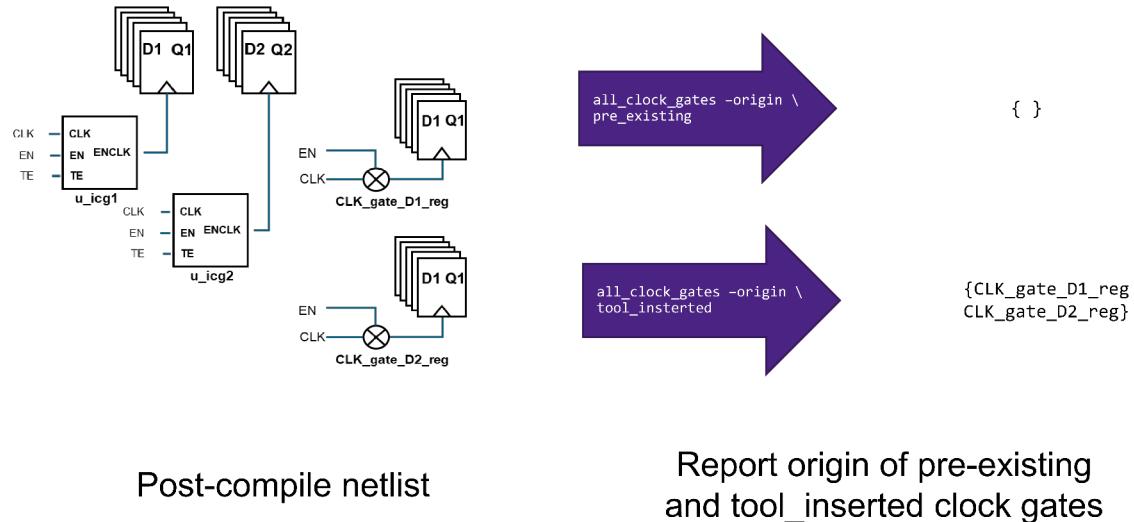


Figure 47 Output of the all_clock_gates command without initially running the identify_clock_gates command



- **tool_inserted:** Use this argument to return the origin of all the clock gates inserted by the tool.

The `report_clock_gating` command issues a comprehensive summary of the reasons for not gating and the percentage of registers being impacted by each reason in the design. The table is sorted by the frequency of the ungated reason, from highest to lowest. The tool does not support self gating.

Figure 48 shows the histogram output of the `report_clock_gating -ungated` command:

Figure 48 Histogram Output

Tool-inserted ungated reasons				
Reason for not gating	Regs	Regs %	Bits	Bits %
Min bitwidth not met	4	50.00	4	50.00
Always enabled register	2	25.00	2	25.00
User excluded register	2	25.00	2	25.00

The histogram report includes the following parameters:

- Reason for not gating: The reason for not gating the register
- Regs: Number of ungated registers for each reason
- Regs %: Percentage of registers ungated due to the specific reason
- Bits: Number of ungated bits for each reason
- Bits %: Percentage of bits ungated due to the specific reason

8

Self-Gating

Self-gating is an advanced clock-gating technique that reduces dynamic power consumption. Self-gating turns off the clock signal during specific clock cycles when the data in the register is unchanged.

For more information, see the following topics:

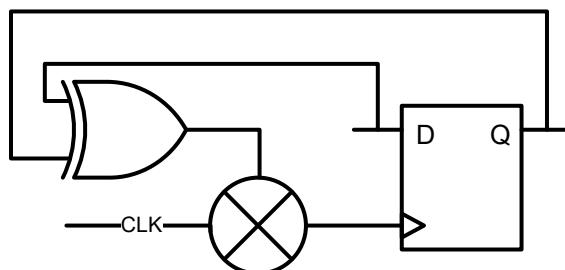
- [Self-Gating Concepts](#)
 - [Self-Gating Flows](#)
 - [Library Requirements for Self-Gating](#)
 - [Inserting Self-Gates](#)
 - [Querying and Reporting Self-Gates](#)
-

Self-Gating Concepts

The self-gating technique reduces the dynamic power of a design by turning off the clock signal of registers during clock cycles when the data in the register remains unchanged. For example, an XOR gate can be used to compare the data stored in the register with the data arriving at the data pin of the register. The XOR gate output controls the enable condition for gating. If the data is unchanged, the unnecessary clock cycles are gated by the output of the XOR gate. [Figure 49](#) shows the XOR gate that generates the enable signal.

Self-gating is available in topographical mode in the Design Compiler and Design Compiler NXT tools.

Figure 49 XOR Self-Gating Cell



By default, the tool supports self-gating only on registers that are not gated.

Registers with an enable condition that cannot be inferred from the existing logic can only be gated using the self-gating technique. They cannot be gated using traditional clock gating. By default, the tool supports self-gating only on registers that are not gated. You can use the `set_self_gating_options` command to allow self-gating on these registers. However, the time duration that the clock signal is turned off might increase for these registers.

To ensure QoR improvements, the self-gating algorithm takes timing and power into consideration. The tool employs self-gating if both of these conditions are true:

- There is enough timing slack available at the register's data pin.

For designs with multiple scenarios, the tool considers the timing of the worst case among active scenarios that are enabled for both setup and dynamic power. The tool issues a PWR-949 information message that reports the scenario on which self-gating is based.

- Internal dynamic power of the circuit is reduced.

For designs with multiple scenarios, the tool uses the average internal dynamic power among active scenarios that are enabled for both setup and dynamic power.

If the design contains multiple scenarios but none of them are enabled for setup or dynamic power, the tool does not perform self-gating and issues a PWR-948 information message.

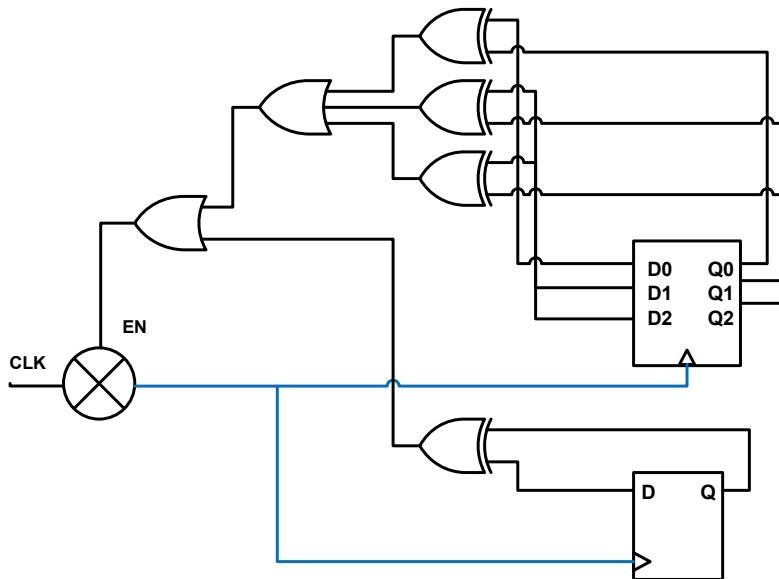
The Power Compiler tool does not support the following types of sequential cells for self-gate insertion:

- Level-sensitive sequential cells
- Level-sensitive scan design registers
- Master-slave flip-flops
- Retention registers
- Single-bit and multibit registers that belong to shift registers
- Multibit registers with multiple clock pins
- Cells that are scan-stitched prior to self-gating

To minimize the area and power overhead, a self-gating cell can be shared across a few registers by creating a combined enable condition with a tree of comparator gates. If the self-gated registers are driven by synchronous set or reset signals, these signals are also included in the construction of the enable signal so that the circuit remains functionally unchanged. [Figure 50](#) is an example of a self-gating cell that is shared across two registers (4 bits). One of the self-gated registers is a multibit register and the other

register is a single-bit register. The tool can also self-gate a group of multibit registers or a group of single-bit registers.

Figure 50 Shared Comparator Cells for Self-Gating



If the following conditions are met, two or more registers can be gated by the same self-gating cell:

- The registers belong to the same hierarchy
- The registers belong to the same clock domain
- The set and reset signals are driven by the same type of signals, either one of the following:
 - Synchronous set and synchronous reset
 - Asynchronous set and asynchronous reset

The Power Compiler tool supports using XOR, OR, and NAND gates for comparator gates in a self-gating flow. OR and NAND gates are typically smaller and faster than XOR cells. However, the optimal choice in terms of power consumption depends on factors such as the static probability of the data pin, its toggle rate, and the power consumption of the specific library cells involved (the registers being gated, the self-gates, and the combinational cells used as comparator cells). You can allow the tool to select the type of comparator cell to use in self-gating or specify the type of comparator cell to use.

Self-Gating Flows

During self-gating, the Power Compiler tool identifies registers where self-gate insertion can potentially save dynamic power without degrading the timing. Dynamic power is calculated using the switching activity annotated on the design. If the switching activity is not available, the tool uses the default activity.

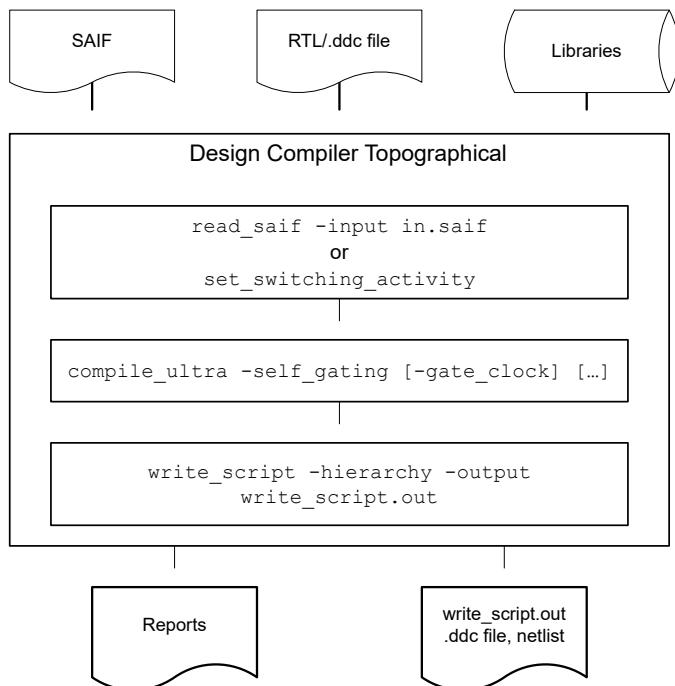
By default, the tool groups registers to create self-gating banks with a minimum size of four bits and a maximum size of eight bits.

In addition, the Design Compiler NXT tool groups registers with similar D pin toggle rates, which limits the detrimental power effects of high toggle rates.

The self-gating comparator cells are inserted without a hierarchical wrapper.

[Figure 51](#) illustrates the general flow for self-gating. The general flow uses logic libraries and a SAIF, RTL, or .ddc file.

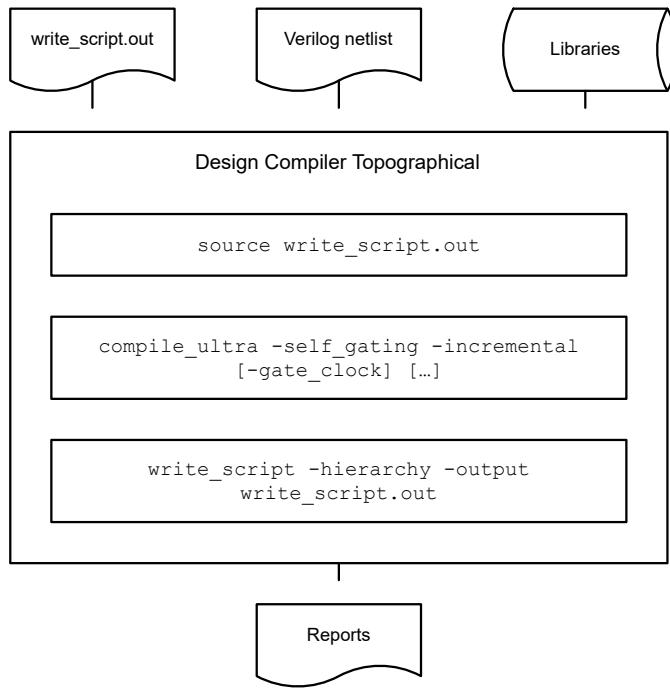
Figure 51 General Self-Gating Flow



When an ASCII netlist with self-gating cells is read into the Power Compiler tool, all attribute information is lost and the tool does not recognize the self-gating cells for reporting or optimization. The Power Compiler tool supports the self-gating ASCII flow using the `write_script` command, as shown in [Figure 52](#). Use the `write_script -hierarchy -output file_name` command to save the current attributes of the design. When the design is read back into the tool, use the `source` command to source the file.

written by the `write_script` command. This sets all the required attributes on the design, including the self-gating cells, for reporting and optimization.

Figure 52 *Self-Gating ASCII Flow*



Library Requirements for Self-Gating

To perform self-gating, the logic library should contain XOR, OR, and AND gates for the corresponding operating conditions. The integrated clock-gating cells in the library that have the following configurations are used as self-gating cells.

- Sequential cell: latch
- Control point: before
- Control signal: scan_enable
- Observation point: none

If the library does not contain cells with these configurations for the corresponding operating conditions, the tool does not insert self-gating cells. If an integrated clock gate compatible with self-gating is specified through the `set_clock_gating_style` command, the tool uses the same integrated clock gate or the clock gate that is most similar to the one specified.

Inserting Self-Gates

To insert self-gates, use the `compile_ultra -self_gating` command. Perform the following steps before you run the `compile_ultra` command:

1. Apply switching activity by using the `read_saif` or `set_switching_activity` command.
 2. (Optional) Specify self-gating settings by using the `set_self_gating_options` command.
 3. (Optional) Override the default selection of self-gating objects by using the `set_self_gating_objects` command.
-

Specifying Objects for Self-Gating

To specify objects for self-gating, use the `set_self_gating_objects` command. You can specify registers, hierarchical cells, power domains, or designs. The options are as follows:

- `-force_include object_list`
Specifies a list of objects to be self-gated, regardless of the `set_self_gating_options` command settings.
- `-exclude object_list`
Specifies a list of objects to be excluded from self-gating.
- `-include object_list`
Specifies a list objects for which self-gating honors the `set_clock_gating_options` command settings. This is the default for all registers in the design.
- `-undo object_list`
Removes all inclusion, exclusion, and forced criteria on the listed objects.
- `-type combinational_logic_type`
Specifies the type of combinational cells to use. This option is available only in the Design Compiler NXT tool. Valid settings are `xor`, `or`, `nand`, and `auto` settings to allow the use of different types of comparator cells. The default is `auto`, in which case the tool automatically decides which comparison logic to use based on switching activity information. This might help to reduce the area while optimizing power. You must use this option with the `-include` or `-force_include` options.

The Design Compiler tool always uses XOR comparator cells for self-gating and does not support the `-type` option.

The following command excludes the D_OUT_reg register bank of the MID subdesign from self-gating:

```
dc_shell-topo> set_self_gating_objects -exclude MID/D_OUT_reg[*]
```

Specifying Options for Self-Gating

To specify conditions for self-gating and define the interaction with clock gating, use the `set_self_gating_options` command. The options are as follows:

- `-min_bitwidth bitwidth`
Controls the minimum size for self-gating banks. The default is 4.
- `-max_bitwidth bitwidth`
Controls the maximum size for self-gating banks. The default is 8.
- `-interaction_with_clock_gating interaction_type`
Registers gated by user-instantiated clock-gating cells are candidates for self-gating. Valid arguments are `none` (skip registers gated by tool-inserted clock gates), `insert` (the default; insert self-gates on gated registers), and `collapse` (collapse tool-inserted clock gates if they are in the same hierarchy).

The following command specifies to insert self-gates for a minimum of two bits and a maximum of nine bits:

```
dc_shell-topo> set_self_gating_options -min_bitwidth 2 -max_bitwidth 9
```

Querying and Reporting Self-Gates

Use the `all_self_gates` command to get a collection of self-gating cells or pins of self-gating cells in the current design. For example, the following command returns the self-gating cells that gate registers clocked by CLK:

```
dc_shell-topo> all_self_gates -clock CLK
```

Use the `report_self_gating` command to report the number of registers with self-gates and optional information about registers without self-gates in the current design.

[Example 28](#) shows the report generated by the `report_self_gating` command when no option is specified.

Example 28 Report Generated by the report_self_gating Command

Self-Gating Summary			
Number of self gating cells	7		
Number of self gated registers	50 (50.00%)		
Number of registers not self-gated	50 (50.00%)		
Total number of registers	100		

Self Gating Multibit Decomposition			
	Actual Count	Single-bit Equivalent	
Number of Self Gated Registers			
1-bit	3	3	
4-bit	3	12	
Total	6	15	
Number of Registers not Self-Gated			
1-bit	0	0	
4-bit	0	0	
Total	0	0	
Total Number of Registers			
1-bit	3	3	
4-bit	3	12	
Total	6	15	

Use the `report_self_gating -ungated` command to obtain a report that describes why self-gating did not occur and suggests how to get self-gating to occur on these registers. The report is similar to [Example 29](#).

Example 29 Report From the report_self_gating -ungated Command

Ungated Register Report		
Ungated Register	Reason	What Next?
y_reg[9]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[8]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[7]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[6]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[5]	Self gating creates negative slack on path	Relax timing constraints on this path

Self Gating Summary		
Number of self-gating cells	0	
Number of self gated registers	0 (0.00%)	
Number of registers not self-gated	5 (100.00%)	
Total number of registers	5	

Use the `report_self_gating -gated` command to display information about self-gating banks. The report is similar to [Example 30](#).

Example 30 Report From the report_self_gating -gated Command

Self-Gated Register Report		
Self-Gating Bank		Gated Register
self_gate_q_reg		q_reg[0] q_reg[1] q_reg[2] q_reg[3]
Total		4
self_gate_y_reg		y_reg[0] y_reg[1] y_reg[2] y_reg[3]
Total		4
Sum Total	2	8

9

Power Optimization

The Power Compiler tool performs additional steps to optimize the design for dynamic and leakage power.

For information about power optimization, see the following topics:

- [Overview](#)
 - [Gate-Level Power Optimization](#)
 - [Enabling Power Optimization](#)
 - [Performing Power Optimization](#)
-

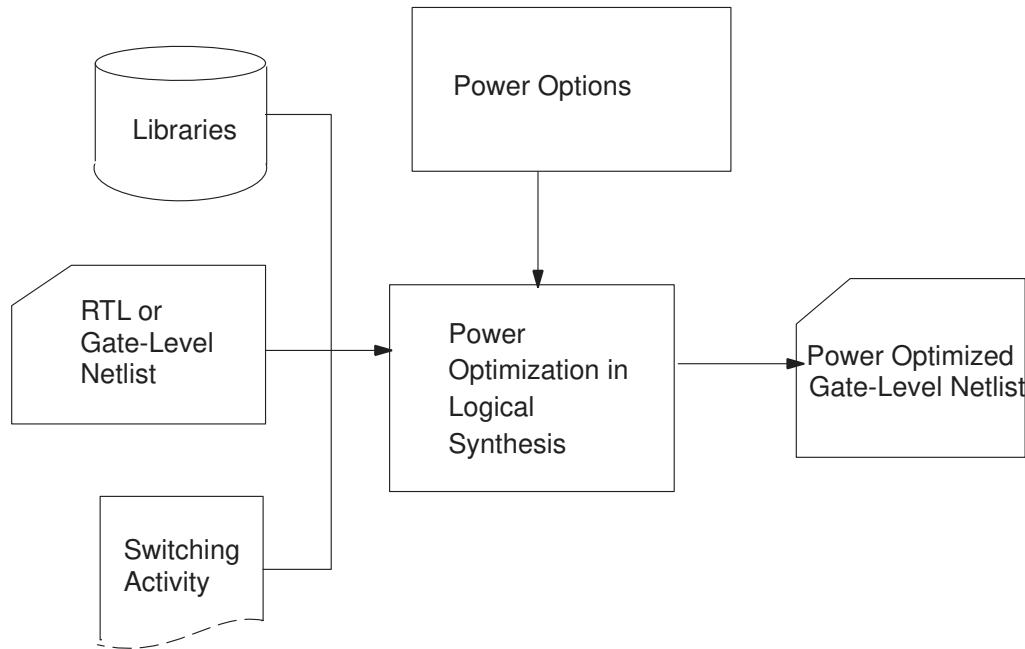
Overview

Transistor threshold voltage has an exponential effect on the transistor's leakage power. Minimizing leakage power is an important goal of IC design.

Every design contains both critical and noncritical timing paths. Using a lower-speed cell on a noncritical path does not affect the performance of the design. A slower cell usually allows the use of a higher threshold voltage, which reduces leakage power dramatically. Optimizing high-speed and low-speed cells on different timing paths leads to a balanced design with high performance and low leakage power.

[Figure 53](#) illustrates the inputs and outputs of the gate-level power optimization flow.

Figure 53 Power Optimization Flow



The inputs for gate-level power optimization are as follows:

- RTL or gate-level netlist and optional floor plan. This netlist is not optimized for power.
- Power options to enable power optimization
- Libraries

The Power Compiler tool selects different library cells to modify the netlist for power optimization. Providing libraries with multiple threshold voltages is highly recommended to facilitate leakage optimization.

- Switching activity information, which is required for dynamic and total power optimization, and recommended for high accuracy in leakage optimization.

The output of gate-level power optimization is a new gate-level netlist that has been optimized for power. The optimization is implemented with the `compile` or `compile_ultra` commands.

Gate-Level Power Optimization

To perform power optimization, the Power Compiler tool reduces power consumption on paths with positive timing slack. The more paths in the design with positive slack, the more opportunity exists for the tool to reduce power consumption by using low-power cells.

Designs with excessively restrictive timing constraints have little or no positive slack to trade for power reduction.

Designs that have opaque cells such as RAM and ROM cells and customized subdesigns that have the `dont_touch` attribute can benefit from power optimization.

To enable total power optimization, set the `compile_enable_total_power_optimization` variable to `true` (the default is `false`). This feature is available only in the Design Compiler NXT tool in the Synopsys physical guidance flow. With total power optimization, the tool uses placement and sizing techniques to reduce power consumption. For better accuracy, annotate the switching activity from RTL simulation.

Leakage Power Optimization

Leakage power optimization is an additional step to timing optimization. During leakage power optimization, the tool tries to reduce the leakage power of the design without affecting the performance. Leakage power optimization is performed on paths that are not timing-critical. When the target libraries are characterized for leakage power and contain cells characterized for multiple threshold voltages, the tool uses library cells with appropriate threshold voltages to reduce the leakage power of the design.

Dynamic Power Optimization

Dynamic power optimization is an additional step to the timing optimization. After the optimization, the design consumes less dynamic power without affecting the performance. Dynamic power optimization requires information about the switching activity. Optimizing dynamic power incrementally provides better QoR and take less runtime.

Annotating the correct switching activity information by using a SAIF file improves the dynamic power optimization. You can annotate switching activity in the following ways:

- Read the SAIF file

Use the `read_saif` command to read a SAIF file to annotate the switching activity information on nets, pins, ports, and cells in the design.

- Use the `set_switching_activity` command

You can also use the `set_switching_activity` command to annotate the switching activity information.

If switching activity is not annotated, the default toggle rate is applied to the primary inputs and outputs of opaque cells. The Power Compiler tool propagates the default toggle rate throughout the design. The propagated toggle rates are used for dynamic power optimization.

High-Effort Power Optimization

In the Design Compiler NXT tool, you can specify high-effort total power optimization by using the `set_compile_power_high_effort` command. The tool performs aggressive optimization to reduce power at the `compile_ultra -spg`, `compile_ultra -incremental -spg`, and `optimize_netlist -area` commands. You can use the `set_compile_power_high_effort` command only in physical guidance mode.

High-effort total power optimization might increase runtime and might affect other QoR metrics such as area and timing.

The `-total` option of the `set_compile_power_high_effort` command requires a Power Compiler license. If you set this option to `true` (the default is `false`), the tool ignores the `-leakage` option. The `-leakage` option does not require a Power Compiler license.

Enabling Power Optimization

Leakage power optimization is automatically enabled for all Design Compiler tools except DC Expert. When using the DC Expert tool, use the following command to enable leakage optimization:

```
set_leakage_optimization true
```

To enable dynamic optimization in all tools, use the following command:

```
set_dynamic_optimization true
```

When both leakage and dynamic power options are enabled in the DC Expert tool, the tool performs leakage power optimization.

The following example script shows the default usage model for power optimization.

```
# Specify all multivoltage threshold libraries
set_app_var target_library "hvt.db nvt.db lvt.db"
set_app_var link_library "* $target_library"

read_verilog rtl.v
link
compile_ultra
report_power
```

Note:

The `report_power` and `report_constraint` commands use state-dependent information to calculate leakage power.

Leakage Optimization for Multicorner-Multimode Designs

For multicorner-multimode designs, you can specify the leakage power optimization for specific scenarios, using the `set_scenario_options` command.

For more information about leakage power optimization for multicorner-multimode design, see [Power Optimization in Multicorner-Multimode Designs](#).

Leakage Power Optimization Based on Threshold Voltage

Leakage power optimization can use single threshold voltage or multiple threshold voltage libraries. However, multiple threshold voltage libraries can save more leakage power.

Leakage power is very sensitive to threshold voltage. The higher the threshold voltage, the lower the leakage power. On the other hand, the lower the threshold voltage, the faster the timing.

For designs that have strict timing constraints, you can optimize for leakage power only on the paths that are not timing-critical by using the higher threshold-voltage cells from the multiple threshold voltage libraries. If the design has a relatively easy-to-meet timing constraint, you might have a large number of low threshold voltage cells in the design, resulting in higher leakage power consumption. One way to avoid this situation without having to change your target library settings is to use the `set_multi_vth_constraint` command to specify a low percentage usage value for the lower threshold voltage cells. For optimum results, start by specifying 1 to 5 percent of the number of cells in the design for the low threshold voltage cells. Gradually increase the percentage until the timing constraint is met. With this technique, the design can meet the timing constraint while minimizing the leakage power consumption.

Multiple Threshold Voltage Library Attributes

For accurate multiple threshold voltage leakage optimization, define threshold voltage groups in the libraries. Use the `set_attribute` command and add the following attributes:

- Library-level attribute:

```
default_threshold_voltage_group :string;
```

- Library-cell-level attribute:

```
threshold_voltage_group :string;
```

With these attributes, the threshold voltages are differentiated by the string you specify. When the library has at least two threshold voltage groups, or if you have defined threshold voltage groups for your library cells using the `set_attribute` command, the library cells are grouped by the threshold voltage.

The `set_multi_vth_constraint` Command

Use the `set_multi_vth_constraint` command to set the multiple threshold voltage constraint. This command has options to specify the constraint in terms of area or number of cells of the low threshold voltage group. You can also specify whether this constraint should have higher or lower priority than the timing constraint.

The `set_multi_vth_constraint` command supports the `-type` option to specify the type of the constraint. When you specify `-type hard`, the Power Compiler tool tries to meet this constraint, even if this results in timing degradation. When you specify `-type soft`, the tool tries to meet this constraint without degrading the timing.

Note:

The `-type soft` option is supported only in Design Compiler topographical mode.

When calculating the percentage of low threshold voltage cells in the design, the tool does not consider opaque cells. To consider these cells in the percentage calculation, specify the `-include_blackboxes` option.

After synthesis, use the `report_threshold_voltage_group` command to report the percentage of the total design, by cell count and by area, that is occupied by the low-threshold voltage cells.

In the following example, the maximum percentage of low threshold voltage cells in the design is set to 15 percent. When the tool tries to meet this constraint, the timing constraint is not compromised.

```
dc_shell-topo> set_multi_vth_constraint \
    -lvth_groups {lvt svt} -lvth_percentage percentage \
    -type soft -include_blackboxes
```

The `set_multi_vth_constraint` command takes precedence over leakage optimization.

Performing Power Optimization

The `compile_ultra` command performs power optimizations, by default, along with the timing and area optimizations.

An incremental compile uses the existing netlist as a starting point for continued optimization. Usually, this ensures improvement for timing, power, and area (or for other active constraints you define). If you have a design goal that is not met (a violated constraint), a subsequent optimization session attempts to meet the violated constraint by sacrificing lower-priority design goals.

Settings for Power Optimization

The power optimization and prediction settings are used when you run the `compile_ultra` or `compile_ultra -incremental` command to perform accurate power estimation. The tool also uses these settings to get accurate post-synthesis power numbers comparable with the place-and-route numbers. Design Compiler Graphical supports IEEE 1801, also known as Unified Power Format (UPF), in the physical guidance flow. For more details on UPF, see [UPF Multivoltage Design Implementation](#).

To enable clock-gate optimization, use the `-gate_clock` option and the `-spg` option of the `compile_ultra` command. The Power Compiler tool inserts, modifies, or deletes clock-gating cells, except where you have set the `dont_touch` attribute on a clock-gating cell or its parent hierarchical cell.

When you enable the power prediction feature by using the `set_power_prediction` command, the tool performs clock tree estimation during the last phase of the `compile_ultra` command.

The `report_power` command reports the correlated power when the design is mapped to technology-specific cells. When the power prediction feature is disabled, the `report_power` command reports only the total power, static power, and dynamic power used by the design without accounting for the estimated clock-tree power. For more details about using the low-power placement feature in multicorner-multimode designs, see [Optimizing for Dynamic Power Using Low-Power Placement](#).

Power Optimization in the Physical Guidance Flow

The Synopsys physical guidance feature enables Design Compiler Graphical to save the physical guidance information and pass this information to the IC Compiler II tool. This section discusses the settings required for the low-power placement feature. For general details of the physical guidance flow, see the *Design Compiler User Guide*.

Settings for Low-Power Placement

Use the following command and variable settings to enable low-power placement:

- `set_dynamic_optimization true`
- `power_low_power_placement true`

When you enable the low-power placement feature, the tool optimizes the dynamic power by shortening the net lengths of high-switching activity nets. Since the dynamic power saving is based on the switching activity of the nets, annotate the switching activity by using the `read_saif` command. Then, synthesize the design using the `compile_ultra -spg` command. [Example 31](#) shows how to enable and use the low-power placement feature.

Example 31 Enabling and Using the Low-Power Placement Feature

```
set_dynamic_optimization true
set_app_var power_low_power_placement true
read_saif -input s1.saif -instance_name inst_1
compile_ultra -spg
report_power
```

It is recommended, but not required, to read in the RTL SAIF file before optimization. If the RTL SAIF file is not available, the tool uses the defaults, a static probability of 0.5 and a toggle rate of 0.1. If you want to annotate your own values, use the `set_switching_activity` command.

Note:

The `power_low_power_placement` variable is not supported in the Design Compiler NXT tool. When using the Design Compiler NXT tool, you can enable total power optimization by setting the `compile_enable_total_power_optimization` variable to `true`.

10

Multivoltage Design Concepts

In multivoltage designs, the subdesign instances operate at different voltages. In multisupply designs, the voltages of the various subdesigns are the same, but the blocks can be powered on and off independently. In this user guide, unless otherwise noted, the term multivoltage includes multisupply and mixed multisupply-multivoltage designs.

For information about multivoltage designs and library requirements, see the following topics:

- [Multivoltage and Multisupply Designs](#)
 - [Library Requirements for Multivoltage Designs](#)
 - [Power Domains](#)
 - [Voltage Areas](#)
-

Multivoltage and Multisupply Designs

The logic synthesis tools support the following types of low-power designs:

- Multivoltage
- Multisupply
- Mixed multivoltage and multisupply

To reduce power consumption, multivoltage designs typically make use of power domains. The blocks of a power domain can be powered up and down, independent of the power state of other power domains (except where a relative always-on relationship exists between two power domains).

Multivoltage designs have nets that cross power domains to connect cells operating at different voltages. Some power domains can be always-on, that is, they are never powered down, while others might be always-on relative to some specific power domain. Some power domains shut down and power up independently, but might require isolation and other special cells. In general, voltage differences are handled by level shifters, which step the voltage up or down from the input side of the cell to the output side. The isolation cells isolate the power domain. Note that an enable-type level shifter can be used as isolation cells.

Library Requirements for Multivoltage Designs

To synthesize a multivoltage design using the Power Compiler tool, the logic libraries must conform to the Liberty syntax. The libraries should also contain special cells such as clock-gating cells, level-shifters, isolation cells, retention registers, and always-on buffers and inverters. To support synthesis of multivoltage designs, the tool supports multiple libraries characterized at different voltages. The following topics describe the types of cells that support multivoltage or low-power designs:

- [Liberty PG Pin Syntax](#)
- [Level-Shifter Cells](#)
- [Isolation Cells](#)
- [Requirements of Level-Shifter and Isolation Cells](#)
- [Retention Register Cells](#)
- [Power-Switch Cells](#)
- [Always-On Logic Cells](#)

Liberty PG Pin Syntax

In the traditional, non-multivoltage designs, all components of the designs are connected to a single power supply at all times. Therefore the logic libraries used for synthesizing such designs do not contain details of power supply and ground connections of cells because all the cells are connected to the same type of VDD and VSS.

For the synthesis of multivoltage designs, it is necessary to specify the power supplies that can be connected to specific power pins of a cell. The Liberty syntax supports the specification of power rail connection to the power supply pins of the cells. This power and ground (PG) pin information allows the synthesis tool to optimize the design for power and to analyze the design behavior where multiple supply voltages are being used.

Before loading multicorner-multimode libraries, you can set the `mv_align_library_pg_pins` variable to `true` to get a consistent ordering of the power pins across the libraries. The default is `false`.

For specific information about the PG pin syntax and the modeling of power supply pin connections, see the Advanced Low Power Modeling chapter in the *Library Compiler User Guide*.

For an older library that does not contain PG pins, you can convert the library into PG pin library format in Design Compiler. For more details, see [Converting Libraries to PG Pin Library Format](#).

Level-Shifter Cells

In a multivoltage design, a level shifter is required where a signal crosses from one power domain to another. The level shifter operates as a buffer with one supply voltage at the input and a different supply voltage at the output. Thus, a level shifter converts a logic signal from one voltage level to another, with a goal of having smallest possible delay from input to output.

Level-shifter cells are of three types:

- Level shifters that convert from high voltage to low voltage (H2L)
- Level shifters that convert from low voltage to high voltage (L2H)
- Level shifter that can do both, high to low and low to high conversion

PG Pin Configuration Support

In addition to the different types of voltage conversions, the Power Compiler tool supports level-shifter cells with different PG pin configurations as specified by the Library Compiler models:

- Single-rail level shifter
- Dual-rail level shifter which has two PG pins. One pin is designated as the main rail and connected to the primary power supply while the other pin is connected to a secondary rail.
- Level shifter with a feedthrough standard cell main rail (SCMR) PG pin enables shifting of always-on signals between shutdown power domains. The feedthrough SCMR PG pin is connected to the domain's primary supply, which is not part of either the power domains involved in the level shifting.
- Level shifter inside a macro cell, which is connected directly to the macro cell's input pins. This model eliminates the need to insert external level shifters.
- Enable level shifter, which performs both level shifting and isolation functions. There are two types of enable level shifters. In all the following cases, the enable pin's PG pin is connected to the isolation supply.

- A cell that is modeled as a level shifter followed by an isolation cell. For this cell, the input data pin is powered by one supply while the enable and output pins are powered by another supply.
- A cell that is modeled as an isolation cell followed by a level shifter cell. This cell has four terminal pins, and it can be modeled as follows:
 - input, enable, and output pins are each powered by a different supply
 - input and enable pins are powered by one supply, and the output pin is powered by another supply

Support for NOR-Type Enable Level-Shifter Cells

The Power Compiler tool supports the following NOR-type enabled level-shifter cells:

- NOR-type enabled level-shifter cells
 - Single-rail overdriven
 - Dual rail
 - Always-on with placeholder `std_cell_main_rail` Liberty attribute
- NOR-type isolation and level shifter combination cells
 - Dual rail
 - Always-on with placeholder `std_cell_main_rail` Liberty attribute

These cells are inferred by using an explicit or empty isolation supply expression in the `set_isolation` command. For example,

```
set_isolation -isolation_supply {}
set_isolation -isolation_supply {PD.primary}
```

For more information about modeling level shifters, see the *Library Compiler User Guide*.

Isolation Cells

Isolation cells are required when a logic signal crosses from a power domain that can be powered down to a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a constant output signal when the input side is powered down.

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

For more information about creating and using isolation cells and enable level-shifter cells, see the *Library Compiler User Guide*.

Using Standard Cells as Isolation Cells

When your target library does not contain a complete set of isolation cells, you can use the basic 2-input AND, OR, NAND, and NOR gates as isolation cells. This flexibility allows you to use these basic cells for their usual logic as well as for isolation logic. Only the following types of basic gates can be used as isolation cells:

- 2-input AND, OR, NAND, and NOR gates
- 2-input AND, OR, NAND, and NOR gates with one of the inputs inverted

To enable this feature, you must set the `mv_use_std_cell_for_isolation` variable to `true`. You must then set the following attributes using the `set_attribute` command:

- Set the library cell-level attribute `ok_for_isolation` to `true` on the library cell.

This attribute denotes that the library cell can be used as a standard logic cell as well as an isolation cell. The following example shows how to set the `ok_for_isolation` attribute on the library cell A:

```
set_attribute [get_lib_cells lib_name/A] ok_for_isolation true
```

- Set the `isolation_cell_enable_pin` attribute to `true` on the library cell pin. This attribute specifies the pin to be used as the control pin of the isolation cell.

The following example script shows how to set the `isolation_cell_enable_pin` attribute to `true` on the `in` pin of the library cell A:

```
set_attribute [get_lib_pins lib_name/A/in] \
  isolation_cell_enable_pin true
```

Single-Rail and Dual-Rail Isolation Cells

When selecting an isolation cell for mapping, the tool automatically selects single-rail or dual-rail cells based on the isolation cell's rail information and location.

Typically, you use a single-rail isolation cell if the cell is inserted in a domain where the primary rail remains on during shutdown mode. Use a dual-rail isolation cell when the isolation cell needs to be inserted in the shutdown domain and requires a secondary rail connection, so the cell continues to be powered during shutdown mode by a backup supply. You can create exceptions to this rule by using the `use_interface_cell` command to restrict the availability of cells. The tool can insert a conflicting cell if the proper cell is not available. For example, if only single-rail cells are available, the tool inserts them even in a shutdown region. However, the tool does issue a warning message. The tool checks and reports warnings for rail violations, for example, a single-rail isolation cell used in a shutdown domain or a dual-rail isolation cell used in a domain that is powered on more than the shutdown domain.

NOR-Style Isolation Cells

The Power Compiler tool automatically selects and inserts NOR-style (or clamp-to-zero) isolation cells in the shutdown domain. This isolation cell is a single-rail cell that clamps its output to ground (zero) when the domain's primary supply is turned off. The tool can also optimize dual-rail isolation cells to a single NOR-style isolation cells to reduce the power and area used.

To specify a NOR-style isolation cell, specify the isolation strategy with the `set_isolation` command and the `-isolation_supply` option. The `-clamp_value` option must be set to zero. For example,

```
dc_shell> set_isolation ISO1 -domain PD_BLK -isolation_supply {} \
           -clamp_value 0 -applies_to outputs
```

Note:

Since the empty isolation supply set {} indicates that the isolation cell has no power supply when operating in isolation mode, the tool uses a clamp-to-zero isolation cell (or a NOR-type isolation cell).

The tool optimizes the design using NOR-type isolation cells when it can. For example,

```
dc_shell> set_isolation ISO1 -domain PD_BLK \
           -isolation_supply {TOP_AO_SS} -clamp_value 0 \
           -applies_to outputs
```

The tool optimizes the implemented isolation cell to use the domain's primary supply, PD_BLK.primary, by using a NOR-type isolation cell.

For more details on modeling and using NOR-style isolation cells, see [SolvNetPlus article 2370685, "Single-Rail Clamp-to-0 Nor-Type Isolation Cells."](#)

Isolation Cells With Asynchronous Set or Reset Pins

An isolation cell with a clamp value set to `latch` might have an asynchronous set or reset pin. To insert this type of isolation cell with the `set_isolation` command, you must use the `-async_set_reset` option to specify the net name of the asynchronous control signal and its sensitivity (high or low).

For example, the following command creates isolation strategy ISO1 for a latch-type isolation cell. The command specifies that net RS1 is an asynchronous set or reset control signal that is active in the high state:

```
dc_shell> set_isolation ISO1 -domain PD1 \
           -clamp_value latch -async_set_reset {RS1 high}
```

You can optionally use the `-async_clamp_value` option to specify whether the pin is a set input (with an argument of `1`) or a reset input (with an argument of `0`). The following command specifies that the asynchronous control pin on the isolation cell is a reset pin:

```
dc_shell> set_isolation ISO1 -domain PD1 \
    -clamp_value latch -async_set_reset {RS1 high} \
    -async_clamp_value 0
```

You might not want to specify whether the asynchronous pin should act as a set or reset pin at the time of isolation strategy definition. In this case, use the `set_isolation` command without the `-async_clamp_value` option. Then you can later use the `set_port_attributes` command to set the `UPF_async_clamp_value` attribute on specific ports outside the isolation strategy to `1` for set or `0` for reset, as shown in the following example:

```
dc_shell> set_port_attributes {port1 port2} \
    -attribute {UPF_async_clamp_value 0}
```

The Power Compiler tool honors the `set_isolation` command options as follows:

- If either or both of the `-async_clamp_value` or `-async_set_reset` options are specified for an isolation strategy but the `-clamp_value` option is not set to `latch`, the tool issues an error message.
- If a net name is specified with the `-async_set_reset` option but the sensitivity is not set for that signal, the tool issues an error message.
- If a conflict exists between the `-async_clamp_value` option of the `set_isolation` command and the `UPF_async_clamp_value` attribute for a specific pin, the tool honors the attribute setting and issues a warning.
- If the `set_isolation` command uses the `-async_set_reset` option but there is no clamp value set on the specified pin using either the `-async_clamp_value` option or the `UPF_async_clamp_value` attribute, the tool ignores the `-async_set_reset` option with a warning.
- If the `-async_set_reset` option is not used, the tool ignores and drops the `-async_clamp_value` option with a warning when the tool executes an action command (such as the `check_mv_design`, `save_upf`, `compile`, and `insert_mv_cells` commands). In this case, the strategy becomes a normal isolation latch strategy.
- If the `-async_set_reset` option is not defined for the isolation strategy associated with the port, the tool silently ignores the `UPF_async_clamp_value` attribute.

For more information about creating this type of isolation cell, see the *Library Compiler User Guide*.

Requirements of Level-Shifter and Isolation Cells

The following are the requirements of level-shifter and isolation cells:

- Two power supplies.
 - Buffer-type and enable-type level-shifter library cells must have the `is_level_shifter` library attribute set to `true`.
 - Enable-type level shifters must also have the `level_shifter_enable_pin` library attribute set on the enable pin.
 - Isolation library cells must have the `is_isolation_cell` library attribute set to `true`.
 - Isolation cells must have the `isolation_cell_enable_pin` library attribute set on the enable pin.
 - Level shifters and isolation cells are selected by the logic synthesis tool from the target libraries. Therefore, at least one of the libraries must contain these required cells.
 - Level-shifter and isolation cells can only be inserted on unidirectional ports.
-

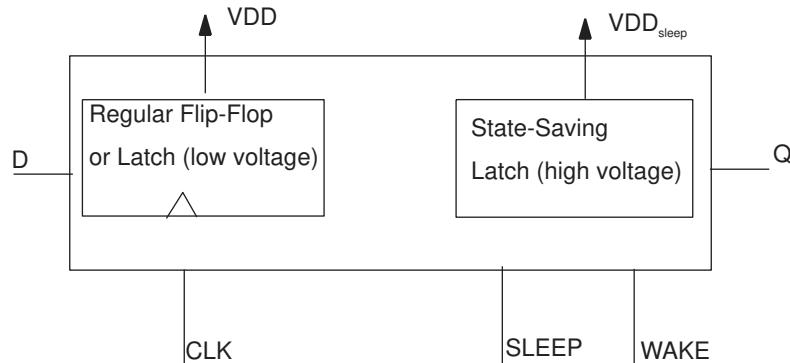
Retention Register Cells

In a design with power switching, one of the ways to save register states before power-down and restore them upon power-up is to use retention registers. These registers can maintain their state during power-down by means of a low-leakage register network and an always-on power supply. Retention cells occupy more area than regular flip-flops. These cells continue to consume power when the power domain is powered down.

Multithreshold-CMOS Retention Registers

Retention cells are sequential cells that can hold their internal state when the primary power supply is shut down and that can restore the state when the power is brought up. So the retention registers are used to save leakage power in power-down applications. During normal operation, there is no loss in performance and during power-down mode, the register state is saved. These features are possible with the addition of a state-saving latch, which holds the data from the active register. [Figure 54](#) shows the basic elements of the retention register.

Figure 54 Retention Register Components



The retention register consists of two separate elements:

- Regular flip-flop or latch

The regular flip-flop or latch consists of low-threshold voltage MOS transistors for high performance

- State-saving latch

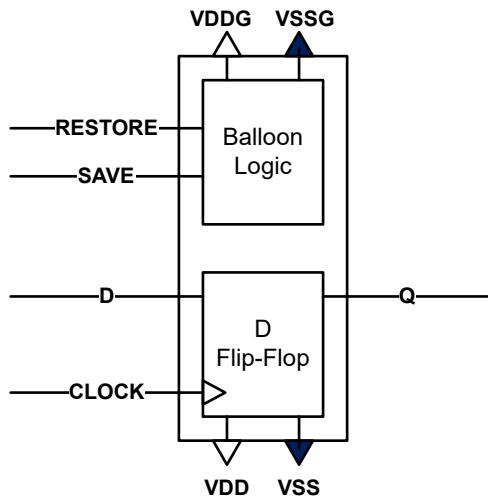
The state-saving latch consists of a balloon circuit modeled with high-threshold voltage MOS transistors. It has a different power supply: VDD_{SLEEP}

The behavior of these elements depends on the circuit mode. During active mode, the regular register operates at speed and the retention latch does not add to the load at the output. During sleep mode, the Q data is transferred to the state-saving latch, and the power supply to the flip-flop is shut off, thus eliminating the high-leakage standby power. When the circuit is activated with the wake-up signal, the data in the retention latch is transferred to the regular register for continuous operation.

Along with the separate power supplies, additional signals such as SLEEP and WAKE are required to enable the data transfer from the regular register to the state-saving latch and back again, based on the mode of operation.

Based on the application, different retention register types are available to address the clocking of the data from the register to the latch and back again. Library Compiler supports modeling of retention registers with two control pins as well as only one control pin. [Figure 55](#) shows a retention register that has two control signals, save and restore, to save and restore the data. In this figure, triggering the Save pin puts the register in the active mode meaning the register works as a regular D flip-flop. Triggering the Restore pin puts it in sleep mode meaning the register works as a state-saving latch.

Figure 55 Two-Pin Retention Register



The Library Compiler tool also supports single-pin retention registers. For single-pin retention registers, the control pin, called a `save_restore` pin, saves and restores the state of the cell depending on the voltage state of the pin. Single-pin retention registers behave like the two-pin control retention register in [Figure 55](#). The only difference is that the control pin is a single pin instead of two pins. For a single-pin retention register to work like a regular latch (D flip-flop, in this case), the `save_restore` pin needs to be put into "save" mode. When the retention register is put into "restore" mode, it works like a retention cell. That is, the D-input of the register is not passed to the balloon logic. Thus, the balloon logic of the retention register has the last known value saved in it. This value is fed to the Q-output when restore mode is enabled.

An example of a retention library cell might be defined as follows:

```
RETENTION_PIN: (save_restore, 1)
```

The disable value is 1, which means the retention cell is working in normal save mode when the `save_restore` pin is driven to 1 (high) and working in restore mode when the `save_restore` pin is driven to 0 (low).

For a UPF file specification, you need to define the retention register control in the UPF file as follows to make this work correctly:

```
set_retention_control PD1_RFF -domain PD1 -save_signal {SRPG1 high} \
    -restore_signal {SRPG1 low}
```

Power-Switch Cells

In a design with power switching, the power-switch cells provide the supply power for cells that can be powered down. The library description of a power-switch cell specifies the

input signal that controls power switching, the pin or pins connected to the power rail, and the pin or pins that provide the virtual or switchable power.

There are two types of power-switch cells, the header type and the footer type. A header type power switch connects the power rail to the power supply pins of the cells in the power-down domain. A footer type power switch connects the ground rail to the ground supply pins of the cells in the power-down domain.

For more information about creating power-switch cells, see the *Library Compiler User Guide*.

Always-On Logic Cells

Multivoltage designs can contain some power domains that can be shut down during the operation of the design. These are also called power-down domains. In some of the power-down domains, logic cells need to remain powered on even when the power domain is shut down. Such cells are called always-on cells. The control signals of the always-on cells should also be powered on when the power domain is shut down. These control signal paths are called always-on paths.

The always-on cells can be of two types:

- Single-power standard cell

Buffers and inverters from the standard cell libraries can be used as always-on cells.

For the Power Compiler tool to use the standard cells as always-on cells, you must

- Define the power domain as a shutdown domain.

For more details on always-on logic, see [Shut-Down Blocks](#).

- Set the `always_on_strategy` attribute to `cell_type` and `single_power`.

- Dual-power special cell

Special cells in the target library, such as buffers and inverters with dual power, can be used for always-on logic. The tool automatically infers the backup power supply for these cells based on the supply load on these cells. For more details, see [Always-On Logic](#).

For more information about always-on logic, see [Shut-Down Blocks](#).

Power Domains

Multivoltage designs contain design partitions which have specific power behavior compared to the rest of the design. A power domain is a basic concept in the Synopsys low-power infrastructure, and it drives many important low-power features across the flow.

By definition, a power domain is a logical grouping of one or more logic hierarchies in a design that share the same power characteristics, including:

- Primary voltage states or voltage range (that is, the same operating voltage)
- Process, voltage, and temperature (PVT) operating condition values (all cells of the power domain except level shifters)
- Power net hookup requirements
- Power down control and acknowledge signals, if any
- Power switching style
- Same set or subset of nonlinear delay model (NLDM) target libraries

Thus, a power domain describes a design partition, bounded within logic hierarchies, that has a specific power behavior with respect to the rest of the design.

Each power domain has a supply network consisting of supply nets and supply ports and might contain power switches. The supply network is used to specify the power and ground net connections for a power domain. A supply net is a conductor that carries a supply voltage or ground. A supply port is a power supply connection point between the inside and outside of the power domain. Supply ports serve as the connection points between supply nets. A supply net can carry a voltage supply from one supply port to another.

When used together, the power domain and supply network objects allow you to specify the power management intentions of the design.

Every power domain must have one primary power supply and one primary ground. In addition to the primary power and ground nets, a power domain can have any number of additional power supply and ground nets.

A power domain has the following characteristics:

- Name
- Level of hierarchy or scope where the power domain is defined or created
- The set of design elements that comprise the power domain
- Associated set of supply nets that are allowed to be used within the power domain
- Primary power supply and ground nets
- Synthesis strategies for isolation, level-shifters, always-on cells, and retention registers

Note:

A power domain is strictly a synthesis construct, not a netlist object.

Shut-Down Blocks

Multivoltage designs typically have some power domains that are shut down and powered up during the operation of the chip while other power domains are always powered up. The always-on paths starting from an always-on block must connect to the specific pins of always-on cells in the power-down block. These cells can be special, dual power cells (isolation cells, enable-type level shifters, retention registers, special RAMs, and so on) or standard cells that when placed are confined to special always-on site rows within the power-down block.

Specific commands are supported by the tool can be used to specify the always-on methodology to be applied to a particular power-down block. If special cells are used, they need to be marked appropriately so that the tool can determine the always-on paths and correctly optimize these paths.

Only buffers and inverters can be used as dual-power, always-on cells. They must have two rails connections: a primary rail that is connected to a shut-down power supply, and a secondary rail that is connected to an always-on power supply.

Marking Pass-Gate Library Pins

In the current implementation, the tool has the ability to stop always-on cells from connecting to cells with pass gate inputs. An always-on buffer should not drive a gate that has pass transistors at the inputs (pass-gate). Pass-gate input cells should be driven by a standard cell in a shut-down power domain. Therefore, if your library contains any of these cells, you must mark them as pass-gates in each session.

For example, to mark the pin A of the mux cell MUX1, run the following command as part of a Design Compiler script:

```
set_attribute [get_lib_pins lib_name/MUX1/A] pass_gate true
```

Voltage Areas

Corresponding to the power domains of logic synthesis, you define *voltage areas* in physical synthesis as placement areas for the cells of the power domains. Except for level shifter cells, all cells in a voltage area operate at the same voltage.

There must be an exact one-to-one relationship between logical power domains and physical voltage areas. The Design Compiler and IC Compiler II tools can align the logic hierarchies of the power domains with their voltage areas with appropriate specifications. *The power domain name and the voltage area name should be identical.*

If you do not make these specifications, you are responsible for ensuring that the logic hierarchies are correctly aligned, as well as being correctly associated with the appropriate operating conditions.

A voltage area is the physical implementation of a power domain. A voltage area is associated with a power domain in a unique, tightly bound, one-to-one relationship. A voltage area is the area in which the cells of specific logic hierarchies are to be placed. A single voltage area must correspond to another single power domain, and vice versa. The power domains of a design are defined first in the logical synthesis phase and then the voltage areas are created in the physical implementation phase, in Design Compiler topographical mode or in the IC Compiler II tool. The information that pertains to logic hierarchies, which belongs to a voltage area boundary is derived from a corresponding power domain. Also, all the cells that belong to a given voltage area have the power behavior described by the power domain characteristics.

11

UPF Multivoltage Design Implementation

The IEEE 1801 specification, also known as Unified Power Format (UPF), provides a standard set of commands that define how to synthesize multivoltage designs in the Power Compiler tool.

For more information about multivoltage design concepts and UPF, see the following topics:

- [Multivoltage Design Flow Using UPF](#)
- [Power Intent Concepts](#)
- [Defining Power Intent With UPF Commands](#)
- [Setting the UPF Command Scope](#)
- [Creating Power Domains](#)
- [Creating Atomic Power Domains](#)
- [Creating Supply Ports](#)
- [Creating Supply Nets](#)
- [Connecting Supply Nets](#)
- [Specifying Supply Sets](#)
- [Refining Supply Sets](#)
- [Querying for Supply Sets](#)
- [Querying for Related Supply Sets](#)
- [Always-On Logic](#)
- [Comparing Voltage Levels and Voltage Status](#)
- [Specifying Level-Shifter Strategies](#)
- [Specifying Isolation Strategies](#)
- [Merging and Cloning Multivoltage Cells](#)

- Setting UPF Attributes on Ports and Hierarchical Cells
- Querying for UPF Design and Port Attributes
- Assigning Supplies to Pad Ports
- Specifying Retention Strategies
- Specifying Repeater Strategies
- Deferring Element Definitions in Power Management Strategies
- Matching Tool and IEEE LRM Defaults
- Creating Power Switches
- Power Models
- Power State Tables
- Support for Well Bias
- Using a Non-Bias Block in a Bias-Enabled Design
- Skipping Bias Checks
- Inserting Power Management Cells
- Reviewing the UPF Specifications
- Examining and Debugging UPF Specifications
- Writing the Power Information
- Writing and Reading Verilog Netlists With Power and Ground Information
- The Golden UPF Flow
- Reporting Commands for the UPF Flow
- UPF-Based Hierarchical Multivoltage Flow Methodology

Multivoltage Design Flow Using UPF

The Unified Power Format (UPF) is a standard set of Tcl-like commands used to specify the low-power design intent for electronic systems. UPF commands provide the ability to specify the power intent early in the design process.

To synthesize the multivoltage design, the recommended method is to use the top-down approach. With the power intent defined in the UPF file, follow these steps to synthesize a multivoltage design:

1. Read the RTL file.
2. Read the power definitions for the multivoltage design using the `load_upf` command.

In the UPF flow, the RTL file cannot have power definitions. The Power Compiler tool issues an error message if it encounters power definitions in the RTL file. All the power definitions must be specified in the UPF file. The UPF file can be used for synthesis, simulation, equivalence checking, and sign off.

By default, the `load_upf` command executes the commands in the associated UPF file in the current level of hierarchy. If the identifiers do not adhere to the naming rules specified in the UPF standard, the following error message is issued.

`Error:Symbol symbol_name violates the UPF naming conventions (UPF-200).`

The Design Compiler commands and variables and the UPF commands and variables defined in the UPF file share the same namespace. While executing the `load_upf` command, the tool checks for namespace conflicts for the commands and variables already defined, and those in the UPF file being read.

For more information, see [Name Spacing Rules for UPF Objects and Attributes](#).

If you have modified the UPF file after reading it, you can use the `remove_upf` command to remove the UPF constraints. However, you cannot use the `remove_upf` command after synthesizing the design or if you read a synthesized design.

After updating or removing a UPF file, use the `load_upf` command to reload the file.

Note:

The Design Vision GUI supports the Visual UPF dialog box, which is accessible from the Power menu. Using the Visual UPF dialog box, you can define the power domains, their supply network, connections with other power domains, and relationships with elements in the design hierarchy.

For more information see [Defining the Power Intent in the GUI](#).

3. Specify the set of target libraries to be used.

The target library must comply with the power and ground pin Liberty library syntax. The target library should also support special cells such as isolation cells and retention registers.

For more details on the target library requirement for multivoltage implementation see [Library Setup for Power Optimization](#). For more information about the PG pin Liberty

library syntax, see the Advanced Low-Power Modeling chapter in the *Library Compiler User Guide*.

4. Use the `set_operating_conditions` command to set the operating condition on the top level of the design hierarchy and to derive the process and temperature conditions for the design.

Use the `set_voltage` command to set the current operating voltage value for the power and the ground supply nets.

5. Specify power optimization requirements.

When you use any of the power optimization constraints in the Design Compiler topographical technology, the tool also enables power prediction using the clock tree estimation. For more information about power prediction, see [Performing Power Correlation](#).

6. Check whether isolation and enable level shifter cells in the design can be mapped successfully by using the `analyze_mv_feasibility` command. If any cell cannot be mapped, the tool generates a report that lists the elements that cannot be mapped and the reasons.

By default, the `analyze_mv_feasibility` command analyzes both isolation and enable level shifter cells. Use the `-isolation` option to analyze only isolation cells or the `-enable_level_shifter` option to analyze only enable level shifter cells. You can generate a detailed HTML report by using the `-format html` option. This report lists all the library cells that the tool would attempt to map but that would be discarded, along with the reason for discarding it.

7. Compile the design by using the `compile_ultra` command.

Note:

When you synthesize the design for the first time using Design Compiler topographical mode, use the `compile_ultra -check_only` command. The `-check_only` option checks the design and the libraries for all the data that is required by the `compile_ultra` command to successfully synthesize the design. For more information, see the *Design Compiler User Guide*.

8. Use the `check_mv_design` command to check for multivoltage violations.

The command checks the design for inconsistencies between the design and the target libraries and for violations related to power management cells and their strategies. Use the `-verbose` option to get the details of the violations. The `-max_messages` option controls the number of violations that are reported.

To identify the multivoltage inconsistencies, use the MV Advisor feature of the Design Vision GUI. For more information, see [Examining and Debugging UPF Specifications](#).

9. Write the synthesized design using the `write -format` command. When you write the design in the ASCII format, use the `change_names` command before you write out the design.

To generate the multivoltage reports, use the various reporting commands such as `report_power_domain`. For more details on multivoltage reporting commands, see [Reporting Commands for the UPF Flow](#).

10. Use the `save_upf` command to save the updated power constraints in another UPF file.

After completing the synthesis process, the UPF file written by the Design Compiler tool is used as input to the downstream tools, such as the IC Compiler II, PrimeTime, PrimePower, and Formality tools. This file is similar to the one read into the Design Compiler tool, but with the following additions:

- A comment on the first line of the UPF file generated by the Design Compiler tool. An example is as follows:

```
#Generated by Design Compiler(H-2013.03) on Wed Feb 20 14:26:58 2011
```

- Explicit power connections to special cells such as level-shifter cells and dual supply cells.
- Any additional UPF commands that were specified at the command prompt in the Design Compiler session.

If you have specified UPF commands at the Design Compiler command prompt during synthesis, update the UPF file along with the RTL design with these commands. Without this update to the UPF file, the Formality tool does not verify the design successfully.

An alternative method to maintain the UPF power intent of the design is called the golden UPF flow. In this method, the original UPF file that you specify is used throughout the synthesis, physical implementation, and verification steps along with supplemental UPF files generated by the tools. For more information, see [The Golden UPF Flow](#).

Power Intent Concepts

The UPF language provides a way to specify the power requirements of a design, but without specifying explicitly how those requirements are implemented. The language specifies how to create a power supply network to each design element, the behavior of supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. It does not contain any placement or routing information. The UPF specification is separate from the RTL description of the design.

In the UPF language, a *power domain* is a group of elements in the design that share a common set of power supply needs. By default, all logic elements in a power domain use the same primary supply and primary ground. Other power supplies can be defined for a power domain as well. A power domain is typically implemented as a contiguous *voltage area* in the physical chip layout, although this is not a requirement of the language.

Each power domain has a *scope* and an *extent*. The *scope* is the level of logic hierarchy designated as the root of the domain. The *extent* is the set of logic elements that belong to the power domain and share the same power supply needs. The scope is the hierarchical level at which the domain is defined and is an ancestor of the elements belonging to the power domain, whereas the extent is the actual set of elements belonging to the power domain.

Each scope in the design has *supply nets* and *supply ports* at the defined hierarchical level of the scope. A *supply net* is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net that spans more than one power domain is said to be “reused” in multiple domains. A *supply port* is a power supply connection point between two adjacent levels of the design hierarchy, between the parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

A *supply set* is an abstract collection of supply nets, consisting of two supply functions, power and ground. A supply set is *domain-independent*, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can physically implemented (placed and routed), its supply sets must be *refined*, or associated with actual supply nets.

A *supply set handle* is an abstract supply set created for a power domain. By default, a power domain has supply set handles for the domain’s primary supply set, a default isolation supply set, and a default retention supply set. These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can be physically implemented, its supply set handles must be *refined*, or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

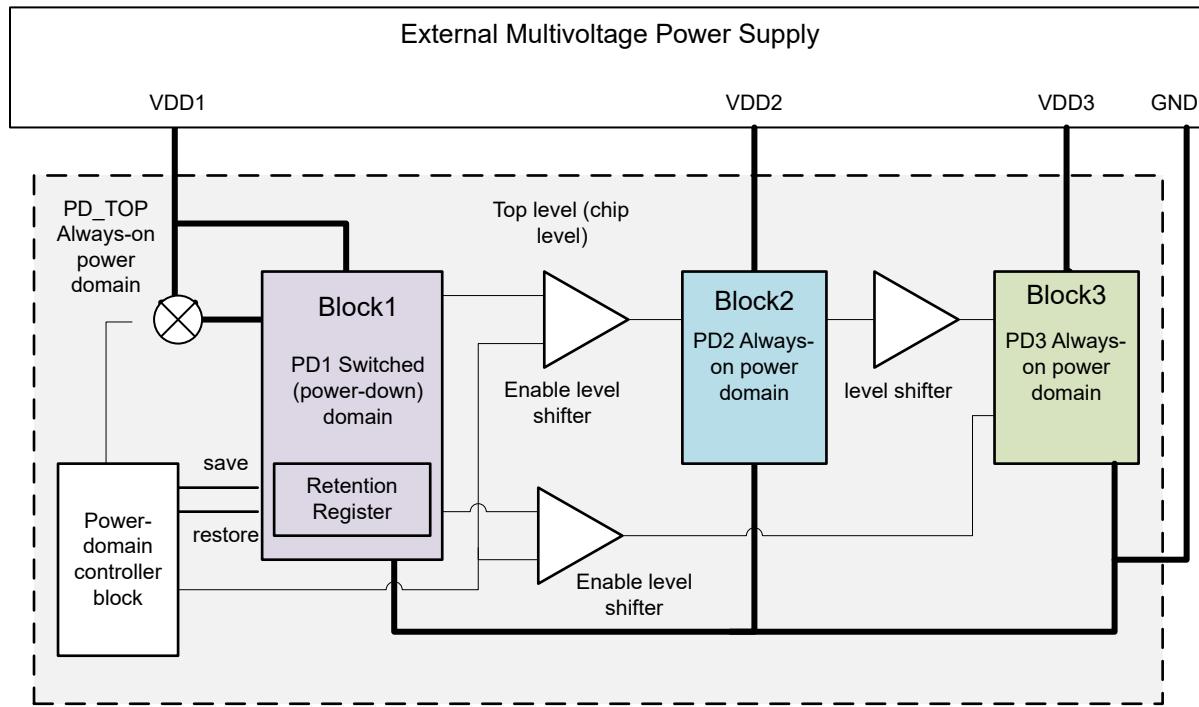
A *power switch* (or simply *switch*) is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals. A *power state table* lists the allowed combinations of voltage values and states of the power switches for all power domains in the design.

A *level shifter* must be present where a logic signal leaves one power domain and enters another at a substantially different supply voltage. The level shifter converts a signal from the voltage swing of the first domain to that of the second domain.

An *isolation* cell must be present where a logic signal leaves a switchable power domain and enters a different power domain. The isolation cell generates a known logic value during shutdown of the domain. If the voltage levels of the two domains are substantially different, the interface cell must be able to perform both level shifting (when the domain is powered up) and isolation (when the domain is powered down). A cell that can perform both functions is called an *enable level shifter*.

In a power domain that has power switching, any registers that must retain data during shutdown must be implemented as *retention registers*. A retention register has a separate, always-on supply net, sometimes called the backup supply, which keeps the data stable in the retention register while the primary supply of the domain is shut down.

Figure 56 Power Intent Specification Example



The power network example shown in Figure 56 demonstrates some of the power intent concepts. This chip is designed to operate with three power supplies that are always on (although the UPF syntax also supports externally switchable power supplies), at three different voltage levels. The top-level chip occupies the top-level power domain, PD_TOP. The domain PD_TOP is defined to have four supply ports: VDD1, VDD2, VDD3, and GND. The black squares along the border of the power domain represent the supply ports of that

domain. Note that this diagram shows the connections between power domains and is not meant to represent the physical layout of the chip.

In addition to the top-level power domain, PD_TOP, there are three more power domains defined, called PD1, PD2, and PD3, created at the levels of three hierarchical blocks, Block1, Block2, and Block3, respectively. Each block has supply ports (shown as black squares in the diagram) to allow supply nets to cross from the top level down into the block level.

In this example, PD_TOP, PD2, and PD3 are always-on power domains that operate at different supply voltages, VDD1, VDD2, and VDD3, respectively. PD1 is a power domain that has two supplies: a switchable supply called VDD1g and an always-on supply from VDD1. The always-on power supply maintains the domain's retention registers while VDD1g is powered down.

A power switch shuts off and turns on the power net VDD1g, either by connecting or disconnecting VDD1 and VDD1g. A power-down controller logic block at the top level generates the control signal for the switch. It also generates the save and restore signals for the retention registers in domain PD1 and the control signals for the isolation cells between domain PD1 and the always-on domains PD2 and PD3. These isolation cells generate known signals during times that VDD1g is powered down.

Because domains PD1, PD2, and PD3 operate at different supply voltages, a level shifter must be present where a signal leaves one of these domains and enters another. In the case of the signals leaving PD1 and entering PD2 or PD3, the interface cells must be able to perform both level shifting and isolation functions, because PD1 can be powered down.

UPF Script Example

[Example 32](#) shows the UPF script that defines the various concepts supported by UPF.

Example 32 UPF Script to Define the Power Intent

```
## CREATE POWER DOMAINS
create_power_domain TOP
create_power_domain PD_ALU -elements {I_ALU} -scope I_ALU
create_power_domain PD_STACK_TOP -elements {I_STACK_TOP} \
-scope I_STACK_TOP
create_power_domain PD_REG_FILE -elements {I_REG_FILE} -scope I_REG_FILE

## SUPPLY NETWORK - PD_ALU
create_supply_net VDD -domain I_ALU/PD_ALU
create_supply_net VSS -domain I_ALU/PD_ALU

create_supply_port VDD -domain I_ALU/PD_ALU
create_supply_port VSS -domain I_ALU/PD_ALU

connect_supply_net I_ALU/VDD -ports {I_ALU/VDD}
connect_supply_net I_ALU/VSS -ports {I_ALU/VSS}
```

```

set_domain_supply_net I_ALU/PD_ALU -primary_power_net I_ALU/VDD
-primary_ground_net I_ALU/VSS

## SUPPLY NETWORK - PD_STACK_TOP
create_supply_net VDDT -domain I_STACK_TOP/PD_STACK_TOP
create_supply_net VSS -domain I_STACK_TOP/PD_STACK_TOP

create_supply_port VDDT -domain I_STACK_TOP/PD_STACK_TOP
create_supply_port VSS -domain I_STACK_TOP/PD_STACK_TOP
connect_supply_net I_STACK_TOP/VDDT -ports {I_STACK_TOP/VDDT}
connect_supply_net I_STACK_TOP/VSS -ports {I_STACK_TOP/VSS}

set_domain_supply_net I_STACK_TOP/PD_STACK_TOP
-primary_power_net I_STACK_TOP/VDDT -primary_ground_net I_STACK_TOP/VSS

## SUPPLY NETWORK - PD_REG_FILE
create_supply_net VDDT -domain I_REG_FILE/PD_REG_FILE
create_supply_net VSS -domain I_REG_FILE/PD_REG_FILE

create_supply_port VDDT -domain I_REG_FILE/PD_REG_FILE
create_supply_port VSS -domain I_REG_FILE/PD_REG_FILE

connect_supply_net I_REG_FILE/VDDT -ports {I_REG_FILE/VDDT}
connect_supply_net I_REG_FILE/VSS -ports {I_REG_FILE/VSS}

set_domain_supply_net I_REG_FILE/PD_REG_FILE
-primary_power_net I_REG_FILE/VDDT -primary_ground_net I_REG_FILE/VSS

## SUPPLY NETWORK - TOP
create_supply_port VDD
create_supply_port VSS
create_supply_port VDDT

create_supply_net VDD -domain TOP
create_supply_net VSS -domain TOP
create_supply_net VDDT -domain TOP

set_domain_supply_net TOP -primary_power_net VDD -primary_ground_net VSS

connect_supply_net VDDT -ports {VDDT I_STACK_TOP/VDDT I_REG_FILE/VDDT}
connect_supply_net VSS
-ports {VSS I_ALU/VSS I_STACK_TOP/VSS I_REG_FILE/VSS}
connect_supply_net VDD -ports {VDD I_ALU/VDD}

## LEVEL-SHIFTER STRATEGY
set_level_shifter ls_alu -domain I_ALU/PD_ALU -applies_to inputs \
-rule both -location self
set_level_shifter ls_stack_top -domain I_STACK_TOP/PD_STACK_TOP \
-applies_to inputs -rule both -location self
set_level_shifter ls_reg_file -domain I_REG_FILE/PD_REG_FILE \
-applies_to inputs -rule both -location self

```

```

set_level_shifter ls1_alu -domain I_ALU/PD_ALU -applies_to outputs \
    -rule both -location self
set_level_shifter ls1_stack_top -domain I_STACK_TOP/PD_STACK_TOP \
    -applies_to outputs -rule both -location parent
set_level_shifter ls1_reg_file -domain I_REG_FILE/PD_REG_FILE \
    -applies_to outputs -rule both -location parent

## ISOLATION STRATEGY
set_isolation iso_stack_top -domain I_STACK_TOP/PD_STACK_TOP
    -isolation_power_net VDD -isolation_ground_net VSS -clamp_value 1 \
        -applies_to outputs -diff_supply_only TRUE
set_isolation iso_reg_file -domain I_REG_FILE/PD_REG_FILE \
    -isolation_power_net VDD -isolation_ground_net VSS -clamp_value 1 \
        -applies_to outputs -diff_supply_only TRUE

# POWER STATE TABLE
## CREATE PORT STATES
add_port_state VDD      -state {TOP 1.08}
add_port_state VDDT     -state {BLOCK 0.864} -state {BLOCK_off off}

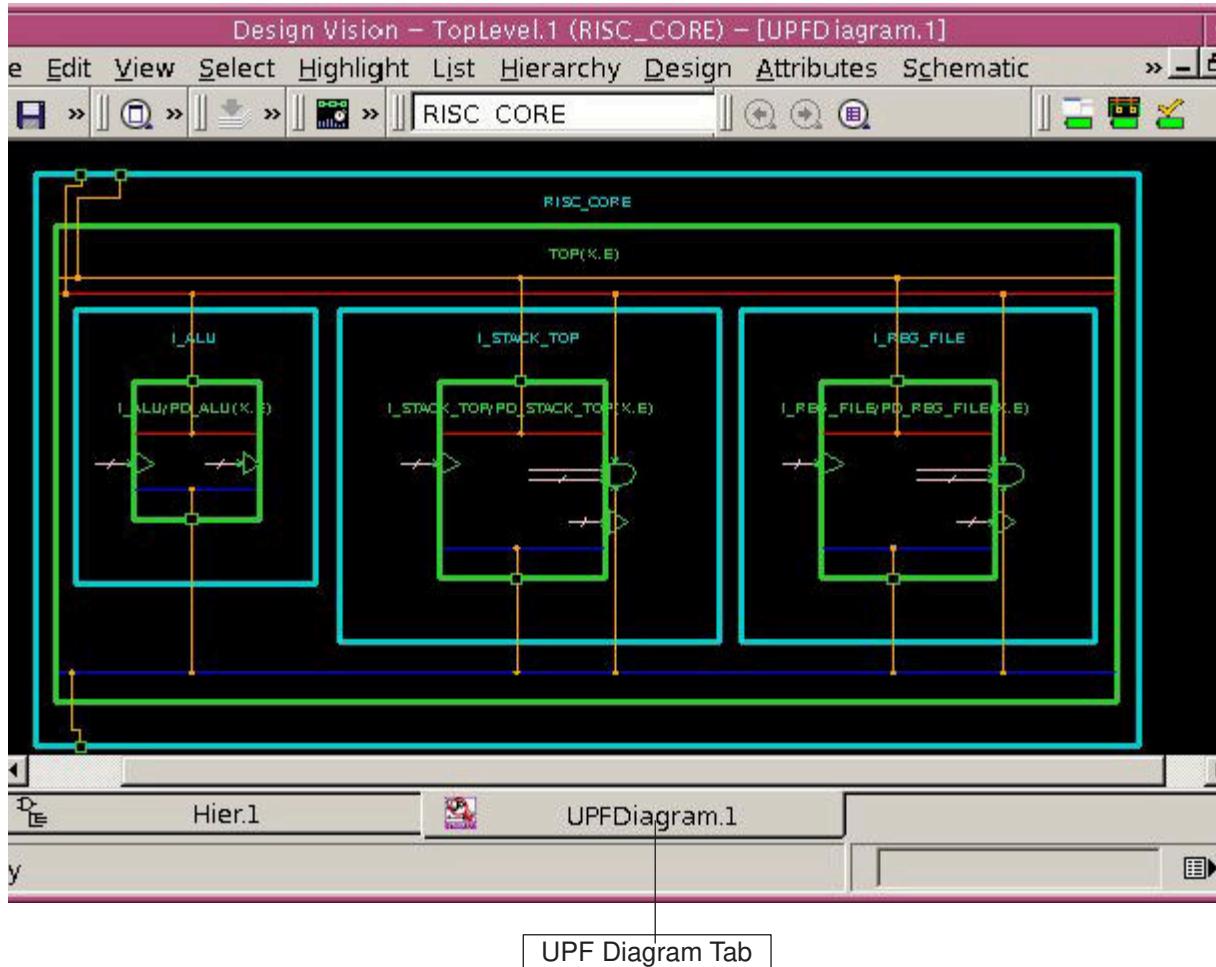
## OPERATING VOLTAGES
create_pst risc_core_pst -supplies          {VDD  VDDT}
add_pst_state s0 -pst risc_core_pst -state {TOP BLOCK}
add_pst_state s1 -pst risc_core_pst -state {TOP BLOCK_off}

set_port_attributes -elements {I_ALU} -applies_to outputs \
    -attribute repeater_power_net I_ALU/VDD \
    -attribute repeater_ground_net I_ALU/VSS
set_port_attributes -elements {I_STACK_TOP} -applies_to inputs \
    -attribute repeater_power_net VDD -attribute repeater_ground_net VSS
set_port_attributes -elements {I_REG_FILE} -applies_to inputs \
    -attribute repeater_power_net VDD -attribute repeater_ground_net VSS

```

[Figure 57](#) shows the UPF Diagram in the Design Vision GUI, for the UPF specification in [Example 32](#).

Figure 57 UPF Diagram in the GUI for the Specified UPF



Defining Power Intent With UPF Commands

The Power Compiler tool supports UPF commands to define, review, and examine the power intent specification. Alternatively, you can use the Design Vision GUI to define and examine the power intent specification.

This section discusses how to use the UPF commands and the GUI to specify the power intent.

Name Spacing Rules for UPF Objects and Attributes

The Power Compiler tool verifies the object names created by the UPF commands to ensure that the names do not conflict with the names of existing objects in the same logic

hierarchy. The tool checks the names of ports, power domains, power state tables, power switches, supply sets and nets, or signal nets. [Table 17](#) shows the name spacing rules applied by the tool for UPF commands:

Table 17 Name Spacing Rules Applied for UPF Commands

UPF command names	Name spacing rule
create_power_domain, create_power_switch, create_pst, create_supply_set	Within a logic hierarchy, a power domain cannot have the same name as an existing cell, instance, logic port, supply port, logic net, supply net, power switch, power domain, supply set, or power state table.
create_logic_net, create_supply_net	Within a logic hierarchy, a net cannot have the same name as an existing cell, instance, logic net, supply net, power switch, power domain, supply set, or power state table.
create_logic_port, create_supply_port	Within a logic hierarchy, a port cannot have the same name as an existing cell, instance, logic port, supply port, power switch, power domain, supply set, or power state table.
set_isolation, set_level_shifter, set_retention	The isolation, level-shifter, and retention strategies in a power domain must have unique names.
add_port_state	One or more connected ports cannot have the same port-state names. However, two ports of a mutually connected network can have the same port state (the name and value are same).
add_power_state	A supply set cannot have power states with the same name. However, two ports of a mutually connected network can have the same power state (both name and value are same).
add_pst_state	The power state table cannot have states with the same name as the already existing states.

Defining the Power Intent in the GUI

The Power menu in the GUI allows you to specify, modify, and review your power architecture. It also lets you view the UPF diagram to examine the UPF specification defined in your design.

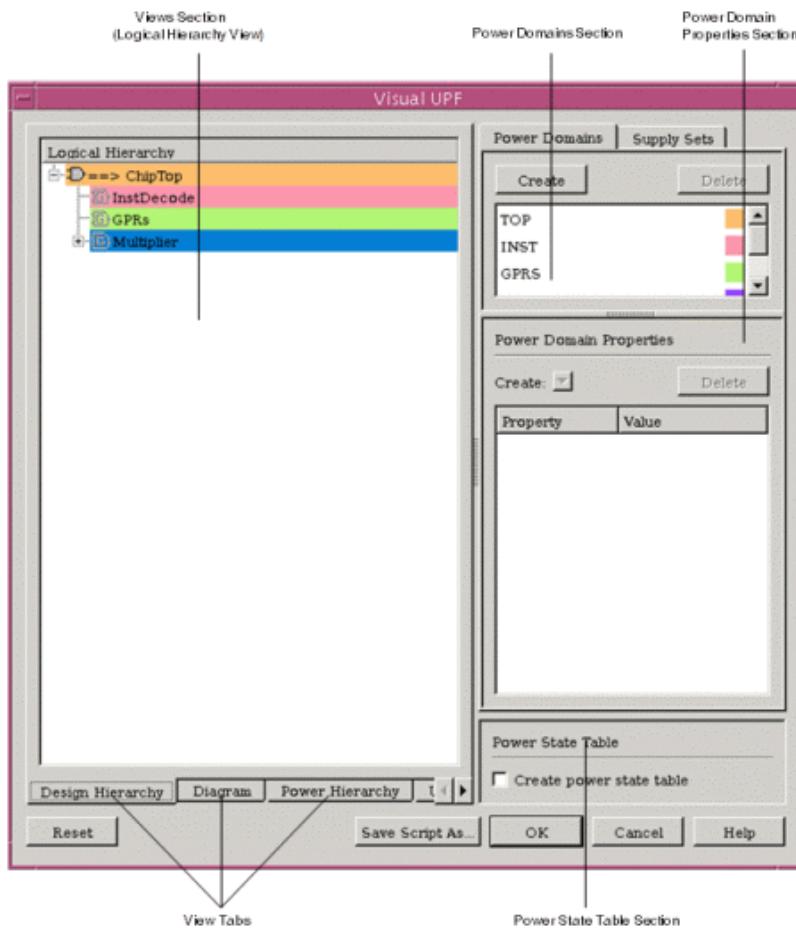
The Visual UPF dialog box in the GUI allows you to define, edit, and review your power intent. You can also generate a UPF script for your power intent.

To open the Visual UPF dialog box,

- Choose Power > Visual UPF

When you open the Visual UPF dialog box, it appears as shown in [Figure 58](#).

Figure 58 Logic Hierarchy View of the Visual UPF



If you have not yet defined the power intent for your design, use the Power Domains and Power Domain Properties sections to create the power domains and other components, such as power switches and level shifters. For the first power domain that you create, the tool assigns the name TOP by default.

If you have already defined the power intent for your design, the Visual UPF displays the details of your power specification. Using the Power Domains and Power Domain Properties sections, you can edit the power definitions. Allowable edits include adding new components, redefining the association of the hierarchical cells with the power domains, and deleting a power domain.

For more details about how to review the UPF intent in the GUI, see [Reviewing the Power Intent Using the Design Vision GUI](#).

UPF Diagram View

The UPF diagram view displays the UPF power intent as it is defined in the design database. When you change the database, for example by entering a UPF command, the tool reflects the updates in the UPF diagram immediately. You can view the UPF diagram at any point in the design flow.

To open the UPF diagram view:

- Choose Power > UPF Diagram > New UPF Diagram View.

When the UPF diagram view appears, Design Vision displays a tab at the bottom of the workspace area, as shown in [Figure 58](#). You can use this tab to return to the UPF diagram view after working with other views.

The UPF diagram view represents each power object with a unique symbol. For more information about these symbols, see the “UPF Diagram Symbols and Standards” topic in Design Vision Help. The tool uses default colors to differentiate the types of power objects. You can customize the diagram by using the View Settings panel to change object colors or apply a color theme.

For more information, see the “Changing UPF Diagram Display Properties” topic in Design Vision Help.

Setting the UPF Command Scope

The scope of a UPF command is the level of design hierarchy to which the UPF command applies. The following terms describe aspects of scope definition:

- The *root scope* is the top-level scope in the design hierarchy.
- The *design top module* is the module for which the UPF file that expresses the power intent has been written.
- The *current scope* is an instance that is either the design top instance or a descendant of the top instance (represented by a relative path name from the design top instance).
- A *design-relative hierarchical name* is interpreted relative to the design top instance by removing the leading slash character (/) and interpreting the remainder as a rooted name in the scope of the current design top instance.

Set the scope in one of the following ways:

- Use the `load_upf` command with the `-scope` option to specify the new design top instance. The tool changes the current scope, the design top instance, and the design top module, then executes the UPF commands. After completion, the tool restores the design top instance and design top model to their previous values.

If you use the `load_upf` command without the `-scope` option, the tool executes the UPF commands in the current scope and does not modify the current scope, the design top instance, or the design top module.

- Use the `apply_power_model` command to apply a power model to specified instances. The tool changes the current scope, the design top instance (to the macro instance to which the model is being applied), and the design top module, then executes the UPF commands. After completion, the tool restores the design top instance and design top model to their previous values.
- Use the `set_scope` command to change the current scope locally. If you use the command without an argument, the scope is the top level of the current design. You can optionally specify an instance name as an argument to change the scope. However, you can only change the scope within the current design subtree. In other words, you cannot set the scope to a scope above the design top instance or below a leaf-level instance.

The `set_scope` command returns the name of the previous scope as a design-relative hierarchical name.

The tool interprets arguments of the `set_scope` command as follows:

- The `set_scope /` command sets the current scope to the current design top instance.
- The `set_scope .` command does not change the current scope.
- The `set_scope ..` command changes the current scope to the parent scope. However, if the current scope is the current design top instance, the current scope is unchanged and the tool issues a warning message.

Note:

The `-scope` option of the `create_power_domain` command specifies where to create a new power domain but does not change the current scope for subsequent UPF commands.

[Table 18](#) illustrates the effect of the `set_scope` and `load_upf` commands.

Table 18 Effect of UPF Commands on the Scope

Design top instance before command	Current scope before command	Command	Design top instance after command	Current scope after command
/top	/top/mid	<code>set_scope bot</code>	/top	/top/mid/bot
/top	/top/mid/bot	<code>set_scope .</code>	/top	/top/mid/bot

Table 18 Effect of UPF Commands on the Scope (Continued)

Design top instance before command	Current scope before command	Command	Design top instance after command	Current scope after command
/top	/top/mid/bot	set_scope ..	/top	/top/mid
//top	/top/mid	set_scope /	/top	/top
/top	/top	load_upf -scope mid	/top/mid	/top/mid
/top/mid	/top/mid	set_scope .	/top/mid	/top/mid
/top/mid	/top/mid	set_scope bot	/top/mid	/top/mid/bot
/top/mid	/top/mid/bot	set_scope /	/top/mid	/top/mid
/top/mid	/top/mid	set_scope ..	n/a (error)	n/a (error)
/top/mid	/top/mid	# load_upf finished	/top	/top
/top	/top	load_upf abc.upf	/top	/top
/top	/top	set_scope mid/bot	/top	/top/mid/bot
/top	/top/mid/bot	set_scope /	/top	/top
/top	/top	load_upf macro.upf -scope macro_inst	/top/macro_inst	/top/macro_inst
/top/macro_inst	/top/macro_inst	set_scope /bot	/top/macro_inst	/top/macro_isnt/bot
/top/macro_inst	/top/macro_inst/bot	set_scope /	/top/macro_inst	/top/macro_inst
/top/macro_inst	/top/macro_inst	set_scope ..	n/a (warning)	n/a (warning)
/top/macro_inst	/top_macro_inst	# load_upf finished	/top	/top

Creating Power Domains

To create a power domain, use the `create_power_domain` command.

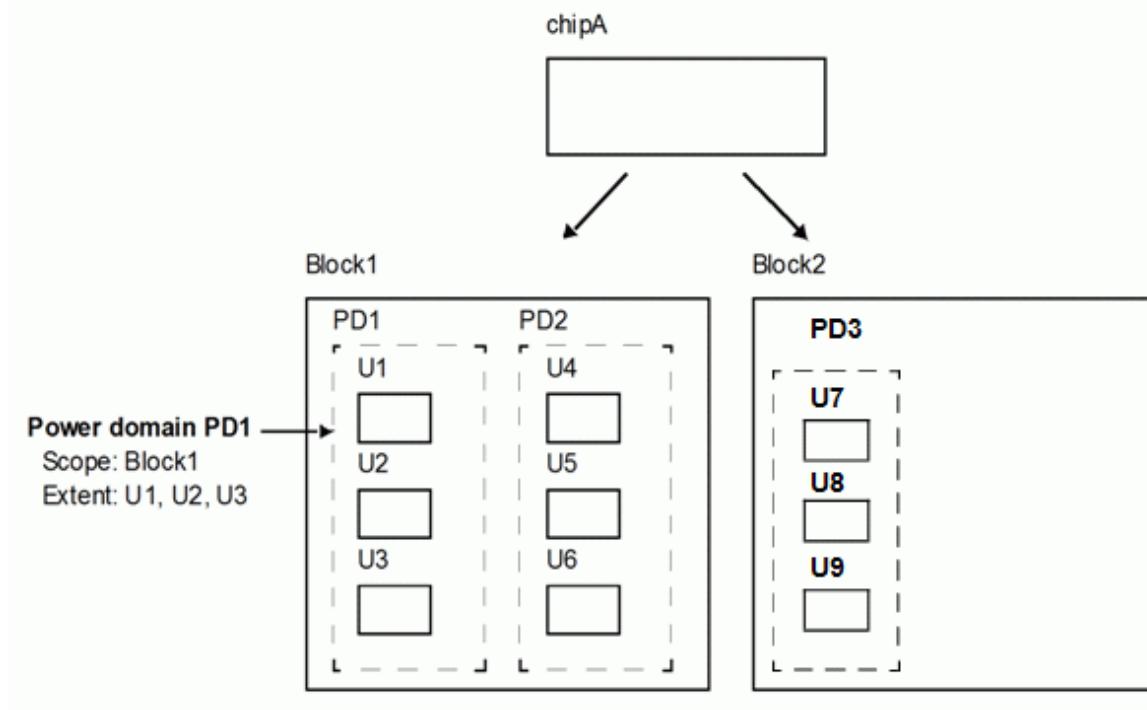
The `-elements` option specifies the hierarchical, macro, leaf-level macro, pad, buffer, and inverter cells that are added to the extent of the power domain. If the required scope is

at a lower level than the current scope, use the `-scope` option to specify the name of the instance where the power domain is to be defined.

The UPF standard requires a simple name for the `domain_name` argument. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

Figure 59 illustrates the usage of the `create_power_domain` command.

Figure 59 Defining a Power Domain and Scope



To create the PD1 and PD2 power domains, use the following commands:

```
create_power_domain -elements {U1 U2 U3} -scope Block1 PD1
create_power_domain -elements {U4 U5 U6} -scope Block1 PD2
```

Alternatively, you can use the `set_scope` command to first set to the required scope and then to create the power domain, as in the following example:

```
set_scope Block1
create_power_domain -elements {U1 U2 U3} PD1
create_power_domain -elements {U4 U5 U6} PD2
```

You can also use the `-elements {.}` option to include the current scope to share the supply of the power domain.

For example, if the current scope is set to the instance `Block2`, the following commands create a power domain `PD3`:

```
set_scope Block2
create_power_domain PD3 -elements { . }
```

In this case, the element `U9` shares the supply of power domain `PD3`, though `U9` is not explicitly specified as part of the power domain `PD3`.

To add new elements to any hierarchy, except those that are already specified as an element of another power domain, use the `-update` option. You can only add new elements.

You can define a power domain with an empty elements list and defer the definition of the element list. For example,

```
dc_shell> create_power_domain D1 -elements {}
```

Later, you can add to the elements list using the `-update` option as follows:

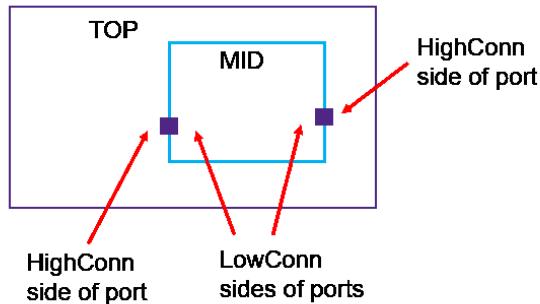
```
dc_shell> create_power_domain D1 -elements e1 -update
```

Power Domain Boundaries

By default, the Power Compiler tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801 (UPF) standard, the tool can consider a power domain boundary to include the boundary of another domain contained in it. You can specify the elements on the lower-domain boundary for level-shifter and isolation strategy definition, which gives you additional flexibility in selecting the location of the power management cells.

In UPF terminology, a port has two sides: the HighConn side and the LowConn side, as shown in [Figure 60](#). The HighConn side is visible to the parent of the instance whose interface contains the port. The LowConn side is visible inside the instance. In this example, `TOP` is the parent and `MID` is the instance.

Figure 60 Sides of a Port



In general, the upper boundary of a power domain is its interface with power domains that are higher in the design hierarchy. You can control how the lower boundary of a power domain is defined.

For more information about lower domain boundaries, see [Lower-Domain Boundary Support](#).

Excluding Elements From Power Domains

To exclude elements from a power domain, use the `-exclude_elements` option with the `create_power_domain` command.

An excluded cell must be explicitly specified as a root element of another power domain (in other words, an object in the argument list of an `-elements` option) before the UPF is committed. For example, the following commands result in an error because cell B does not belong to any power domain:

```
dc_shell> create_power_domain PDtop -elements {C}
dc_shell> create_power_domain PD1 -elements {A B} -exclude_elements {B}
```

By contrast, the following commands are successful because cell B is part of the PDtop power domain even though it is excluded from the PD1 power domain:

```
dc_shell> create_power_domain PDtop -elements {B C}
dc_shell> create_power_domain PD1 -elements {A B} -exclude_elements {B}
```

Wildcards are supported. In the following example, the command creates a power domain that contains cells A and B and all elements in the first level of B's hierarchy. The `-exclude_elements` option excludes only element B/b1. The result is a power domain whose element list is {A B/b2 B/b3 ...} and so on, containing all elements in the first level of hierarchy under B except for b1.

```
dc_shell> create_power_domain PD2 -elements {A B/*} -exclude_elements {B/b1}
```

In the following example, the tool does not exclude element B/b1 because it is not included in the scope of the argument list of the `-elements` option. The power domain contains only the top levels of cells A and B.

```
dc_shell> create_power_domain PD3 -elements {A B} -exclude_elements
           {B/b1}
```

The following conditions apply to the `-exclude_elements` option:

- The argument list can be empty.
- The tool ignores duplicate elements in the argument list.
- The argument list cannot contain elements that are not part of the current design.
- A child element does not inherit power domain membership from its parent. For example, consider the following commands:

```
dc_shell> create_power_domain PD_TOP -elements {.}
dc_shell> create_power_domain PD_A -elements {A A/a*} \
           -exclude_elements {A/a1}
```

The tool reports an error because element A/a1 does not belong to any power domain.

The `-exclude_elements` option excludes A/a1 from the PD_A power domain.

In addition, element A/a1 does not belong to the PD_TOP power domain. The `{.}` argument of the `-elements` option of the `create_power_domain` command means that the PD_TOP power domain includes only the logic hierarchy level where the domain is created (the scope).

- The tool ignores duplicate `-exclude_elements` options.

When you use the `-exclude_elements` option with the `-update` option, the power domain is updated to exclude instances in the `-exclude_elements` argument list. The tool computes the effective element list after reading all UPF commands. The following example first creates an empty power domain, then makes changes to it, resulting in an effective element list of {A}:

```
dc_shell> create_power_domain PD -elements {}
dc_shell> create_power_domain PD -exclude_elements {B} -update
dc_shell> create_power_domain PD -exclude_elements {C} -update
dc_shell> create_power_domain PD -elements {A B C} -update
```

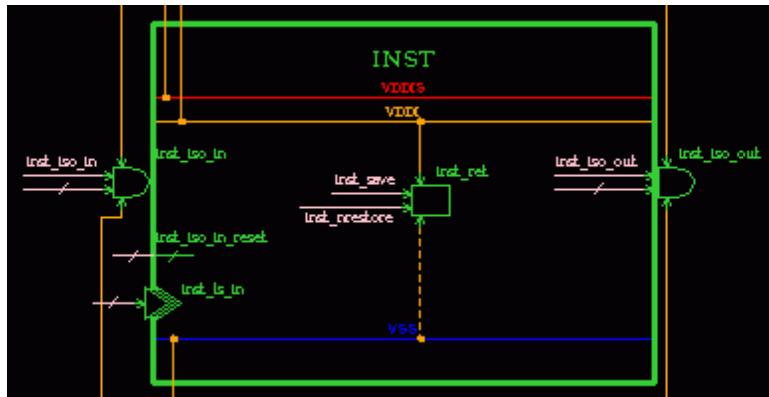
Representation of Power Domain in the UPF Diagram View

The UPF diagram view displays all power domains that are defined in the current design and its subdesigns. The power domains are organized hierarchically, such that each power domain is located inside its parent power domain.

A power domain is represented by a green colored rectangular bounding box. The name of the power domain is displayed inside the bounding box. Figure 61 shows the INST power domain and all the UPF objects contained in the power domain.

The size of the power domain symbol varies according to the number and size of the objects that reside within the power domain. The symbol is big enough to display all the UPF objects that are contained in it.

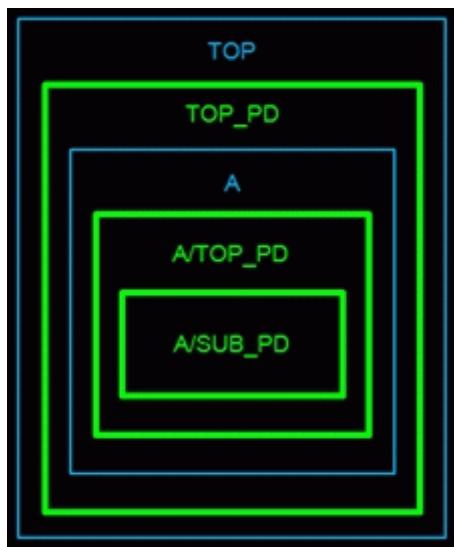
Figure 61 An Example of a Power Domain Representation in the UPF Diagram



Scope

In the UPF diagram view, scope is represented by a blue colored rectangular bounding box. The scope appears within the hierarchy of the power domains. The bounding box of the scope surrounds the top-most child domain in the scope. Figure 62 shows an example of how power domains and scopes appear within the UPF diagram.

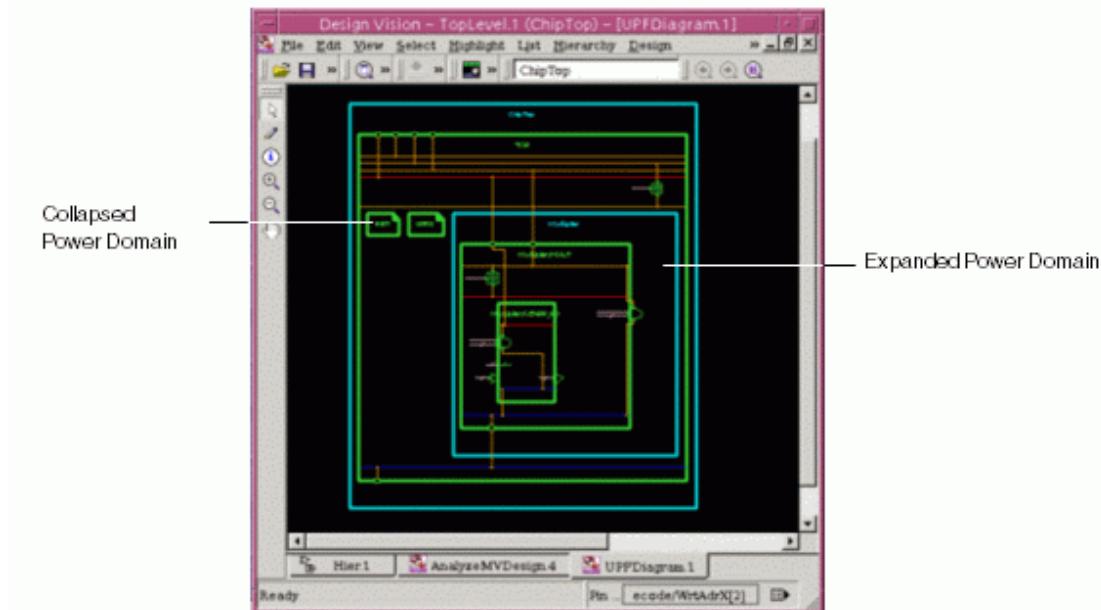
Figure 62 Representation of Power Domains and Scopes in the UPF Diagram



Expanding and Collapsing Power Domains in the GUI

In the UPF diagram view, you can collapse or expand a selected power domain or scope. This is useful when you have large designs with several power domains. When you open the UPF diagram view, by default the power domains are expanded, as shown in [Figure 57](#). When you collapse a power domain, all its internal details disappear from the view, and only its name is displayed, as shown in [Figure 63](#). When you expand a power domain, all its internal details are displayed in the view.

Figure 63 UPF Diagram With Collapsed Power Domains



You can use either of the following methods to expand or collapse a power domain.

After selecting one or more power domains that you want to expand,

- Choose Power > UPF Diagram > Expand Selected Domains.
- Right-click the diagram and choose Expand Selected Domains.

After selecting one or more power domains that you want to collapse,

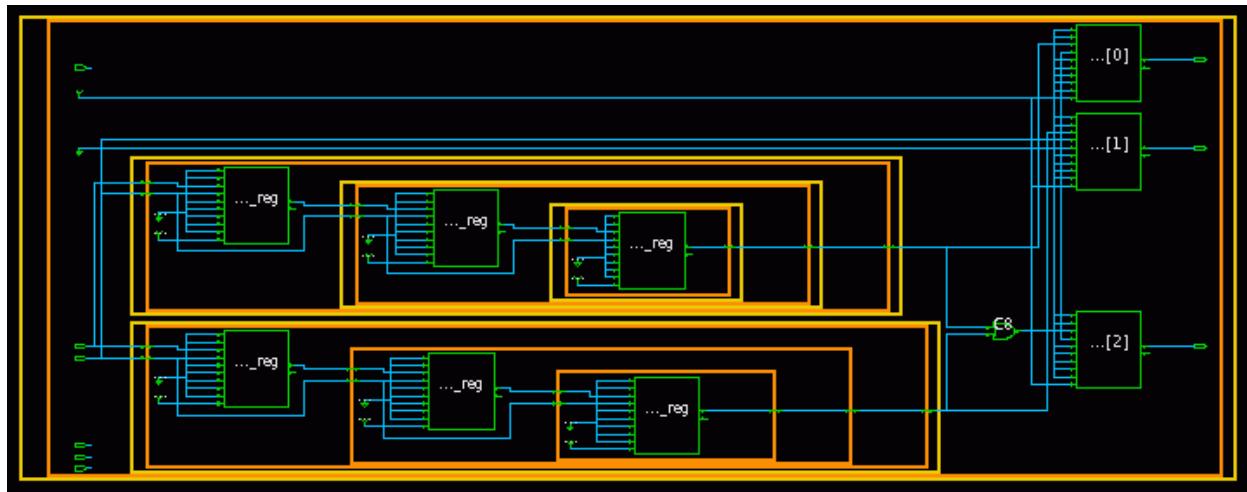
- Choose Power > UPF Diagram > Collapse Selected Domains.
- Right-click the diagram and choose Collapse Selected Domains.

Viewing Hierarchical Cell and Power Domain Boundaries

By default, the schematic view displays timing paths and design logic in a flat, single-sheet schematic that can span multiple hierarchy levels. Hierarchy crossings are represented by diamond shapes and indicate where the nets traverse a level of hierarchy.

You can improve your view of the hierarchical structures in the design by arranging the schematic to display objects hierarchically. Rectangular boundaries appear around objects that share the same hierarchical parent block. Hierarchical cell boundaries appear orange and power domain boundaries appear yellow as shown in [Figure 64](#).

Figure 64 Hierarchical and Power Domain Boundaries



To display or hide hierarchical boundaries in the active schematic view,

- ▶ Choose Schematic > Show Logic/Power Hierarchy.

A check mark beside the command on the Schematic menu indicates that the boundaries are visible.

You can color the objects in a schematic based on the hierarchical power relationships of the design.

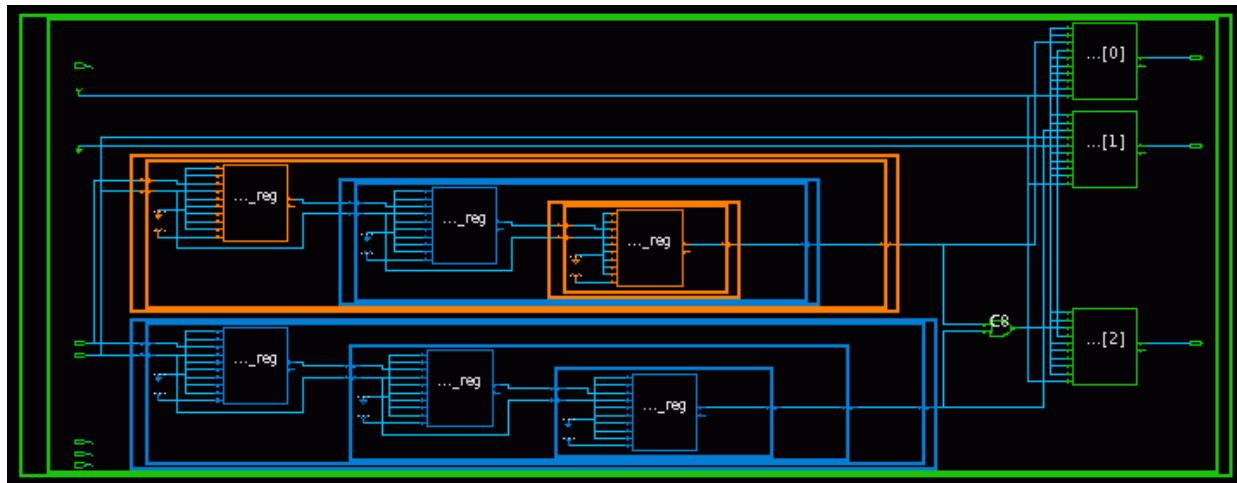
To display or hide boundary coloring based on their power domains,

- ▶ Choose Schematic > Color by Power Hierarchy.

A check mark beside the command on the Schematic menu indicates that the boundary coloring is visible. The tool displays the objects for each power domain with a unique color.

In [Figure 65](#), PD_TOP power domain and its elements appear green. PDA power domain and its elements appear orange and PDB power domain and its elements appear blue.

Figure 65 Hierarchical and Power Domain Boundaries Colored by Power Hierarchy



For more information, see the “Viewing Cell and Power Domain Hierarchies” topic in Design Vision Help.

Creating Atomic Power Domains

To create an atomic power domain, use the `-atomic` option with the `create_power_domain` command. See the following syntax:

```
create_power_domain name -atomic
  [-elements element_list]
  [-exclude_elements exclude_list]
  [...]
```

You must specify the `-atomic` option when you first define the power domain. The tool does not allow updating a non-atomic power domain as atomic, that is, option `-atomic` cannot be specified with `-update`.

However, you can update the extent of an atomic power domain, that is, updating the lists for `-elements` and `-exclude_elements` are allowed.

You can define an empty power domain as atomic. However, you must update the empty power domain with elements before running any action command.

The tool always creates atomic power domains first, independent of the order in which the power domains are defined. Because atomic power domains are processed first, their extents are determined and the tool can identify all the power domains that share extent.

The tool does not allow any instance in the descendant subtree of an atomic power domain to be included in the extent of another power domain, unless:

- the instance name is excluded from the atomic domain or
- the instance name is in the descendant subtree of an instance which is excluded from the atomic domain

The tool has checks in place to verify and report any violations for the preceding cases.

Note:

It is not mandatory to add an instance excluded from a power domain as the root cell of another power domain. There can be cases where the power domain cannot be determined for a few elements in the design. The tool provides an error message for such elements.

Examples

Example 33

```
create_power_domain PD_TOP
  -elements {m2 m1/b1} -include_scope
create_power_domain PD_MID -atomic
  -elements {m1}
  -exclude_elements {m1/b1}
```

In this example, PD_MID is an atomic power domain which is created first. Excluding m1/b1 from the atomic domain allows it to be used in PD_TOP.

Example 34

```
create_power_domain PD_EMPTY -elements {} -atomic
```

You can define an empty power domain as atomic, as in this example. Before you run any action command, you must ensure to update the empty domain so that the domain contains elements.

Example 35

```
create_power_domain PD_TOP -elements {.} -include_scope
                    -exclude_elements {i_core} -atomic

create_power_domain PD_CORE -elements {i_core}
commit_upf
```

As shown in this example, excluding the element on which PD_CORE will be created from atomic PD_TOP is allowed.

Reporting Atomic Power Domains

You can use `report_power_domain` in your design to check which power domains are defined as atomic.

The following is an example of the `report_power_domain` output for an atomic power domain:

```
*****
Report : Power Domain
Design : top
*****

-----
Power Domain      : PD_CORE (atomic)
Current Scope    : / (top scope)
Elements         : core1
Voltage Area     : DEFAULT_VA
Available Supply Nets : VDD_CORE, VDD_ISO, VDD_REG, VDDtop, VSStop
Available Supply Sets  : SS_CORE, SS_ISO, SS_REG_BANK, SS_TOP

Default Supplies           - Power -      - Ground -
Primary                  : VDD_CORE [0.95, switchable]
                           VSStop [0.00]
Isolation                : --
Retention                : --
...
...
```

Hierarchical Flow Support for Atomic Power Domains

This topic discusses the hierarchical flow support for atomic power domains with examples.

Top-Down Hierarchical Flow

There is no special handling of atomic power domains in the top-down hierarchical flow. The tool uses the existing methodology of characterizing power domains for characterizing atomic power domains.

The following rules apply:

- If the atomic power domain is defined at or below the scope that gets characterized, then the atomic power domain is characterized to the BLOCK as is.
- If the cell being characterized is the root cell of an atomic power domain:
 - Domain split happens and an atomic power domain with same name is characterized to BLOCK.
 - As is the case currently for non-atomic power domains, no explicit `merge_domain` attribute is needed.
- If the cell being characterized is not a root cell of an atomic power domain:
 - Domain split happens and an atomic power domain with same name is characterized to BLOCK.
 - The `merge_domain` attribute is set for this cell in TOP UPF.

Bottom-Up Hierarchical Flow

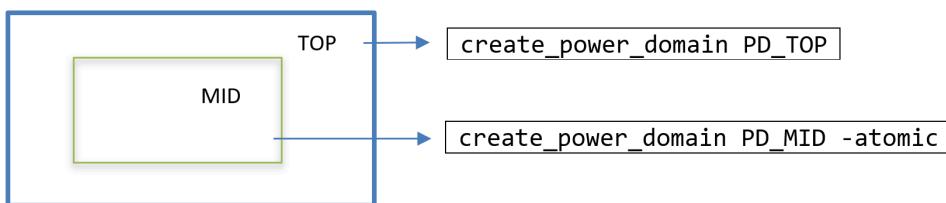
In the bottom-up hierarchical flow, the tool propagates constraints from BLOCK to TOP design. There are two scenarios with respect to the handling of power domains.

1. Non-Domain Merging

In this scenario, atomic power domains defined in BLOCK are propagated to BLOCK scope. If this propagation violates the rule that elements in the extent of an atomic power domain cannot be defined as the root cell of another power domain, the tool reports a warning message to indicate this violation.

Success Case Example

Consider the following example:



In this case, during constraint propagation, the atomic power domain from BLOCK PD_MID gets propagated to the TOP design. The following syntax shows the full UPF post constraint propagation:

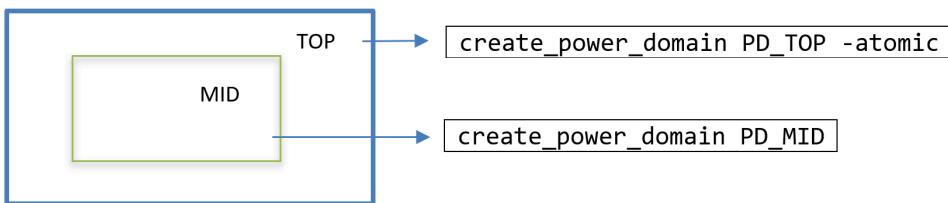
```

create_power_domain PD_TOP -include_scope
create_power_domain PD_MID -elements mid_inst -scope mid_inst -atomic
    
```

This full UPF does not violate any rule of atomic power domains. So, during the execution of the synthesis command, the tool does not generate any warning or error messages related to atomic power domains.

Error Case Example

Consider another example:



In this case, during constraint propagation, the non-atomic power domain from BLOCK PD_MID gets propagated to the TOP design. The following syntax shows the full UPF post constraint propagation:

```

create_power_domain PD_TOP -include_scope -atomic
create_power_domain PD_MID -elements mid_inst -scope mid_inst

```

This full UPF violates the rule (of atomic power domains) that elements in the extent of an atomic power domain cannot be defined as the root cell of another power domain. The element mid_inst, part of atomic power domain PD_TOP (it is not excluded in PD_TOP), is the root cell of a non-atomic power domain mid_inst/PD_MID. During constraint propagation, the tool reports the following warning message and continues the flow:

Warning: Element mid_inst in the extent of atomic power domain PD_TOP has been defined as root cell of another power domain mid_inst/PD_MID

You must fix the UPF by excluding mid_inst from the atomic power domain PD_TOP. The tool then allows mid_inst to be the root cell of mid_inst/PD_MID.

2. Domain Merging

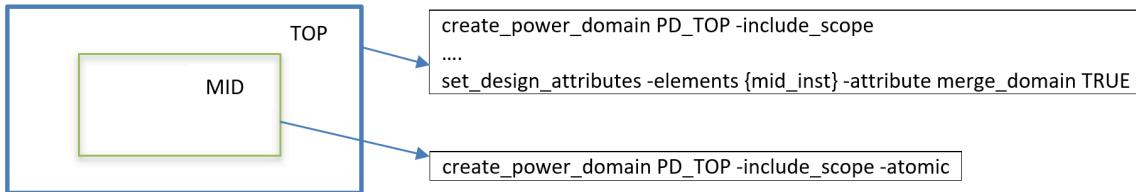
The following table describes the four possible combinations of TOP and BLOCK power domains, for domain merging during constraint propagation, and the tool behavior in all these combinations:

	Non-atomic power domain in BLOCK	Atomic power domain in BLOCK
Non-atomic power domain in TOP	Existing behavior	Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 1: TOP is Non-Atomic and BLOCK is Atomic .

Atomic power domain in TOP	Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 2: TOP is Atomic and BLOCK is Non-Atomic .	<ul style="list-style-type: none"> • Matching domains: Only TOP atomic power domain is retained in the final UPF. See Scenario 3: TOP and BLOCK are Atomic and Domain Merging Succeeds. • Non-matching domains: Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 4: TOP and BLOCK are Atomic and Domain Merging Fails.
----------------------------	---	--

Scenario 1: TOP is Non-Atomic and BLOCK is Atomic

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the non-atomic power domain in the TOP design and the power domains do not match. So, domain merging fails with the following message and the tool retains both the domains:

```
Error: Unable to merge domain PD_TOP and mid_inst/PD_TOP because
mid_inst/PD_TOP is atomic but PD_TOP is not atomic. (UPF-168)
```

The following is the full UPF post constraint propagation:

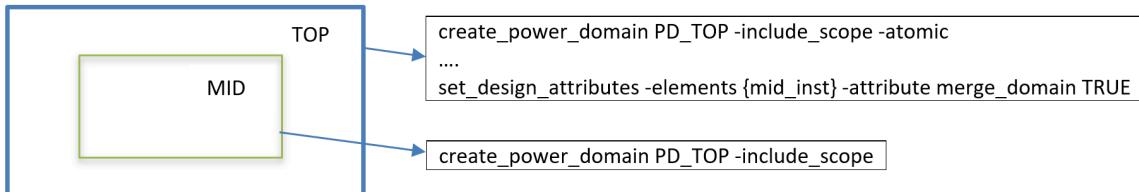
```

create_power_domain PD_TOP -include_scope
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst -atomic
...
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
  
```

Since the tool generated an error message during constraint propagation, you must update the UPF before proceeding with the flow.

Scenario 2: TOP is Atomic and BLOCK is Non-Atomic

Consider the following example:



In this scenario, during constraint propagation, the tool compares the non-atomic power domain from BLOCK with the atomic power domain in the TOP design and the power domains do not match. So, domain merging fails with the following message and the tool retains both the domains:

```
Error: Unable to merge domain PD_TOP and mid_inst/PD_TOP because PD_TOP
is atomic but mid_inst/PD_TOP is not atomic. (UPF-168)
```

The following is the full UPF post constraint propagation:

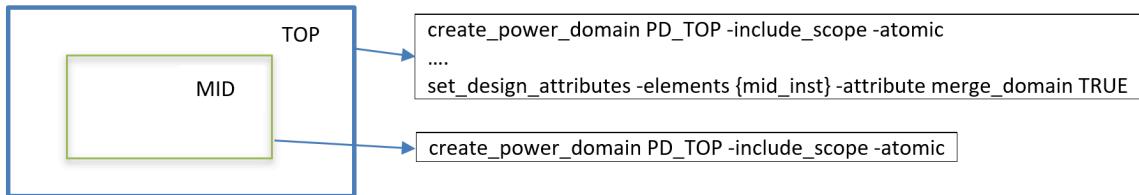
```

create_power_domain PD_TOP -include_scope -atomic
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst
...
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
  
```

In this scenario too, since the tool generated an error message during constraint propagation, you must update the UPF before proceeding with the flow.

Scenario 3: TOP and BLOCK are Atomic and Domain Merging Succeeds

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the atomic power domain in the TOP design and the power domains say all their properties match. So, domain merging succeeds and the tool drops the domain from BLOCK.

The following is the full UPF post constraint propagation:

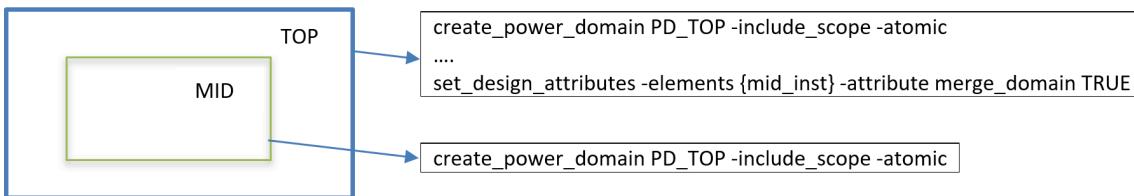
```

create_power_domain PD_TOP -include_scope -atomic
...
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
  
```

This full UPF does not violate any rule of atomic power domains. So, during the execution of the checker/synthesis commands, the tool does not generate any warning/error messages related to atomic power domains.

Scenario 4: TOP and BLOCK are Atomic and Domain Merging Fails

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the atomic power domain in the TOP design and say some of their properties do not match. So, domain merging fails and the tool retains both the domains.

The following is the full UPF post constraint propagation:

```

create_power_domain PD_TOP -include_scope -atomic
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst -atomic
...
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
  
```

In this scenario too, since there is an error during constraint propagation, you must update the UPF before proceeding with the flow.

Creating Supply Ports

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

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

You can optionally use the `-direction` option of the `create_supply_port` command to specify the port direction and to define how the state information is propagated through the supply network when connected to the port. The option arguments are as follows:

- The `in` argument specifies an input port (the default). The state information of the external supply net connected to the port is propagated into the domain.
- The `out` argument specifies an output port. The state information of the internal supply net connected to the port is propagated outside the domain.

- The `inout` argument specifies an input/output port. Top-level inout supply ports and leaf-level inout supply ports are considered to be drivers.
- The `internal` argument specifies an internal port. Use this value for ports that connect virtual nets in the design. If a UPF supply net only connects to leaf ports with the internal direction, the tool recognizes that this net should not exist in the physical implementation. An example usage is to connect block supplies that are known to switch together. Using the `internal` designation allows you to provide a name that can then be referenced by other commands such as the `connect_supply_net` or `find_objects` commands.

The UPF standard requires a simple name for the `port_name` argument of the command. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

The following example shows how to create supply ports VDD1, VDD2 and VDD3 and GND at the top level of the design hierarchy:

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

To create the supply ports VDD1, VDD1g and GND in the power domain PD1, use the `create_supply_port` command as follows:

```
create_supply_port VDD1 -domain PD1
create_supply_port VDD1g -domain PD1
create_supply_port GND -domain PD1
```

To create the supply ports VDD2 and GND in the power domain PD2 and VDD3 and GND in power domain PD3, use the `create_supply_port` command as follows:

```
create_supply_port VDD2 -domain PD2
create_supply_port GND -domain PD2
create_supply_port VDD3 -domain PD3
create_supply_port GND -domain PD3
```

Note:

Connectivity is not defined when the supply port is created. To define connectivity use the `connect_supply_net` command.

You can use the `-direction` option of the `create_supply_port` command to define how the state information is propagated through the supply network when connected to the port. If the port is an input port (the default), the state information of the external supply net connected to the port is propagated into the domain. Similarly, for an output port, use the `out` property with the `-direction` option. In this case, the state information of the internal supply net connected to the port is propagated outside the domain.

You can also specify the `inout` property of the `-direction` option so that top-level inout supply ports and leaf-level inout supply ports are considered as the driver.

Adding Port State Information to Supply Ports

The `add_port_state` command adds state information to a supply port. This command specifies the name of the supply port and the possible states of the port. The first state specified is the default state of the supply port. The port name can be a hierarchical name. Each state is specified as a state name and the voltage level for that state. The voltage level can be specified as a single nominal value, set of three values (minimum, nominal, and maximum), or 0.0, or the keyword off to indicate the off state. The state names are also used to define all possible operating states in the Power State Table.

Note that supply states specified at different supply ports are shared within a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

[Example 36](#) shows the definition of states for the power nets:

Example 36 Defining the States of the Power Nets

```
dc_shell> add_port_state header_sw/VDD -state {HV 0.99} -state {LV 0.792}
          -state {OFF off}
```

[Example 37](#) shows the definition of states for the ground nets:

Example 37 Defining the States of the Ground Nets

```
dc_shell> add_port_state footer_sw/VSS -state {LV 0.0} -state {OFF off}
```

[Example 38](#) has the HV1_1 and HV2_1 states with the same voltage value, 1.2, on the VDD1 supply port. The duplicate port states are useful in hierarchical flow, where the top level and block-level ports have different state names but the same voltage value.

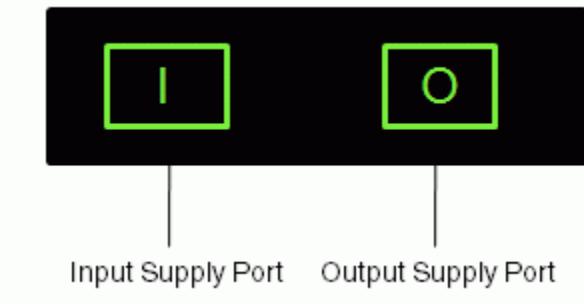
Example 38 Defining Duplicate Port States

```
dc_shell> add_port_state VDD1 -state {HV1_1 1.2} -state {HV2_1 1.2}
```

Representation of Supply Ports in the UPF Diagram View

In the UPF diagram view, a supply port is represented by a bounding box. A letter in the bounding box indicates the direction of the port, as shown in [Figure 66](#).

Figure 66 Representation of Power Supply Port in the UPF Diagram



The UPF diagram displays all the supply ports in the current design and its subdesigns. It also shows the connectivity of the supply ports with the supply nets, their locations, and the power domains where they belong.

Supply ports are located on the border of the power domain where they belong. They are located at the top or at the bottom boundary of the power domain, depending on the supply net connected to the supply ports. In addition, input ports are located on the left side, and the output ports are located on the right side.

Creating Supply Nets

A supply net connects supply ports or supply pins. To create a supply net, use the `create_supply_net` command.

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

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

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

The UPF standard requires a simple name for the `net_name` argument. By default, the Power Compiler tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

All supply nets, including the ground, must be assigned an operating voltage value. If any supply net does not have an assigned operating voltage, the tool issues a UPF-057 error message during the execution of the `compile_ultra` command. Before compiling the design, use the `check_mv_design -power_nets` command to ensure that operating

voltages are defined for all the supply nets. For more details, see [Examining and Debugging UPF Specifications](#).

The operating voltage that you have already set cannot be removed. However, you can override the existing settings by using the `set_voltage` command again.

Creating Custom Resolution Functions

You can use the `create_supply_net` command to create custom resolution functions. If the string following the `-resolve` option is not `unresolved`, `parallel`, `one_hot`, or `parallel_one_hot`, the tool assumes the string to be a custom resolution function name. This string is parsed and saved to the output UPF.

The following example specifies a custom resolution function:

```
create_supply_net VDD -resolve my_package::my_resolution
```

If a custom resolution function is specified on a net, the tool allows multiple drivers on that net and checks that all connected nets have the same resolution function. If there are different resolution functions specified on the connected nets, the tool issues a UPF-099 error message indicating an inconsistent resolution type.

Specifying Primary Supply Nets for a Power Domain

To define the primary power supply net and primary ground net for a power domain, use the `set_domain_supply_net` command.

Every power domain must have one primary power and one ground connection. When a supply net is created it is not a primary supply net. You must use the `set_domain_supply_net` command to designate the specific supply net as the primary supply net for the power domain. All cells in a power domain are assumed to be connected to the primary power and ground net of the power domain. If the power or ground pins of a cell in a power domain, is not explicitly connected to any supply net, the power or ground pin of the cell is assumed to be connected to the primary power or ground net of the power domain to which the cell belongs.

The following example shows the commands to specify VDD and GND nets as the primary power and ground net, respectively, of the PD_TOP power domain.

```
dc_shell> set_domain_supply_net -primary_power_net VDD \
    -primary_ground_net GND PD_TOP
```

Note:

If you use supply sets to define the primary supply and ground, the supply nets that you specify must belong to the same supply set. Otherwise the tool issues an error message. For more details see, [Specifying Supply Sets](#).

Representing Supply Nets in the UPF Diagram View

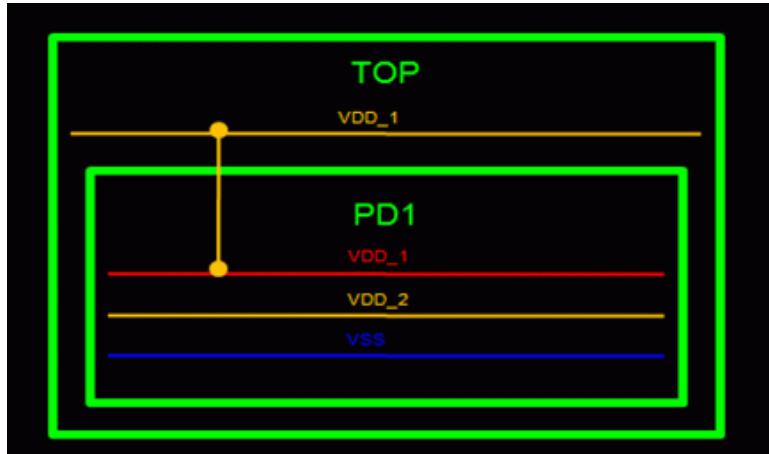
In the UPF diagram view, a supply net is represented by a line or a line segment. Different colors are used to differentiate the type of the net, as shown in [Table 19](#).

Table 19 Colors Used to Represent Types of Net Segments

Color	Net segment
Red	Primary power net
Blue	Primary ground net
Yellow	All other net segments

As shown in [Figure 67](#), the UPF diagram view displays all the supply nets in the current design and the current design's subdesigns, and their supply net connections.

Figure 67 Representation of Types of Power Supply Nets in the UPF Diagram



The location of the supply nets in the diagram is based on the location of the power domains where they belong and also on the type of the supply net. Each power domain that a supply net belongs to contains a line segment indicating the supply net.

Horizontal segments represent supply nets inside the power domain. Vertical segments represent nets that are reused in multiple power domains and that are connected to another object, such as a supply port or a power switch.

Power supplies extend down from the top of the power domain, and ground nets extend up from the bottom of the power domain.

In [Figure 67](#), the VDD_1 net is the primary supply net of PD1 power domain. However, it is not the primary supply net of the power domain TOP. Similarly, VSS is the primary ground net of power domain PD1.

Connecting Supply Nets

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

You can also use the `connect_supply_net` command to connect to the internal PG pins of macro cells containing fine-grained switches. For more information about macro cells with fine-grained switches, see [Macro Cells With Fine-Grained Switches](#).

The UPF standard requires a simple name for the `supply_net_name` argument. By default, the Power Compiler tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

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

```
connect_supply_net GND_NET -ports GND
connect_supply_net GND_NET -ports {B1/GND B2/GND B3/GND} GND
```

You can also use the function of a supply set with the `connect_supply_net` command, as shown in the following example:

```
create_supply_set ss
connect_supply_net ss.ground -ports {B1/GND}
```

Use the `create_supply_net -resolve parallel` command when

- A supply net connects to the internal PG pins of more than one macro cell with a fine-grained switch.
- A supply net is associated with a supply set group that has multiple drivers at the scope of the supply net. For more information about associating supply sets, see [Associating Supply Sets](#).

Note:

The `connect_supply_net` command ignores connections to the pins of physical-only cells.

Interpreting PG Connections From the RTL

You can control the use of PG connections present in the RTL by using the following variables:

- The `dc_allow_rtl_pg` variable enables the tool to obtain PG information from the RTL. The default is `false`.
- The `dc_allow_rtl_pg_to_signal_pins` variable allows PG nets to connect to all signal pins. The default is `true`.
- The `dc_allow_rtl_pg_to_analog_pins` variable allows PG nets to connect to analog pins. The default is `true`. The behavior is as follows:
 - When the `dc_allow_rtl_pg_to_signal_pins` variable is `true` (the default), the tool ignores the `dc_allow_rtl_pg_to_analog_pins` variable.
 - When the `dc_allow_rtl_pg_to_signal_pins` variable is `false` and the `dc_allow_rtl_pg_to_analog_pins` variable is `true`, PG connections are allowed only to signal pins that have the `is_analog` attribute set to `true` in the cell. The tool drops connections to other signal pins, connects those signal pins to constants, and issues an error message.
 - When the `dc_allow_rtl_pg_to_signal_pins` variable is `false` and the `dc_allow_rtl_pg_to_analog_pins` variable is `false`, PG connections are not allowed to any signal pins. The tool drops connections to the signal pins, connects those pins to constants, and issues an error message.

Converting PG Information in the RTL to UPF

The Power Compiler tool can convert information about PG nets and PG pin connections in the RTL into UPF constraints. The PG nets or pins in the RTL must be declared as ordinary wires or ports. PG connections can only be made to macros, I/O pad cells, and power management cells.

In a flow that uses a UPF, the tool writes out a full PG Verilog netlist by default when you use the `write_file -format verilog -pg` command. This netlist includes PG connections to all cells, including standard cells. If you set the `upf_write_only_rtlpg_to_pg_netlist` variable to `true`, the tool writes only the PG connections that are present in the RTL to the Verilog netlist.

If your flow does not use a UPF, the `upf_write_only_rtlpg_to_pg_netlist` variable has no effect.

Follow these steps to convert the PG information in RTL into UPF constraints for use during synthesis:

1. Set the `dc_allow_rtl_pg` variable to `true` to enable PG information from the RTL. By default, PG connections are allowed to all signal pins.
2. (Optional) Set the `dc_allow_rtl_pg_to_signal_pins` variable to `false` to allow PG connections to analog signal pins but not to any other signal pins.
3. (Optional) Set the `dc_allow_rtl_pg_to_signal_pins` and `dc_allow_rtl_pg_to_analog_pins` variables to `false` to disallow PG connections to all signal pins.
4. (Optional) Set the `upf_write_only_rtlpg_to_pg_netlist` variable to `true` to specify that the output netlist should contain only the PG connections present in the RTL.
5. Read the RTL design.
6. Link the design.
7. Load the UPF file.
8. Run the `convert_pg` command to resolve conflicts between the PG connections in the RTL and the power intent specifications in the UPF. This command translates all PG connections into UPF commands.
9. Run the `write_file -format verilog -pg -output filename` command to write the PG Verilog netlist.
10. Run the `save_upf filename` command to save the modified UPF.

Note:

When the RTL design has PG connection details and the power constraints are specified in UPF, you must specify the `convert_pg` command before you run the following commands:

- `compile_ultra`
- `insert_mv_cells`
- `insert_dft`
- `dft_drc`
- `analyze_mv_design`
- `check_mv_design`
- `save_upf`

For example, consider the netlist in [Example 39](#), which contains instantiations of both macros and standard cells (inverters). The PG pins of the macros are connected to PG nets. However, the PG pins of the standard cells are not connected.

Example 39 Input Netlist Example

```
module TOP (VDD1, VSS, IN1, IN2, ...);
  input VDD1, VSS;

  MACRO macro_inst_1 (.VDD(VDD1), .VSS(VSS), .SIGNAL_PIN(IN1), ...);
  MACRO macro_inst_2 (.VDD(VDD1), .VSS(VSS), .SIGNAL_PIN(IN2), ...);

  INV inv_inst_1 (.IN(IN3), .OUT(OUT1));
  INV inv_inst_2 (.in(IN4), .OUT(OUT2));

endmodule
```

Suppose that you load the following UPF file, which specifies one power domain (PDTOP) with power net VDD2 and ground net VSS:

```
create_supply_set SS
create_supply_net VDD2
create_supply_net VSS
create_supply_set SS -function {power VDD2} -update
create_supply_set SS -function {ground VSS} -update
create_power_domain PDTOP -supply {primary SS}
```

If you run the `convert_pg` command followed by the `write_file -format verilog -pg -output filename` command, the tool writes the PG netlist shown in [Example 40](#) by default. This is a full PG netlist with connections to the PG pins of all cells, including the standard cells (the inverters). These PG pins are connected to the primary power and ground nets of the domain (VDD2 and VSS).

Example 40 Default PG Netlist

```
module TOP (VDD1, VDD2, VSS, IN1, IN2, ...);
  input VDD1, VDD2, VSS;

  MACRO macro_inst_1 (.VDD(VDD1), .VSS(VSS), .SIGNAL_PIN(IN1), ...);
  MACRO macro_inst_2 (.VDD(VDD1), .VSS(VSS), .SIGNAL_PIN(IN2), ...);

  INV inv_inst_1 (.VDD(VDD2), .VSS(VSS), .IN(IN3), .OUT(OUT1));
  INV inv_inst_2 (.VDD(VDD2), .VSS(VSS), .IN(IN4), .OUT(OUT2));

endmodule
```

However, if you set the `upf_write_only_rtlpg_to_pg_netlist` variable to `true`, the tool writes only the PG connections that are present in the RTL to the Verilog netlist. The output netlist is the same as the original input netlist shown in [Example 39](#).

Preserving Assign Statements on PG Nets

By default, the Power Compiler tool preserves assign statements on PG nets (for example, a supply net connected to an input supply port and an output supply port) in the PG Verilog netlist and/or UPF, based on your tool flow without or with UPF.

In the flow without UPF, the tool preserves the assign statements by writing them out in the PG Verilog netlist.

In the flow with UPF, when `convert_pg` is run, after loading the UPF, the assign statements present in the RTL are converted to equivalent `create_supply_net`, `create_supply_port`, and `connect_supply_net` commands. These commands are written out in the UPF. Additionally, if you write the PG Verilog netlist, the assign statements are written out in the PG Verilog netlist too.

Example 1: Assign Statement in the Top-Level Module

Consider the following RTL example that has an assign statement in the top-level module:

```
module top (in, out1, out2, vddt);
    input in, vddt;
    output out1, out2;

    macro M1 (.in(in), .out(out1), .VDD(vddt));
        assign out2 = vddt;
    endmacro

endmodule
```

The Power Compiler tool considers the PG net “vddt” to be connected to the PG pin VDD of macro M1 and the PG port “out2”.

In the flow without UPF, the assign statement present in the top-level module is preserved and written out in the PG Verilog netlist. In other words, the PG Verilog netlist looks the same as the preceding RTL. In the non-PG Verilog netlist, supply ports/nets “vddt” and “out2” are not written out.

In the flow with UPF, after `convert_pg`, the UPF has the following commands:

```
create_supply_net vddt
create_supply_port vddt
connect_supply_net vddt -ports {vddt}
connect_supply_net vddt -ports {M1/VDD}
create_supply_port out -dir out
connect_supply_net vddt -ports {out}
```

Example 2: Assign Statement in the Lower-Level Module

Consider the following RTL example that has an assign statement in the lower-level module:

```
module top (in, out1, out2, vddt);
    input in, vddt;
    output out1, out2;

    mid mid_inst (.in(in), .out1(out1), .out2(out2), .vddm(vddt));

endmodule

module mid (in, out1, out2, vddm);
    input in, vddm;
    output out1, out2;

    macro M1 (.A(in), .Z(out1), .VDD(vddm));
        assign out2 = vddm;
    endmacro

endmodule
```

In the flow without UPF, the assign statement present in the lower-level module is preserved and written out in the PG Verilog netlist. In other words, the PG Verilog netlist looks the same as the RTL. In the non-PG Verilog netlist, supply ports/nets “vddt”, “vddm” and “out2” are not written out.

In the flow with UPF, after `convert_pg`, the UPF has the following commands:

```
create_supply_net vddt
create_supply_port vddt
connect_supply_net vddt -ports {vddt}
set_scope mid_inst
create_supply_net vddm
create_supply_port vddm
connect_supply_net vddm -ports {vddm}
connect_supply_net vddm -ports {M1/VDD}
create_supply_port out2 -dir out
connect_supply_net vddm -ports {out2}
set_scope /
connect_supply_net vddt -ports {mid_inst/vddm}
create_supply_net out2
create_supply_port out2 -dir out
connect_supply_net out2 -ports {out2}
connect_supply_net out2 -ports {mid_inst/out2}
```

Specifying Supply Sets

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

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

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can be physically implemented (placed and routed), its supply sets must be refined (associated with actual supply nets).

You can access the functions of the supply set by using the name of the supply set and the name of the function. To access the power function of the supply set *SS*, specify *SS.power*. To access the ground function of the supply set *SS*, specify *SS.ground*.

Creating Supply Sets

A supply set is an abstract collection of supply nets consisting of two supply functions: power and ground. A supply set is domain-independent, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can be physically implemented (placed and routed), its supply sets must be refined, or associated with actual supply nets.

A supply set consists of the following functions:

- Power
- Ground

You can access the functions of the supply set by using the name of the supply set and the name of the function. To access the power function of the supply set *SS*, specify *SS.power*. To access the ground function of the supply set *SS*, specify *SS.ground*.

Creating Supply Sets

To create a supply set, use the `create_supply_set` command. The supply set is created in the current logic hierarchy or the scope.

The UPF standard requires a simple name for the *supply_set_name* argument. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

The following example shows how to create a supply set and associate it with the primary power supply of a power domain:

```
create_supply_set primary_supply_set
create_power_domain PD_TOP
set_domain_supply_net PD_TOP \
    -primary_power_net primary_supply_set.power \
    -primary_ground_net primary_supply_set.ground
```

Note:

When you specify a supply set as the primary power and ground supply of the power domain, both the primary and the ground supply must belong to the same supply set.

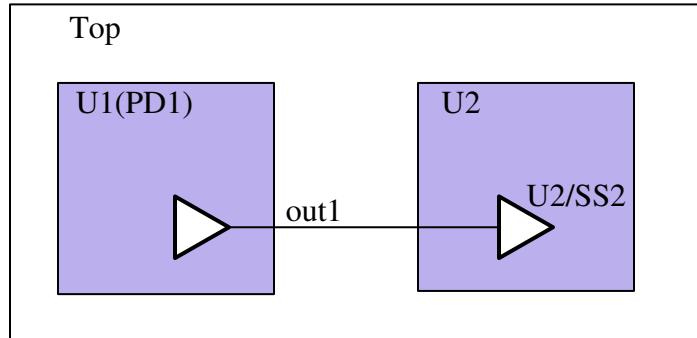
In the UPF Diagram view, a supply set does not appear visually in the diagram. Only the supply nets of a supply set appear in the diagram. Supply nets of a supply set and domain-independent supply nets are implicitly available anywhere from their scope downward in the design.

Reference-Only Supply Sets

When using the hierarchical flow and separating blocks, you might have supply sets that reside outside the current block. In order to refer to these supply sets outside the block after you run the `characterize` command, the tool creates a reference-only supply set. This supply is meant only to resolve supply references in strategies when using the hierarchical flow. You cannot use the supply to power actual cells.

For the example in [Figure 68](#), the U1 power domain has an isolation strategy that refers to a local supply in U2 as a sink supply. If you split U1 into a separate block, the UPF for U1 would need to replace the U2/SS2 in the U1 UPF because U2/SS2 is not valid when the UPF for each block is separated. Meanwhile, the real supply that powers U2/SS2 is only available to power cells in U2 and is not visible outside of U2. After you integrate the design with U1, there needs to be a way to associate U2/SS2 with its replacement in U1. To do this, the tool creates a reference-only supply for U1.

Figure 68 Reference-Only Supply Set Example



```

create_power_domain Top
set_scope U2
create_supply_set SS2
set_scope
create_power_domain PD1 -scope U1 -elements U1
set_isolation iso -domain U1/PD1 -sink U2/SS2

```

To create a reference-only supply, the tool creates the supply set and then marks it as reference-only using the design attribute `reference_only`. For example,

```

create_supply_set SS2'
set_design_attributes -elements . -attribute reference_only {SS2'}

```

The power states of U2/SS2 are also copied to SS2' in the UPF for U1. The receiver supply of the `out1` port is also set to SS2' as follows:

```

set_port_attributes -elements out1 -receiver_supply SS2'

```

These additions to the U1 UPF ensure that when U1 is optimized separately, it has all the information for correct multivoltage cell insertion in U1. In the Top UPF, a `set_equivalent` command is added to establish the relationship between SS2' and U2/SS2:

```

set_equivalent -sets {U1/SS2' U2/SS2}

```

Note:

The `add_supply_state` command can be used with the `add_pst_state` command but not with the `add_power_state` command.

Creating Supply Set Handles

When you create a power domain, the following supply set handles are created by default:

- primary
- default_isolation
- default_retention

In addition to these predefined supply set handles, you can define supply set handles by using the `-supply` option of the `create_power_domain` command. To associate multiple supply sets with a power domain, use the `-supply` option multiple times.

Supply set handles are created at the scope of the power domain and are available for use in the power domains that are at the same or lower scope than the power domains where they are created. Use the following naming convention to refer to a supply set handle: `power_domain_name.supply_set_handle`. When a power domain is deleted, its supply set handles are also deleted.

To disable the creation of supply set handles while creating the power domain, set the `upf_create_implicit_supply_sets` variable to `false` before you load the UPF file.

Note:

After loading the UPF file, the `upf_create_implicit_supply_sets` variable becomes a read-only variable and you can no longer change its value.

You can also specify the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command to restrict the availability of the supplies in the power domain. For more information about using the `extra_supplies_#` keyword, see [Restricting Supply Sets Available to a Power Domain](#).

The following example shows how to create a power domain and associate a supply set with the power domain:

```
# Create the supply sets
create_supply_set primary_supply_set

# Create power domain and associate it with the supply set
create_power_domain PD1 -supply {primary primary_supply_set}
```

When the well-bias mode is enabled, all four supply functions (power, ground, n-well, and p-well) are created if any of the following conditions are true:

- The supply set is used as a domain's primary supply in the design
- The n-well or p-well function of the supply set is explicitly resolved to a supply net

- The n-well or p-well function of the supply set is referred to in a UPF command or PG netlist
- The supply set is associated with another supply set for which the tool has created all four functions

Restricting Supply Sets Available to a Power Domain

Supply sets are domain-independent and can only be updated with domain-independent nets. To restrict the supply sets available to a power domain, use the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
dc_shell> create_power_domain SUB_DOMAIN \
           -supply {extra_supplies_1 supply_set1} \
           -supply {extra_supplies_2 supply_set2} -elements mid1/PD_MID
```

Alternatively, if you do not want the power domain to use extra supply nets other than those that are already defined in other strategies, specify the `extra_supplies ""` keyword (without the index) with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
dc_shell> create_power_domain PD_MID -scope mid1 \
           -supply {extra_supplies ""}
```

It is an error to use both the `extra_supplies_#` and `extra_supplies ""` keywords simultaneously.

By default, a power domain can use domain-independent supply nets and supply nets defined in the power domain. However, when you define supply sets with the `extra_supplies_#` keyword, the power domain is restricted to use

- The primary supply of the power domain
- The supplies listed as `extra_supplies_#`
- The supplies specified by the isolation strategy of the power domain
- The supplies specified by the retention strategy of the power domain
- The supplies defined or reused as domain-dependent supplies in the power domain

Refining Supply Sets

To redefine the functions of a supply set, use the `-update` option of the `create_supply_set` command. You must use the `-update` and the `-function` options together, to associate the function names with the supply nets or ports.

The following example shows how you use the `-update` option to associate supply nets to the functions of the supply set:

```
create_power_domain PD_TOP
create_supply_net TOP_VDD
create_supply_net TOP_VSS
create_supply_set supply_set \
    -function {power TOP_VDD} \
    -function {ground TOP_VSS} \
    -update
```

The following rules apply, when you update a supply set with a supply net:

- **Voltage rule**

The voltage of the supply set handle must match with the voltage of the supply net with which the supply set is updated.

If voltage is not specified for the supply net, then after the update, the voltage on the supply set handle is inferred as the voltage of the supply net.

- **Function rule**

The supply set function must match with the function of the supply net with which the supply set is updated.

The tool issues an error message when,

- The ground handle of a supply set is used to update power handle of another supply set and vice versa.
 - The supply net updated with the ground handle of a supply set is connected to a power supply port or pin of a power object, such as a power domain, and vice versa.
- **Scope rule**

The scope of supply set must match with the scope of the explicit supply net with which the supply set is updated.

- **Availability rule**

The explicit supply net with which the supply set is updated, must be domain-independent.

- **Connection rule**

The explicit supply net with which the supply set is updated, should not be connected to a driver port when the supply set handle is connected to a driver port unless a resolution function is defined for the explicit supply net.

- Conflicting supply state names rule

A supply set handle cannot be updated with a supply net or supply set if their power states are not identical.

- Valid power-state-table rule

When a number of supply sets are updated to the same supply net, only one supply set can be present in the power state table.

Associating Supply Sets

To associate a supply set with another predefined supply set or supply set handle or to associate a list of explicit and implicit supply sets, use the `associate_supply_set` command. When two supply sets are associated, the tool considers the two supply sets to be connected, and their functions resolve to the same supply nets.

Using the `-handle` option is optional. The following commands are equivalent:

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

The `associate_supply_set` command accepts either simple or hierarchical names and accepts a list of supplies to associate.

You can associate two or more supply sets as follows:

```
dc_shell> associate_supply_set {SS1 SS2 mid/ss3}
```

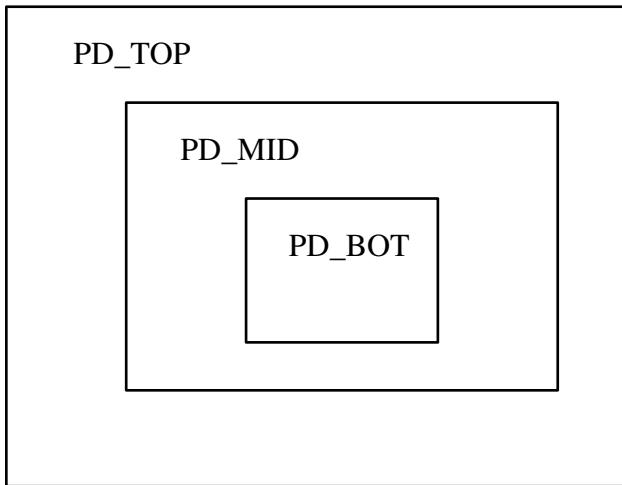
Each of the supply set functions (power, ground, nwell, and pwell) of the associated supply sets are treated as the same supply net or connected supply net.

Associating supply sets with an unequal number of functions causes the supply set with the lesser functions to inherit the remaining functions from the associated supply set. For example,

```
set_design_attributes -elements {.} -attribute enable_bias true
# Creates a two-function supply set
create_supply_set SS1
# Creates a four-function supply set
create_power_domain PD1
# Promotes SS1 to four functions
associate_supply_set {SS1 PD1.primary}
```

You can associate supply sets across different scopes. [Figure 69](#) illustrates the three scopes.

Figure 69 Associated Supplies Across Three Scopes



The following script associates the supplies across the three scopes in [Figure 69](#):

```

create_power_domain PD_TOP -supply {primary SSTop}
create_power_domain PD_MID -supply {primary SSMid}
create_power_domain PD_BOT -supply {primary SSBot}
associate_supply_set {SSTop mid/SSMid mid/bot/SSBot}
  
```

Rules for Associating Supply Sets

The following rules apply, when you associate a supply set with a supply set handle.

- Associating a supply set handle to a supply set can be done only one time.
- Associating a supply set handle to a supply set should not cause circular associations.
- User-defined supply sets cannot be specified with the `-handle` option of the `associate_supply_set` command.
- The supply set handle specified with the `-handle` option of the `associate_supply_set` command must be at the same or below the scope of the specified supply set.
- While associating a supply set handle to supply set, the supply set must be available in the power domain where the supply set handle is available.

Refining Bias Supply Functions Automatically

When you set the `enable_bias` design attribute to the `derived` value, the tool automatically inherits bias functions for every supply set in the scope where `enable_bias` is set to `derived`, based on the power and ground functions, if the bias functions are not

already defined. This means that, you need not specify the bias functions explicitly and if the n-well and p-well functions are the same as the power and ground respectively, this setting saves your effort to specify supply sets with all the four functions in the scope where bias is enabled.

```
dc_shell> set_design_attributes -elements { . }
           -attribute enable_bias derived
```

Or

```
dc_shell> set_design_attributes -elements {a b}
           -attribute enable_bias derived
```

Note:

- Setting `enable_bias` to `derived` is analogous to setting it to `true` because, in both cases, the `enable_bias` attribute is set and bias functions are required for all supply sets. Thus, all rules that currently apply for mixing bias and non-bias blocks with `enable_bias` set to `true` hold for `enable_bias` `derived` as well.
- The interaction of blocks with `enable_bias` `derived` and `enable_bias` `false` is identical to the interaction of blocks with `enable_bias` `true` and `enable_bias` `false`.

For n-well only support, the tool automatically inherits only the n-well function from the power function because, in this case, the p-well function does not exist.

To save the UPF file with the derived bias functions, set the `upf_track_bias_supply_net_resolution` variable to `true`:

```
dc_shell> set upf_track_bias_supply_net_resolution true
```

The value of this application option is `false` by default.

Example 1: No Bias Functions Defined

Consider the following UPF where there is no bias function defined in the supply sets, and the supply set is not resolved to supply nets:

```
#UPF
set_design_attributes -elements . -attribute enable_bias derived

create_supply_set SS1
create_power_domain TOP
associate_supply_set SS1 -handle TOP.primary
```

Since the `enable_bias` attribute is set to `derived`, in the derived UPF, the n-well and p-well functions of SS1 are the same as the power and ground functions. The same is also reflected in the `report_power_domain` command results.

```
#Derived UPF
set derived_upf true
create_supply_set SS1 -function {nwell SS1.power}
    -function {pwell SS1.ground} -update
set derived_upf false

report_power_domain:
Power Domain          : TOP
Current Scope         : top
Elements              : <top_level>
Available Supply Nets: :
Available Supply Sets : SS1, TOP.primary

Connections -- Power -- -- Ground -- -- Nwell -- -- Pwell --
Primary:      SS1.power      SS1.ground      SS1.power      SS1.ground
```

Example 2: Implicit Supply Sets With Resolved Power and Ground

This example shows the implicit supply set TOP.primary is updated for both power and ground, when `enable_bias` is set to `derived`. In the derived UPF, the n-well and p-well functions of it are updated to VDD and VSS, respectively.

```
#UPF
set_design_attributes -elements . -attribute enable_bias derived

create_supply_net VDD
create_supply_net VSS
create_power_domain TOP
create_supply_net TOP.primary -function {power VDD} -function {ground
VSS} -update

#Derived UPF
set derived_upf true
create_supply_set TOP.primary -function {nwell VDD}
    -function {pwell VSS} -update
set derived_upf false

report_power_domain:
Power Domain          : TOP
Current Scope         : top
Elements              : <top_level>
Available Supply Nets: VDD, VSS
Available Supply Sets : SS1, TOP.primary

Connections -- Power -- -- Ground -- -- Nwell -- -- Pwell --
Primary:      VDD           VSS           VDD           VSS
```

Example 3: Implicit Supply Sets With Unresolved Power and Ground

See the following example for an implicit supply set with unresolved power and ground:

```
#UPF
set_design_attributes -elements . -attribute enable_bias derived

create_power_domain TOP

#Derived UPF
set derived_upf true
create_supply_set SNPS_RESOLVE_TOP_primary
create_supply_set SNPS_RESOLVE_TOP_primary
    -function {nwell SNPS_RESOLVE_TOP_primary.power}
    -function {pwell SNPS_RESOLVE_TOP_primary.ground} -update
associate_supply_set {SNPS_RESOLVE_TOP_primary TOP.primary}
set derived_upf false

report_power_domain
Power Domain           : TOP
Current Scope          : top
Elements               : <top_level>
Available Supply Nets : 
Available Supply Sets  : SNPS_RESOLVE_TOP_primary,TOP.primary

Connections -- Power -- -- Ground -- -- Nwell -- -- Pwell --
Primary:   SNPS_RESOLVE_TOP_primary.power
            SNPS_RESOLVE_TOP_primary.ground
            SNPS_RESOLVE_TOP_primary.power
            SNPS_RESOLVE_TOP_primary.ground
```

Example 4: N-Well Only Support

In this example, the design is detected by the tool as n-well-only design, when you set `enable_bias` is set to `derived`. The tool, therefore, only derives the n-well function from the power function; no p-well function is derived.

```
#UPF
set_design_attributes -elements . -attribute enable_bias derived

create_supply_set SS1 -function {power VDD} -function {ground VSS}

#Derived UPF
set derived_upf true
create_supply_set SS1 -function {nwell VDD} -update
set derived_upf false
```

Defining the Power States for a Supply Set

Power states are attributes of a supply set. The supply nets of a supply set can be at different power states at different times. Using the `add_power_state` command, you can define one power state for all those supply nets of the supply set that always occur together. For each power state of the supply set, you must use one `add_power_state` command. By default, the undefined power states are considered illegal states.

The Power Compiler tool automatically accepts the UPF 2.0 or 2.1 version of the `add_power_state` command. You do not have to specify any variables.

For example, both of the following are accepted:

```
add_power_state SS1 -state S1 {-supply_expr {power=={OFF}}}  
add_power_state SS1 -state {S1 -supply_expr {power=={OFF}}}
```

Note:

The S1 state can be specified either inside or outside the curly braces.

You can mix the different versions of the `add_power_state` command. However, you have to use the same style for the same objects. For example, if you specify one state for the SS1 supply set using 2.1 style, you can only use the 2.1 style for specifying additional states for SS1.

For example, the following command combination is not allowed:

```
add_power_state SS1 -state S1 {-supply_expr {power=={OFF}}}  
add_power_state SS1 -state {S2 -supply_expr {ground=={OFF}}}
```

Use the `-domain` option to specify an existing simple power domain name. The scope at which the domain is defined must have the `enable_state_propagation_in_add_power_state` design attribute set to `false`. This is required because the `-domain` option refers to supply set states. By default, the `enable_state_propagation_in_add_power_state` attribute is `false`.

Use the `-state` option to specify the name of the power state of the supply set. This name must be written within curly braces.

Use the `-supply_expr` option to specify the power state and the voltage value for the various supply net components of the supply set.

The following restrictions apply to the `-supply_expr` and `-logic_expr` options:

- Supply expressions containing more than one voltage must be listed in ascending order. Also, parentheses are supported and you cannot use the “.” characters in the domain supply set handle.
- The tool supports only alphanumeric characters and the underscore “_” character. Any other character is not allowed. For example, the following state name is an error:

```
add_power_state SS_AO -state \
{ON&A -supply_expr {power == {FULL_ON 1.05}}}
```

The `add_power_state` command supports all seven simulation states. These states are as follows:

- `NORMAL`
- `NOT_NORMAL`
- `CORRUPT`
- `CORRUPT_ON_ACTIVITY`
- `CORRUPT_STATE_ON_CHANGE`
- `CORRUPT_STATE_ON_ACTIVITY`
- `CORRUPT_STATE_ON_CHANGE`

The UPF standard requires a simple name for the `object_name` argument. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

For more information about power state tables, see [Power State Tables](#).

Specifying Supply Expressions

The supply expression specified with the `-supply_expr` option is used to determine the legal states of the supply nets of the supply set during synthesis. The supply expression uses the following syntax:

```
(net == netstate || net == netstate) && net == netstate
```

The net can be power or ground, and the `netstate` syntax must be one of the following:

- `{status}`
- `{status nom}`
- `{status min nom max}`

The status can be `OFF` or `FULL_ON`.

The *min*, *nom*, and *max* values are floating point numbers representing the minimum, nominal, and maximum voltages of the specified state.

If the status is `FULL_ON`, you can specify zero to three voltages. You can defer specifying a voltage and specify it later using the `-update` option. For example,

```
add_power_state -supply SS2 -state ON1 \
    {-supply_expr {power =={FULL_ON} && ground == {FULL_ON}}}
...
add_power_state -supply SS2 -state ON1 \
    {-supply_expr {power == {FULL_ON 1.0} && ground =={FULL_ON 0.0}}} \
    -update
```

The voltages that you specify with a power state are interpreted by the tool as follows:

- When you specify no voltage, the tool assumes that the voltage will be specified at a later time with the `add_power_state -update` command.
- When you specify a single voltage, the voltage value is considered to be the nominal voltage of the associated state.
- When you specify two voltages, the first value is the minimum voltage and the second value is the maximum voltage. The average of the two values is used as the nominal voltage of the power state.
- When you specify three voltages, the first value is the minimum voltage, the second value is the nominal voltage, and the third value is the maximum voltage of the power state. The values must be specified in increasing order, as shown in the following example:

```
add_power_state SS_AO -state \
    {ON_A -supply_expr { (power == {FULL_ON 0.85 0.9 0.95}) } \
    && (ground == {FULL_ON 0}) }
```

Using OR Operator in `add_power_state -supply_expr`

You must set the `upf_allow_or_operator_in_add_power_state_supply_expr` variable to `true`, before loading the UPF, for the tool to process an `add_power_state` command with a combination of `||`, `&&`, and `==` operators, in its `-supply_expr` expression.

Example 1

In the following example, the `-supply_expr` contains one `||` operator of two `==` operations:

```
add_power_state sst -state SST_OFF
    {-supply_expr { power == {OFF} || ground == {OFF} } }
```

Example 2

In this example, the `-supply_expr` contains two `||` operators and one `&&` operator:

```
add_power_state sst -state SST_HI
{-supply_expr { (power == {FULL_ON 1.2} || power == {FULL_ON 0.7}) &&
(ground == {FULL_ON 0.0} || nwell == {FULL_ON 0.5}) } }
```

OR Operator Support in `add_power_state` for Hierarchical Flows

As in the case of a flat flow, for a hierarchical flow too, you can enable the OR operator usage in the `add_power_state` command by setting the `upf_allow_or_operator_in_add_power_state_supply_expr` variable to true.

- Bottom-Up Flow:

You must set the variable consistently for block and top synthesis. However, it is allowed to have the variable disabled (set to `false`) for block synthesis and enabled for top synthesis. While running the `propagate_constraints` command, the tool removes any duplicate supply from the block, which already exists in top. Supplies are considered duplicate if the comparison of their overall state expression with the OR operator matches between block and top.

- Top-Down Flow:

In this flow, a portion of the system PST that is specific to the block gets characterized into the block:

- If the system PST does not have any supply set states, the block gets a derived PST.
- If the system PST has only supply set states, the block gets derived group states.
- If the system PST cannot be fully represented in terms of supply set states, the block gets a combination of a derived PST and a derived group.

Top-Down Flow Example

This example shows the top-down flow behavior when an OR operator is used in `-supply_expr` of `add_power_state` in the full chip UPF.

```
add_power_state SST -state SST_1
{-supply_expr {power == `{FULL_ON, 0.9} && ground == `{FULL_ON, 0.0}}}

set_scope mid

add_power_state SSM -state SSM_1
{-supply_expr {(power == `{FULL_ON, 0.9} || power == `{FULL_ON, 1.0}) && ground == `{FULL_ON, 0.0}}}

set_scope /
```

In this example UPF, the SSM supply set state SSM_1 is using an OR operator in the supply expression.

During `characterize`, the tool creates new supply set states to capture the partial state. They are named using a SNPS_DERIVED prefix and are referenced in the derived group.

```

add_power_state SST -state SST_1
    {-supply_expr {((power==`{FULL_ON,0.9}) && (ground==`{FULL_ON,0.0}))}}
add_power_state SSM -state SSM_1
    {-supply_expr {(((power==`{FULL_ON,0.9}) || (power==`{FULL_ON,1.0})) && (ground==`{FULL_ON,0.0}))}}
add_power_state SSM -state SNPS_DERIVED_1
    {-supply_expr {((power==`{FULL_ON,0.9}) && (ground==`{FULL_ON,0.0}))}}
add_power_state SSM -state SNPS_DERIVED_2
    {-supply_expr {((power==`{FULL_ON,1.0}) && (ground==`{FULL_ON,0.0}))}}
create_power_state_group group
add_power_state -group group -state group_ps_1
    {-logic_expr {SST == SST_1 && SSM == SNPS_DERIVED_1}}
    -state group_ps_2 {-logic_expr {SST == SST_1 && SSM == SNPS_DERIVED_2}}

```

Operator Precedence

The supply expression follows the operator precedence as described in the IEEE 1801-3.1 standard. See the following tables. The precedence between the same class of operators follows operator precedence mentioned in the SystemVerilog LRM.

Table 20 Boolean Operators

Operator	SystemVerilog equivalent	VHDL equivalent	Meaning
!	!	not	Logical negation
~	~	not	Bit-wise negation
==	==	=	Equal
!=	!=	/=	Not equal
&&	&&	and	Logical conjunction
		or	Logical disjunction

Table 21 Operator Precedence

Operator	Precedence
! ~	Highest
== !=	Next highest
&&	Next
	Lowest

Reporting Support

The `report_pst` command reports only the original power state names and voltage values corresponding to those states, by default. You can use the `-voltage_type` option if you want the tool to report the voltage triplets, that is, minimum, nominal, and maximum, in the `report_pst` command output.

Specifying Logic Expressions

The `-logic_expr` option of the `add_power_state` command specifies a Boolean logic expression in terms of logic nets and supply sets. The option has different effects depending on other options used with the command, as follows:

- The `-logic_expr` option used with the `-supply` option specifies an expression in terms of logic nets and supply nets. This usage is primarily for use by downstream simulation tools and does not affect implementation.
- The `-logic_expr` option used with the `-group` or `-domain` options specifies an expression in terms of group power states, domain power states, supply set states, or logic signal states. This usage affects power state definitions.

Logic Expressions With the `-supply` Option

If you use the `-logic_expr` option with the `-supply` option, you can use logic signals and supply nets in the logic expression. In this usage, the `add_power_state` command is parsed by the Power Compiler tool and written to the UPF for use by downstream simulation tools, but the command has no effect in the Power Compiler tool.

In this usage, you can use all the Boolean operators specified in the IEEE 1801 language reference manual, which are listed in [Table 22](#).

The tool supports parentheses in the Boolean logic expression. For example,

```
add_power_state PDTOP -state {STATE2 \
    -logic_expr { (SSTOP == OFF) && (SSMID == ON) } }
```

Table 22 Boolean Operators

Operator	Meaning
!	Logical negation
~	Bitwise negation
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise conjunction
^	Bitwise exclusive disjunction
	Bitwise disjunction
&&	Logical conjunction
	Logical disjunction

You can also use the || (OR) operator for operands of type logic control signals in the -logic_expr of add_power_state for supply set objects. See the following example:

```
add_power_state ss_peri_sw
  -state ON {-logic_expr {RET_A == 1'b1 || PDW_A == 1'b1} }
  -state OFF {-logic_expr {RET_B == 1'b0 || PDW_B == 1'b0} }
```

Note:

The || operator involving supply set states is not supported in -logic_expr of add_power_state. Only || operators involving logical pins/ports are supported in -logic_expr.

The following example shows how power states can be added to a group. In this example, the RUN12 and RUN1 are the names of the group states that are created:

```
create_supply_set SS1
create_supply_set SS2
create_supply_set SS3
add_power_state -supply SS1 -state ON \
    {-supply_expr {power =={FULL_ON 0.8} && {ground == {FULL_ON 0}}}}
add_power_state -supply SS2 \
    -state ON {-supply_expr {power == {FULL_ON 0.8}}} \
    -state OFF {-supply_expr {power == {OFF}}}
add_power_state -supply SS3 \
    -state ON {-supply_expr {power == {FULL_ON 0.8}}} \
    -state OFF {-supply_expr {power == {OFF}}}
create_power_state_group MY_PST
add_power_state -group MY_PST \
    -state RUN12 {-logic_expr {SS1==ON && SS2==ON && SS3==ON}} \
    -state RUN1 {-logic_expr {SS1==ON && SS2==ON && SS3==OFF}}
...

```

Logic Expressions With the **-group** or **-domain** Options

If you use the **-logic_expr** option with either the **-group** or **-domain** option, you can use supply sets, supply groups, power state tables, and logic signals in the logic expression. The expression must use the following syntax:

`object1 == state && object2 == state`

Only binary values (0 or 1) are allowed when specifying logic signals. For example,

```
add_power_state -domain PD1 \
    -state {ON -logic_expr {SS1 == ON && nPWRUP == 1}}
```

Note:

- The **||** operator involving logic control signals and power states (as operands) in **-logic_expr** of **add_power_state -group** or **add_power_state -domain** is not supported.
- The unary negation and **!=** operators are not allowed with **-logic_expr** with **-group** or **-domain**.

The tool uses the relationship between supplies and logic signals to determine the possible power state table states of the supplies. If a state has only logic signals defined in the logic expression, the state is parsed and written to the output UPF, but has no effect in the Power Compiler tool. During power state table merging, if contradictions are found in the logic values of connected nets in states to be merged, the states are dropped.

If you specify a net name for a logic signal, the tool writes the driving pin or port for that net in the output UPF. For example, consider a net N1 driven by pin U1/Z. The input UPF command might be as follows:

```
add_power_state -group PSG1 -state ON {-logic_expr { ... && N1==1 }}
```

The tool writes the following command in the output UPF:

```
add_power_state -group PSG1 -state ON {-logic_expr { ... && U1/Z==1 }}
```

The tool issues an error message if logic inconsistencies are found, including the following:

- Different logic states specified in the same logic expression for nets connected through a buffer or hierarchical port
- The same logic state specified for nets connected through an inverter

Successive Refinement

Using the `-update` option, you can incrementally add more information to an already defined state. The tool supports successive refinement for all types of objects that are supported by the `add_power_state` command.

The general rules for refinement using the `-update` option are as follows:

- The initial definition of the state can contain just the name of the state. It might or might not specify any additional information.
- If a state defined with the `-supply_expr` option is updated with another `-supply_expr` option, then the new definition is the conjunction of the two definitions.

You can update the supply expression in the following two ways:

- If the initial definition contains just the netstate (`FULL_ON` or `FULL_OFF`) for a functional net without voltage, you can update the definition with the voltage.

```
add_power_state -supply SS1 -state ON \
                {-supply_expr {power == FULL_ON}}
add_power_state -supply SS1 -state ON \
                {-supply_expr {power == FULL_ON 1.0}} -update
```

- The initial definition might be one or more functional nets. You can update the definition by specifying more functional nets.

```
add_power_state -supply SS1 -state ON \
                {-supply_expr {power == FULL_ON 1.0}}
add_power_state -supply SS1 -state ON \
                {-supply_expr {ground == FULL_ON 0.0}} -update
```

- If a state defined with the `-logic_expr` option is updated with another `-logic_expr` option, then the new definition is interpreted as the conjunction of the two expressions. For example, the following statements

```
add_power_state SS1 -state ON {-logic_expr {net1}}
add_power_state SS1 -state ON (-logic_expr {net2}) -update
```

are interpreted as

```
add_power_state SS1 -state ON {-logic_expr {net1 && net2}}
```

- If the simstate is specified as `NOT_NORMAL`, then you can update it to any simstate other than `NORMAL`.
- If the simstate is specified as `NORMAL`, `CORRUPT`, or `CORRUPT_ON_ACTIVITY`, you cannot update it to any other simstate.
- A partially defined state can be used in the logic expression of a group or domain.
- You cannot change a legal state to an illegal state.

Correlated Grouping of Supply Voltage Triplets

If the voltage variation for each supply is correlated with, not independent of, the other supplies, you can define the supplies as correlated, so that the tool considers only the minimum with minimum voltages, only nominal with nominal voltages, and only maximum with maximum voltages, without mixing between minimum, nominal, and maximum. This method of analysis is called correlated grouping of voltage triplets.

The Power Compiler tool supports correlated grouping of the minimum, nominal, and maximum voltages specified as triplets in the `add_port_state` and `add_power_state` commands. To define one or more groups of correlated supply nets, use the `correlated_supply_group` attribute with the `set_design_attributes` command. For example, the following command groups VDD1 and VDD2 supply nets into a correlated supply group and sets the supply voltages as correlated triplets:

```
set_design_attributes -elements {.} \
    -attribute correlated_supply_group "{VDD1 VDD2}"
```

You can define the port state and power state table as follows:

```
add_port_state VDD1 -state {HV 0.9 1.0 1.1}
add_port_state VDD2 -state {HV 1.0 1.1 1.1}
add_port_state VSS -state {ON 0.0}

create_pst PST -supplies {VDD1 VDD2 VSS}
add_pst_state HV_STATE -pst PST -state {HV HV ON}
```

The tool analyzes the design behavior with correlated VDD1 and VDD2 supplies, without mixing between minimum, nominal, and maximum voltages.

For more information about using the `set_design_attributes` command, see [Setting Attributes on Hierarchical Cells](#).

Querying for Supply Sets

Use the `get_supply_sets` command with no arguments to return a collection of supply sets available at the current scope. To return the supply sets available in a given power domain or supply net, use the `get_supply_sets` command with the `-of_objects` option, for example:

```
dc_shell> get_supply_sets -of_objects PD_BOT {SS_CORE}
```

Use the `-hierarchical` option to obtain the supply sets available at all hierarchical scopes. For example:

```
dc_shell> get_supply_sets -hierarchical
           {i_core/SS_CORE SS_ISO SS_CORE SS_TOP
            PD_BOT.primary TOP.primary}
```

Limitations

- If you use `-of_objects` with objects other than power domains or supply nets, the tool looks only for power domains or supply nets with the same object name and generates a warning message if it does not find them.
- You cannot use the `-hierarchical` option with a hierarchical name as an expression for patterns. This produces an UPF-984 error message.
- You cannot use `get_supply_sets -filter` pointing to an attribute not defined for a supply set object. If you use an invalid supply set attribute in the `-filter` expression, the tool errors out silently.

Querying for Related Supply Sets

To obtain a collection of related supply sets for a given collection of design pins and ports, use the `get_related_supply_set` command. The command takes a collection of existing design pins and ports, or a list of pattern strings to use to generate such a collection, as input. The command returns a collection of strings corresponding to the names of the related supply sets in the design. You can use the `-quiet` option to suppress any warnings generated during pin or port lookup.

For an object which is a top-level port or a leaf-level pin and if the related supply set can be derived from it, you can use the `get_upf_port_attribute` command with the `UPF_related_supply_set` attribute. The value of this attribute for this object is the resolved supply set.

Always-On Logic

Generally, multivoltage designs have power domains that are shut down and powered up during the operation of the chip while other power domains remain powered up. The control nets that connect cells in an always-on power domain to cells within the shut-down power domain must remain on during shutdown. These paths are referred to as always-on paths.

- [Attributes for Always-On Cells](#)
 - [Always-On Optimization](#)
 - [Always-On Optimization on Feedthrough Nets](#)
 - [Always-On Optimization on Disjoint Voltage Area](#)
 - [Always-On Tie Cells](#)
 - [UPF Support for Custom Always-On Wrapper Cells](#)
 - [Fixing Multivoltage Violations](#)
-

Attributes for Always-On Cells

You can control how the tool handles always-on cells by setting attributes on the cells, as follows:

- The `always_on` Liberty cell attribute

The Power Compiler tool performs always-on optimization only when the target library contains always-on inverters and always-on buffers. To use a specific library cell in the optimization of always-on paths within the shutdown power domains, mark the cell with the `always_on` Liberty attribute. The tool uses only the always-on cells to optimize the always-on paths within the shutdown power domains. The cells that are not marked as always-on are used outside the shutdown power domains.

When you set the `always_on` attribute on a library cell, the tool does not use the library cell for optimization of other types of paths. To use a library cell in both always-on paths and shutdown paths, set the `always_on` attribute only on the cell instances that are present in the shutdown power domains.

- The `pass_gate` cell attribute

The Power Compiler tool can prevent the connection of always-on cells to cells with pass-gate inputs. An always-on cell should not drive a gate that has pass transistors at the inputs (also known as a pass-gate). Pass-gate input cells should be driven by

a standard cell in a shutdown power domain. Therefore, if your library contains any of these cells, you must mark them as pass-gates in each session.

The following example shows how to mark pin A of cell MUX1 with the `pass_gate` attribute.

```
dc_shell> set_attribute [get_lib_pins lib_name/MUX1/A] pass_gate true
```

- The `dc_upf_user_always_on_buffer` cell attribute

You can specify this attribute only for buffer or inverter cells.

Set the attribute to `true` on a user-instantiated always-on buffer to specify that it is an always-on cell. In this case, the tool does not modify the cell's supply voltage and does not issue a warning if the cell drives a net that is not always-on. The UPF must contain a `connect_supply_net` command that connects the cell's PG pin to a supply net.

Set the attribute to `false` on a buffer or inverter to specify that it should not be always-on. In this case, the tool does not derive a backup supply for the cell and does not issue a warning if the cell drives an always-on net.

Always-On Optimization

The Power Compiler tool constrains, marks, and optimizes always-on nets, including feedthrough nets. The tool considers the `mv_make_primary_supply_available_for_always_on` variable when selecting which power supply to use for inserted buffers.

By default, the variable is `true` and the tool uses the domain's primary supply, the related supply net of the load, or the driver pin as the supply net for the inserted buffers. When the variable is set to `true`, the tool inserts regular buffers instead of always-on buffers on feedthrough nets when the primary power supply can be used to power the buffers without introducing electrical violations. To give preference to load and driver supplies, set the `mv_make_primary_supply_available_for_always_on` variable to `false`.

The tool also ensures that no additional isolation or level-shifting violations are introduced by the automatic always-on synthesis. If the isolation strategy is specified with the `-source` and `-sink` options, the tool preserves the original source and sink relationship on these paths.

To determine the supply nets used for the buffers and inverters inserted during always-on synthesis when the variable is set to `true`, the tool applies the following rules, in the specified order:

1. Highest precedence is given to the domain's primary supply.
2. For a load net, when the related supply net of the load is in the same power domain as the net, the related supply net of the load is used.

3. For a driver net, when the related supply net of the driver is in the same power domain as the net, the related supply net of the driver is used.
4. For a feedthrough net with multiple choices of nets, highest precedence is given to the domain's primary supply. The related supply net of the load takes precedence over the related supply net of the driver.

When the `mv_make_primary_supply_available_for_always_on` variable is set to `false`, the tool does not use the domain's primary supply, and instead uses the related supply net of the load or the driver pin as the supply net for the inserted buffers. Otherwise, the determination of the supplies for the inserted buffers is the same. With the variable set to `false`, the tool gives preference to the load and driver supplies and as a result, more always-on buffers might appear in the netlist.

Regardless of how the `mv_make_primary_supply_available_for_always_on` variable is set, the tool marks the selected nets based on the following rules:

- When the related supply net is in the same power domain as the net and it is not the primary power net of the power domain, the tool marks the net as `always_on`.
- When the related supply net is not in the same power domain as the net, the tool marks the net as `dont_touch`.
- When the related supply net is in the same power domain as the net, and it is the primary power net of the power domain, the tool inserts a regular buffer or inverter, and the net is not marked.

Always-On Optimization on Feedthrough Nets

To perform always-on optimization on top-level feedthrough nets, you must specify the related supply net information on the output port that is driven by the feedthrough net. The Power Compiler tool derives the power and ground net information for the always-on buffers based on the domain's primary supply and related supply net that you specify for the output port driven by the feedthrough net. If the tool detects a level-shifter violation or an isolation violation on a feedthrough net, it sets a `dont_touch` attribute on the feedthrough net. This is done to prevent the shifting of the violation from one power domain to another.

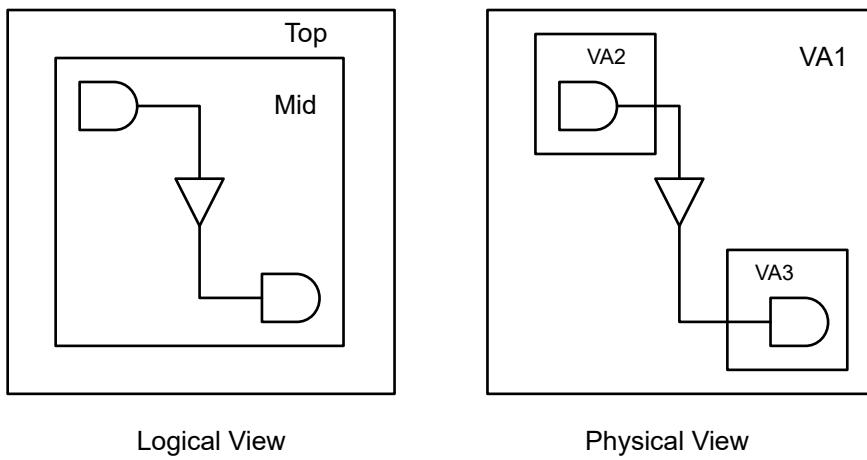
When running the Design Compiler tool in topographical mode, you can enable voltage-aware always-on synthesis on certain physical feedthrough paths. This enables buffer insertion in a physical feedthrough path when there is a disjoint voltage area even though, logically, the voltage areas belong to the same hierarchy. To enable this feature, set the `dct_enable_va_aware_ao_synthesis` variable to `true`.

Always-On Optimization on Disjoint Voltage Area

The Power Compiler tool can insert always-on buffers on long nets that span physically distant voltage areas. Consider a long net as shown in [Figure 70](#). Logically, the net and the buffer are in the same hierarchy Mid, which is an always-on domain. However, physically, the net and the buffer are in two disjoint voltage areas.

If the library supports dual rail always-on cells, and the primary supply defined in the power domain for subdesign Mid is available in the power domain for Top, the tool inserts dual rail always-on cells in the subdesign Mid that physically belongs to the Top design.

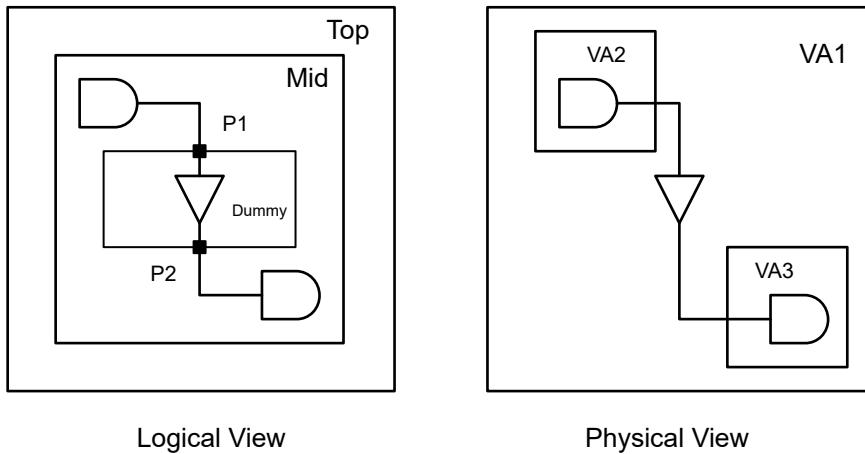
Figure 70 Always-On Buffer Insertion in Disjoint Voltage Areas



The tool follows these steps to support always-on synthesis across disjoint voltage areas:

1. Create a placeholder logic hierarchy inside the existing hierarchy Mid as shown in [Figure 71](#).
2. Create two hierarchical ports P1 and P2 on the placeholder hierarchy and connect the buffer inside the placeholder hierarchy to these ports.
3. Associate the placeholder hierarchy to the already existing voltage area, to which the buffer belongs.

Figure 71 Creating Placeholder Hierarchy to Support Always-On Buffer Insertion in Disjoint Voltage Areas



The creation of the placeholder logic hierarchy and port punching on the placeholder hierarchy allows the tool to perform always-on synthesis and legalization of always-on synthesis. The tool also supports associating the placeholder hierarchy to the default voltage area, if the buffer belongs to the default voltage area.

Always-On Tie Cells

The Power Compiler tool appropriately chooses a normal tie cell, an always-on tie cell, or a combination of a single-power tie cell and a power management cell (isolation, level-shifter, or enable level shifter) to implement each tie-high and tie-low logic value specified in the design. This results in a more efficient implementation of tie-high and tie-low logic values at power domain boundaries.

Basic Always-On Tie Cell Mapping

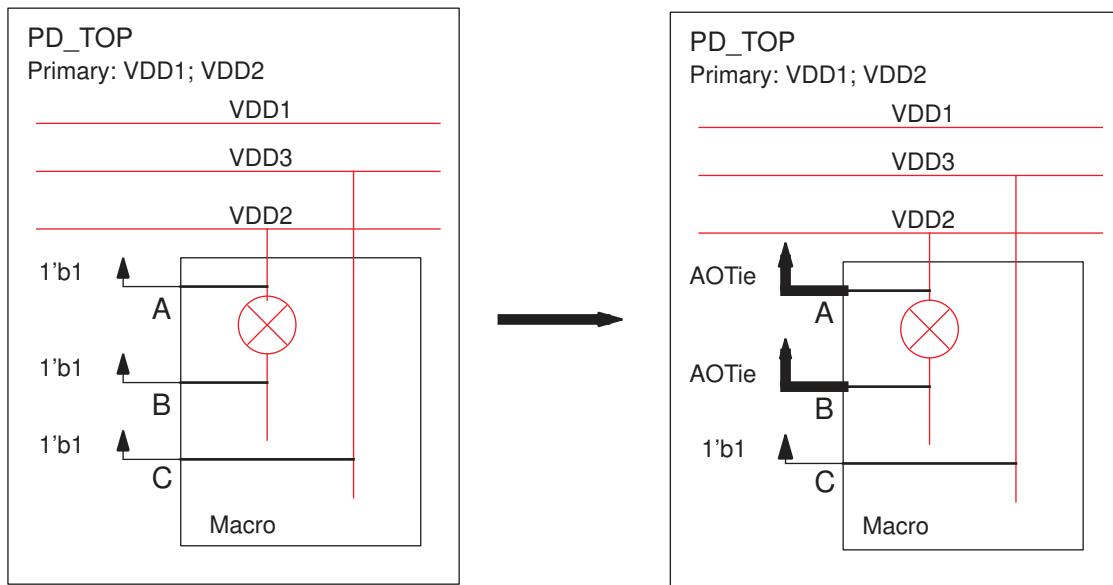
In general, all constants are assumed to be powered by the primary power supply of the domain where they reside. Therefore, each constant is mapped to a normal tie cell powered by its domain's primary power. However, if a constant is not driven by its primary supply during RTL simulation, it is mapped to an always-on (AO) tie cell. This enables the tool to insert tie cells on a constant path with logic using non-primary supplies.

However, in the case of constants driving macro cells that are powered by a supply different from its primary supply, these cells are assumed to be powered by the sink's related supply. If the sink's related supply is not available, for example, a macro cell's internal supply, an available supply that can drive the sink is selected. An AO or normal tie cell is used to map the constant powered by the selected supply.

In [Figure 72](#), the primary supply is VDD1, and VDD2 is an available supply. Pins A and B are related to an internal supply that is not available, so the tool finds another supply

to drive those pins. Pins A and B are mapped to AO tie cells powered by a non-primary supply, VDD2, and pin C is left as a literal constant because VDD3 is not available in the power domain PD_TOP and VDD1 cannot drive it.

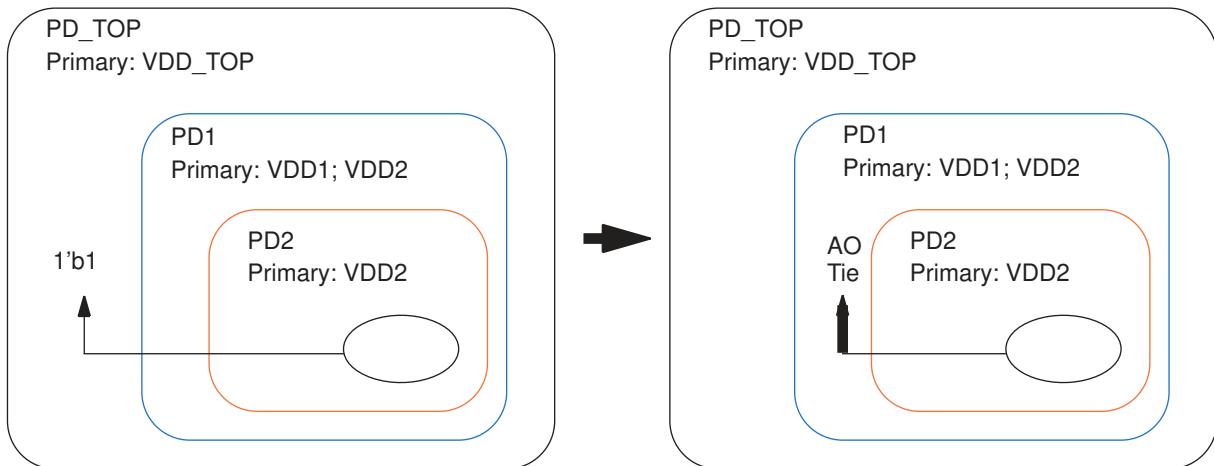
Figure 72 Macro Cells Using Non-Primary Supply



Enhanced Constant Propagation

Using an always-on (AO) tie cell for cell mapping increases the amount of instances that the constant can be propagated since these cells can be powered by a non-primary power supply. The example shown in [Figure 73](#) illustrates this point. The constant can be moved from PD_TOP to PD1 because VDD2 is an available supply. The constant cannot be moved to PD2 because boundary optimization and constant propagation are disabled on this domain. In the example, the VDD2 supply is more often on than VDD1, which is more often on than VDD_TOP.

Figure 73 Enhanced Constant Propagation



Enhanced Always-On Tie Cell Mapping

The enhanced always-on tie cell mapping maps a literal constant according to the following rules:

1. If there is no load, a normal tie cell is mapped
2. If the primary supply can drive the load, a normal tie cell is mapped
3. If a non-primary supply is available that can drive the load, an always-on tie cell is mapped
4. Otherwise, the constant is left as a literal constant

UPF Support for Custom Always-On Wrapper Cells

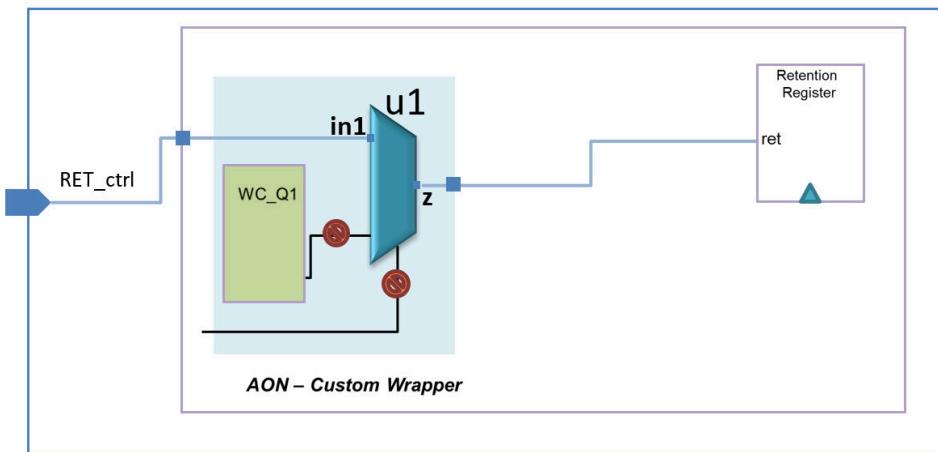
The Power Compiler Design-For-Test (DFT) feature namely, custom wrapper cells on input signals, is used for the insertion of a test logic that wraps the existing input signal. Such a

wrapper cell has a backup supply, to preserve the supply of the wrapped signal and thus its power characteristics. See the following figures:

Figure 74 Example RET_ctrl Always-on Control Signal



Figure 75 Example Wrapper Cell Wrapping RET_ctrl Control Signal, With Possible Isolation/LS Violations Marked



The Power Compiler UPF flow supports the insertion of custom always-on (AO) wrapper cells on input signals. The steps in the Power Compiler UPF flow for creating these custom AO multiplexer (MUX) cells are the following:

1. DFT calls the UPF before the wrapper cell insertion. The UPF tests the insertion and returns true or false depending on whether it allows the wrapper cell to be inserted in the given net segment
2. After the wrapper cell insertion, DFT calls the UPF again, to create the new UPF' statements for the newly-created wrapper cell. The Power Compiler UPF performs the following changes to the UPF for this new cell:
 - Add a `connect_supply_net` statement to the UPF to connect the backup power pin of the dual-rail MUX cell to the driver supply (or the local electrical equivalent) of the target net.
 - Add a `set_design_attribute` statement to the UPF to set the `leaf_cell_as_domain_boundary` attribute to `true` on the dual-rail MUX cell. This

statement adds a leaf cell to the extent of the lower domain boundary of the parent domain of the cell. (See [leaf_cell_as_domain_boundary Design Attribute](#).)

- Add a `set_port_attribute` statement in the UPF to set the `upf_control_signal_trace` attribute to `true` on the appropriate input and output pins of the dual-rail MUX cell. This statement tells the UPF tools how to trace a control signal through the wrapper cell. (See [upf_control_signal_trace Port Attribute](#).)

These statements appear as tool-derived UPF statements.

3. The tool prepares to add the isolation and level-shifter cells (as necessary) on the dual-rail MUX's inputs, when the `insert_mv_cells` command is invoked.

leaf_cell_as_domain_boundary Design Attribute

The `leaf_cell_as_domain_boundary` design attribute can be set to `true` on leaf cell instances.

```
dc_shell> set_design_attributes -elements {cell_list}
           -attribute leaf_cell_as_domain_boundary TRUE
```

When a wrapper cell is inserted, this attribute is automatically inferred by the tool. When the value of this attribute is set to `true` for a cell instance, the tool considers this cell as a part of the lower boundary of its parent power domain. This means that the isolation and level shifter strategies apply to the cell.

This behavior is the same as the behavior specified in the LRM for macro cells, which is currently enabled in Synopsys implementation tools with the `macro_as_domain_boundary` attribute.

Note:

If the `leaf_cell_as_domain_boundary` attribute is set to `true` on a cell that is not both always-on and a MUX leaf cell, the tool generates an error message. Do not set or use this attribute on any cell other than an always-on MUX leaf cell.

upf_control_signal_trace Port Attribute

The `upf_control_signal_trace` port attribute on signal ports, when set to `true`, ensures that the retention or isolation cells, driven by the signal which is wrapped by the always-on wrapper cell, continue to match their strategy after the cell is inserted.

```
dc_shell> set_port_attribute -ports {pin_list}
           -attribute upf_control_signal_trace TRUE
```

When tracing a UPF isolation or retention control signal for matching a strategy, the tool checks for this attribute on the driver pin of the control signal. If present, the tool treats the

cell as a feedthrough path and continues tracing the control signal starting at the input pin with the corresponding attribute.

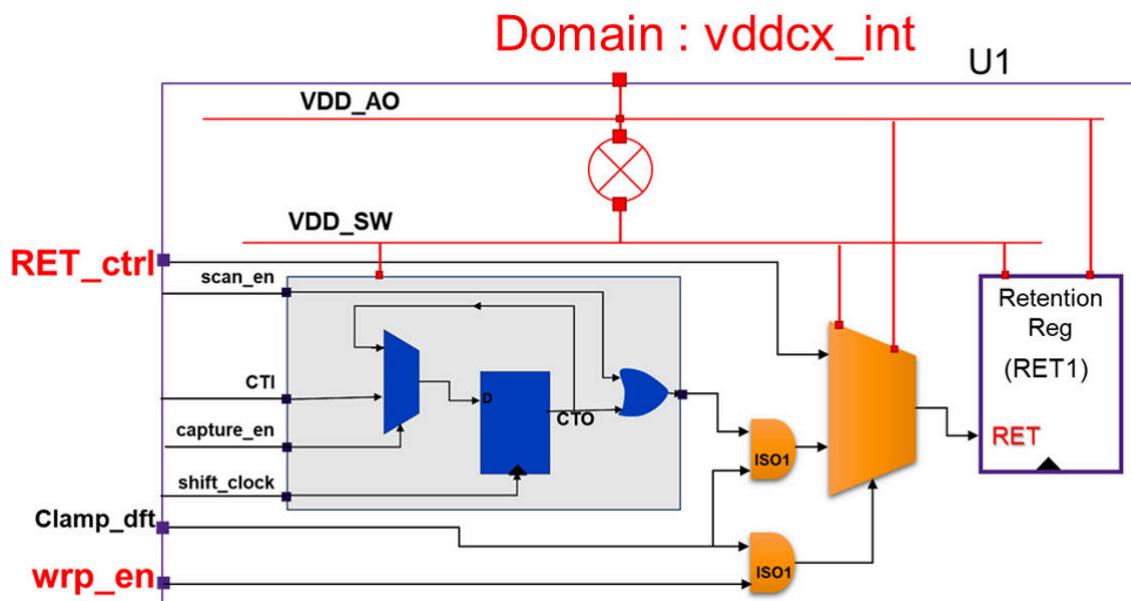
The value of the attribute is set to `true` on exactly one input pin and one output pin of an instance.

Note:

Do not set or use this attribute on any cell other than an always-on MUX leaf cell. The attribute can only be specified with the `-ports` option.

Example

The following figure shows a design with an always-on wrapper cell added:



Before the insertion of this wrapper cell, the UPF tests whether to insert a wrapper signal on the `RET_ctrl` net, bringing in the `wrp_en` DFT control signal. The UPF determines that the `VDD_AO` supply is available in the `vddcx_int` domain and that the cell is bias-compatible with the domain bias supplies. The UPF then allows the DFT to proceed with the cell insertion.

After insertion of the cell, the tool infers the following statements and adds them to the UPF:

```
dc_shell> connect_supply_net VDD_AO -ports {U1/mux_inst/TVDD}
dc_shell> set_design_attributes -elements {U1/mux_inst}
      -attribute leaf_cell_as_domain_boundary TRUE
dc_shell> set_port_attributes -ports {U1/mux_inst/in1 U1/mux_inst/z}
      -attribute upf_control_signal_trace TRUE
```

During isolation insertion, the U1/mux_inst cell is recognized as part of the lower boundary of the vddcx_int domain, due to the `leaf_cell_as_domain_boundary` design attribute. Therefore, isolation cells are inserted according to the domain strategy.

When re-reading the design, if the tool needs to associate the retention register with a defined strategy, then the UPF traces the signal. When the signal trace reaches the U1/mux_inst/z pin, the `upf_control_signal_trace` port attribute is identified. Signal trace continues at the corresponding U1/mux_inst/in1 pin, finally arriving at RET_ctrl.

Fixing Multivoltage Violations

Design changes sometimes result in isolation or voltage violations. The `fix_mv_design -buffer` command can identify and fix violations for buffer trees. The `-buffer` option is required.

The fixes made by this command are performed automatically as part of the `compile_ultra` command. Therefore, the command has no effect on a compiled design. However, if you make manual netlist changes or load a third-party netlist, executing the `fix_mv_design -buffer` command might fix some issues.

The `fix_mv_design -buffer` command can perform the following changes:

- Fix buffers and inverters that have illegal settings, as follows:
 - Process, voltage, and temperature settings
 - Library cell purpose or subset settings
 - Target library subset settings
 - Bias voltages
- Swap buffer library cells to fix illegal settings
- Swap single-rail buffers for dual-rail buffers
- Swap dual-rail buffers for single-rail buffers

The `fix_mv_design` command keeps the placement of the original buffers or inverters and honors the repeater strategies associated with existing buffers and inverters. In addition, the command only uses supplies that are available in the voltage area or voltage area region.

The `fix_mv_design` command does not insert isolation cells or level-shifter cells to fix existing isolation or voltage violations. Netlist changes caused by the command do not introduce new isolation or voltage violations.

To specify a buffer tree, use the `-from` option with the name of a net or driver pin. If you specify a net name, its driver pin is used. In both cases, the driver pin defines a full or partial buffer tree. Wildcards are supported.

To provide a list of single-rail and dual-rail buffer or inverter library cells that the `fix_mv_design` command can use as replacement cells, user the `-lib_cells` option.

Comparing Voltage Levels and Voltage Status

Before you define your isolation and level-shifter strategies, you might want to compare the voltage status and voltage levels of two supplies. To compare two supplies, use the `compare_supplies` command. This command compares the first supply (designated as the *reference supply*) to a second supply (designated as the *specified supply*).

The `compare_supplies` command provides the following options:

- `-on_status`

Compares the specified supply to the reference supply and returns one of the following: `equal`, `more_on`, `less_on`, or `independent`.

- `-voltage_level`

Compares the specified supply to the reference supply and returns one of the following: `equal`, `higher`, `lower`, or `independent`.

When comparing the voltage levels of two supplies that belong to the same correlated supply group, the following rules apply:

1. If the two supplies' minimum, nominal, and maximum voltage values are equal, the command returns `equal`.
2. If the specified supply's minimum, nominal, and maximum voltage values are less than the reference supply's min, nom, and max voltage values, the command returns `lower`.
3. If the specified supply's minimum nominal, and maximum voltage values are greater than the reference supply's minimum, nominal, and maximum voltage values, the command returns `higher`.
4. If none of the preceding rules is true, then the command returns `independent`.

If you compare the voltage levels of two supplies that are not part of the same correlated supply group, the following rules apply:

1. If the two supplies' minimum, nominal, and maximum values are equal, the command returns `equal`.
2. If the reference supply's minimum value is greater than the specified supply's maximum value and rule 1 does not apply, the command returns `lower`.

3. If the reference supply's maximum value is less than the specified supply's minimum value and rule 1 does not apply, the command returns higher.
4. If none of the preceding rules is true, then the command returns independent.

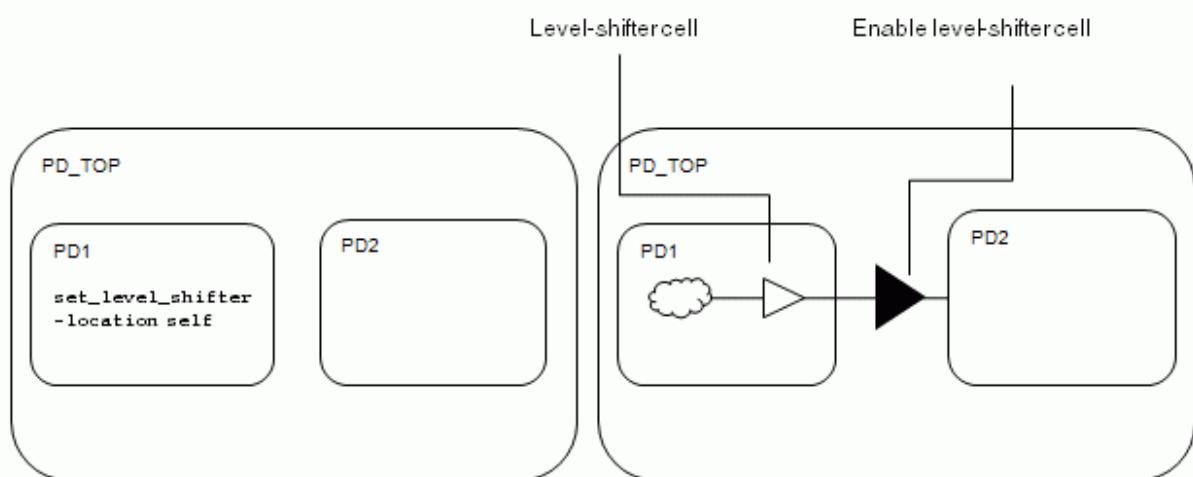
Specifying Level-Shifter Strategies

Use the `set_level_shifter` command to specify the strategy for inserting level-shifter cells between power domains that operate at different voltages. Level shifters are inserted by the tool during execution of the `compile_ultra` command.

If a voltage violation exists at the domain boundary, the tool inserts level shifters at the domain boundary by default, even when a level-shifter strategy is not defined. This flexibility allows the tool to use the strategy that gets the best possible results. You can optionally restrict the insertion of level shifters to domain boundaries where the `set_level_shifter` command is used by setting the `upf_levshi_on_constraint_only` variable to `true`.

Specifying a strategy does not force a level-shifter cell to be inserted unconditionally. The Power Compiler tool uses the power state table and the specified rules, such as threshold, to determine where level shifters are needed. When the tool identifies a potential voltage violation, it tries to resolve the violation by inserting multiple level-shifters or a combination of level-shifter and isolation cells. As shown in [Figure 76](#), when the tool finds a global net that has an isolation constraint, it inserts a level-shifter and an enable level-shifter cell, based on the voltage difference implied by the isolation power and isolation ground. The tool issues a warning message if it determines that a level shifter is not required.

Figure 76 Level-Shifter Insertion on Power Domain Boundaries



The `-input_supply` and `-output_supply` options specify the power and ground supply connections for inserted level-shifter cells based on the driver and load supplies in the path and the available supplies in the domain. The default supply is used when the input or output supply is not specified. If the specified driver and load supplies and an equivalent supply are not available, the tool does not insert any level-shifter cells.

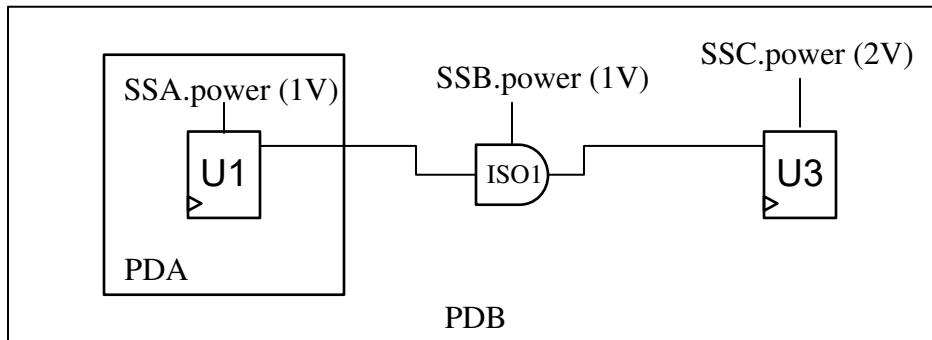
A voltage violation might occur when the specified input and output supplies do not match the driver and load supplies. To fix this violation, the tool automatically inserts level-shifter cells in other places along the path. However, you can use the `-force_shift` option to insert level-shifter cells regardless of any voltage violations.

Use the `-elements` option to specify a list of ports and pins in the domain to which the strategy applies, overriding any `-threshold` or `-rule` settings. The `-no_shift` option prevents the insertion of level-shifter cells on the ports, pins, and nets specified by the `-elements` option. Use the `-exclude_elements` option to exclude elements from the strategy.

Use the `-name_prefix` or `-name_suffix` option of the `set_level_shifter` command to specify the naming of the level-shifter cell instances added during the implementation of a specific level-shifter strategy.

The `-source` and `-sink` options of the `set_level_shifter` command allow you to restrict which paths a level-shifter strategy applies. [Figure 77](#) is an example of source and sink analysis when applying level-shifter strategies.

Figure 77 Example of Source and Sink Analysis With an Isolation Cell on Path



Consider the following strategies:

```
set_isolation isol1 -domain PDA -isolation_supply SSB -location parent
set_level_shifter LS1 -domain PDA -source SSA -sink SSC -location parent
```

For the path from U1 to U3, the source supply is SS1.power and the sink supply is SSC.power. The level-shifter strategy applied is LS1 and the level shifter is inserted on the path from ISO1 to U3 because there is no level shifter violation on the path from U1 to ISO1.

When you use the `-force_shift` with the `-source` and `-sink` options, the level-shifter strategies are applied to the path segment from the source to the sink. For the example shown in [Figure 77](#), the following level-shifter strategies are defined:

```
set_level_shifter LS1 -domain PDA -source SSA -sink SSC -no_shift
set_level_shifter LS2 -domain PDA -threshold 0.0 -location parent
```

For the path from U1 to U3, the source supply is `SSA.power` and the sink supply is `SSC.power`. No level shifter is inserted because the `-no_shift` option is specified.

Use the `-update` option to add information to a level-shifter strategy. You must use either the `-elements` or `-exclude_elements` option with the `-update` option.

Controlling Level-Shifter Locations

To apply a level-shifter strategy to a specific power domain boundary, use the `-applies_to_boundary` option.

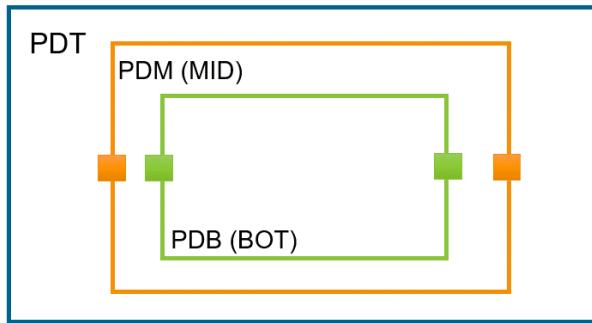
To control the location of level-shifter cells, use the `-location` option of the `set_level_shifter` command, as follows:

- The `-location automatic` setting (the default) allows the tool to choose the location.
- The `-location self` setting specifies that the level-shifter cell is placed inside the model or cell being shifted.
- The `-location parent` setting specifies that the level-shifter cell is placed in the parent of the model or cell being shifted.
- The `-location other` setting specifies that the level-shifter cell is placed in the parent domain for an upper boundary port or in the child domain for a lower boundary port. The upper boundary port cannot be a port of the design top module and the lower boundary port cannot be a pin of a leaf cell or a hard macro.

If you use the `-location` option, you must ensure that the necessary supplies are available in the specified location. You cannot use the `-location` option with the `-update` option.

For example, consider the power domains in [Figure 78](#).

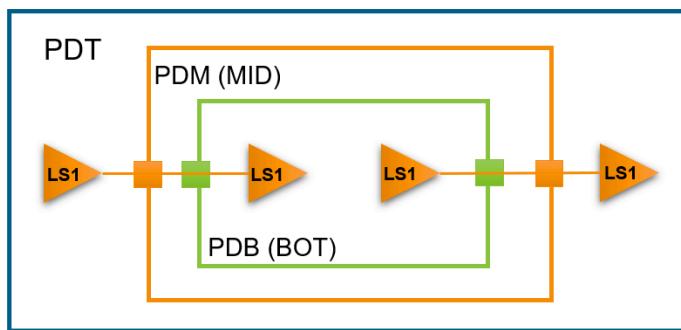
Figure 78 Nested Power Domains



The following UPF commands cause the tool to insert the level-shifter cells shown in [Figure 79](#). Strategy LS1 applies to both the upper and lower boundaries of power domain PDM, for both input and output ports. The tool inserts all level-shifter cells outside the PDM domain because the specified location is `other`. The tool places the level-shifter cells for the upper boundary ports of domain PDM in the top-level domain and the level-shifter cells for the lower boundary ports of domain PDM in the bottom-level domain.

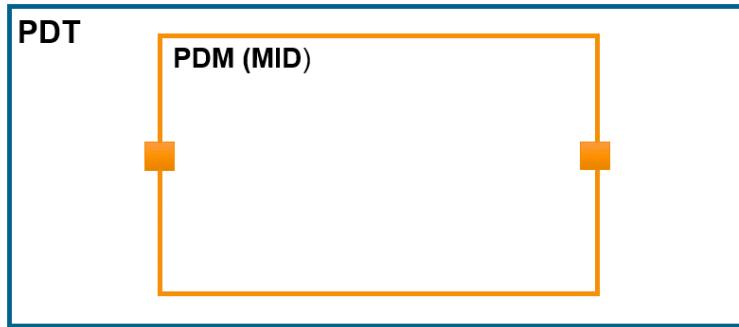
```
create_power_domain PDT
create_power_domain PDM -elements {MID}
create_power_domain PDB -elements {MID/BOT}
set_level_shifter LS1 -domain PDM -applies_to_boundary both \
    -applies_to both -location other
```

Figure 79 Nested Power Domains With Level Shifters



Now consider the simple power domains in [Figure 80](#).

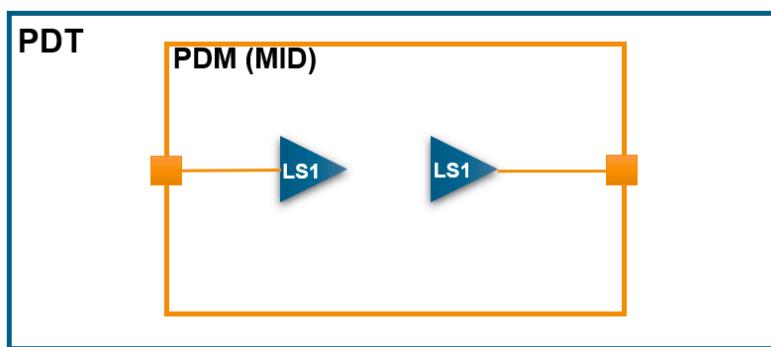
Figure 80 Nested Power Domains



The following UPF commands cause the tool to insert the level-shifter cells shown in [Figure 81](#). Strategy LS1 applies to domain PDT and causes the tool to insert level-shifter cells at the lower boundary of domain PDT, but the location of those cells is inside the PDM domain because the specified location is `other`. Strategy LS2 applies to domain PDM and specifies no level-shifter cell insertion for ports of domain PDM. Therefore, the tool does not insert any level-shifter cells for strategy LS2, but this strategy does not prevent implementation of the level-shifter strategy for domain PDT.

```
create_power_domain PDT
create_power_domain PDM -elements {MID}
set_level_shifter LS1 -domain PDT -applies_to_boundary lower \
    -applies_to both -location other
set_level_shifter LS2 -domain PDM -applies_to both -no_shift
```

Figure 81 Nested Power Domains With Overlapping Strategies



Resolving Level Shifter Strategy Precedence

The following level shifter strategies have decreasing order of precedence, regardless of the order in which they are executed:

1. Strategies applicable to ports, instances, and domains that specify the `-elements` option. Ports have a higher precedence than instances, which have a higher precedence than domains.
2. Strategies using the `-no_shift` option
3. Strategies using the `-sink` or `-source` options

The `-applies_to` option has no effect on precedence resolution.

In the following example, the LS2 strategy has higher precedence because it uses the `-elements` option and applies to a port. The LS1 strategy uses the `-sink` option, which has a lower priority.

```
set_level_shifter LS1 -domain PD1 -sink PD2.primary
set_level_shifter LS2 -domain PD1 -elements block1/p1[0]
```

In the following example, the LS2 strategy has higher precedence because it uses the `-sink` option.

```
set_level_shifter LS1 -domain PD -applies_to outputs
set_level_shifter LS2 -domain PD -sink PD2.primary
```

In the following example, both strategies have the same precedence. The tool issues an error message because it cannot resolve the precedence.

```
set_level_shifter LS1 -domain PD -elements inst1 -applies_to inputs
set_level_shifter LS2 -domain PD -elements inst1
```

Automatically Deriving Level Shifter Strategies for DFT Paths

While specifying the UPF level shifter strategies for DFT paths, the `-source` and `-sink` level shifter strategies are specified in the UPF to target the DFT paths generated during the `insert_dft` command. The issue with this approach is that the `-source` and `-sink` level shifter strategies targeting DFT paths can also be applied to functional paths. This can result in redundant level shifter cells on functional paths.

Also, with the mixed use of both `-element` and `-source` and `-sink`-based level shifter strategies in the UPF, for the same path, it is possible to introduce redundant level shifter cells.

To resolve this issue, that is, to avoid applying the level shifter strategy targeting DFT paths on functional paths, you can enable the tool to apply the `-source` and `-sink` level shifter strategies only on DFT paths.

To automatically derive level shifter strategies for DFT paths, perform the following steps:

1. Specify a placeholder `-source` and `-sink` level shifter strategy in the RTL UPF to target DFT paths, that is, strategy with an empty element:

```
set_level_shifter ls_dft -domain PD_BOT -source SS_TOP
-sink SS_BOT -elements {} ...
```

2. Post the `insert_dft` command, use the `generate_mv_constraints -dft_level_shifter` command to auto update the placeholder level shifter strategies in a power domain with DFT paths.
3. Apply the updated level shifter strategies and run the `insert_mv_cells` command to insert relevant level shifter cells.

The complete syntax for `generate_mv_constraints -dft_level_shifter` is the following:

```
generate_mv_constraints -dft_level_shifter -apply -output filename
```

Where:

- `-dft_level_shifter`: Updates source and sink level shifter strategies with DFT crossings
- `-apply`: Automatically applies the updated level shifter strategies
- `-output filename`: Saves the updated level shifter strategies in the specified file

During `generate_mv_constraints -dft_level_shifter`, the tool considers the following pins and ports on a domain crossing to update the source and sink level shifter strategies:

- Newly punched hierarchical pins during `insert_dft` (You can also query these newly punched DFT pins using the `get_dft_hierarchical_pins` command.)
- Any ports or pins on a domain crossing tagged with the `created_during_dft_eco` predefined attribute
- All ports specified in the `set_dft_signal` command

If an existing level shifter strategy is already applicable for a DFT port or pin, the tool skips adding this port or pin while updating the source and sink level shifter strategies.

Using Specific Library Cells With the Level-Shifter Strategy

When you specify a level-shifter strategy, by default the tool maps the level-shifter cells to a suitable level-shifter cell in the library. Use the `use_interface_cell` command to specify the set of library cells to be used for a level-shifter strategy. This command does not force the insertion of the level-shifter cells. Instead, when the tool inserts the level-

shifter cell, it chooses the library cells that are specified with the `-lib_cells` argument of the `use_interface_cell` command. This command has no effect on instantiated level-shifter cells that have a `dont_touch` attribute set on them.

Allowing Insertion of Level-Shifters on Clock Nets and Ideal Nets

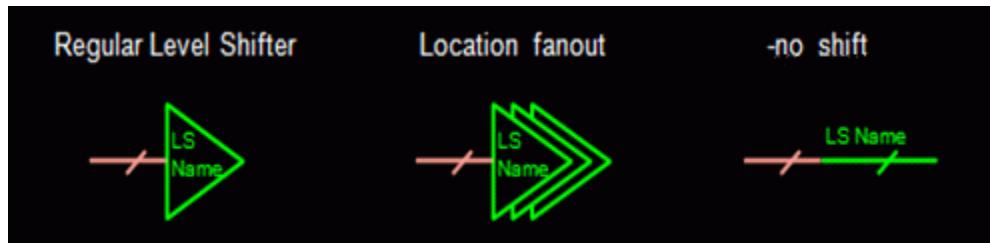
The Power Compiler tool does not insert level-shifter cells on clock nets, by default. Set the `auto_insert_level_shifters_on_clocks` variable to specific clock nets, for the tool to insert the level-shifter cells. Set this variable to `all`, for the tool to insert level-shifter cells on all clock nets that need level shifters.

Similarly, the tool does not insert level-shifter cells on ideal nets, by default. Set the `mv_insert_level_shifters_on_ideal_nets` variable to `all` for the tool to insert level-shifters on ideal nets. The default is an empty string ("").

Representing Level-Shifter Strategies in the UPF Diagram View

In the UPF diagram view, the level-shifter symbol is similar to a buffer symbol and includes a line segment representing the inputs that are shifted, as shown in [Figure 82](#). The location-fanout symbol looks similar to several buffers bundled together and indicates that the level-shifter cells occur on all fanout locations (sink) of the port that they are shifting. The no-shift symbol is represented by a line that shows the continuation of the inputs.

Figure 82 Representation of Level-Shifter Cells in the UPF Diagram



The symbol for each level-shifter strategy is located adjacent to the boundary of its parent power domain. The location depends on whether it shifts inputs or outputs.

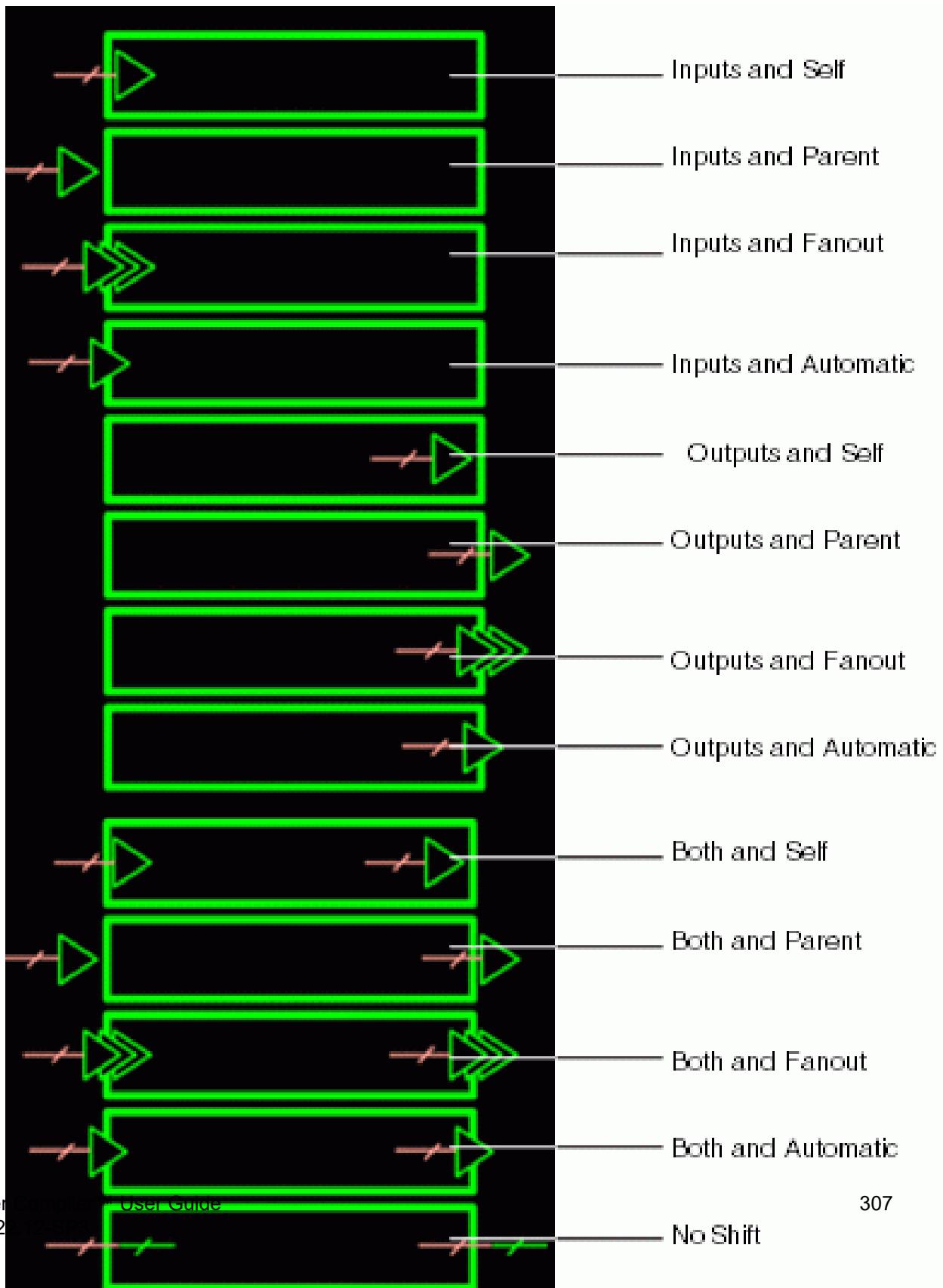
[Figure 83](#) shows the possible combinations of level-shifter symbols and locations, based on the values specified with the `-applies_to` and `-location` options of the `set_level_shifter` command.

The symbol appears at the left edge of the boundary if the strategy applies to input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports. If the strategy applies to both inputs and outputs, symbols appear at both left and right edges of the boundary.

If you specify the location in the level-shifter strategy as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

When you specify a list of elements to the level-shifter strategy using the `set_level_shifter -elements -applies_to` command, the UPF diagram positions the symbol relative to the left or right edge of the power domain boundary, based on whether the list contains input elements, output elements, or both.

Figure 83 Representation of Various Level-Shifter Strategies in the UPF Diagram



Specifying Isolation Strategies

Use the `set_isolation` command to define the isolation strategy for a power domain and the elements in the power domain where the strategy is applied. The definition of an isolation strategy contains specification of the enable signal net, the clamp value, and the design elements where the strategy is applied.

The isolation power and ground nets must operate at the same voltage as the primary power and ground nets of the power domain where the isolation cells is located.

When you specify only the `-isolation_power_net` argument, the primary ground net is used as the isolation ground supply. Similarly, when you specify only the `-isolation_ground_net` argument, the primary supply net is used as the isolation power supply.

The `-isolation_supply` option specifies the power and ground functions of the same supply set to be used as the isolation power and isolation ground nets respectively. The Power Compiler tool can infer a regular isolation cell when the `-isolation_supply {}` option is specified. Normally, this option infers a NOR-type isolation cell, but if that is not possible, a regular isolation cell is inferred. When using a regular isolation cell, the tool uses the primary supply of the domain as the isolation cell's power supply if the primary supply is more on or equally on than the isolation control signal supply. If the supply is not on at least as much as the isolation control signal supply, then the isolation cell is not mapped.

The `-isolation_supply` option is mutually exclusive with the `-isolation_power_net` and the `-isolation_ground_net` options.

The `-source` option filters the ports connected to a net that is driven by the specified supply set. When you use this option, the supply sets that are associated with each other are considered as connected.

The `-sink` option filters the ports driving a net that fans out to the logic driven by the specified supply set. Supply sets that are associated with each other are considered as connected.

When you specify both the `-source` and `-sink` options, isolation is applied to only those ports that have the specified source and sink.

The `-diff_supply_only` option determines the isolation behavior between the driver and the receiver supply sets or supply nets.

When the `-diff_supply_only` option is set to `true`, and the same supply set connects the driver and the receiver of a port on the interface of the reference power domain, the isolation cell is not added in the path from the driver to the receiver. Also in this case, the default for the `-applies_to` option is `both`.

Note:

With the `-diff_supply_only` option, you can use only one of the `-source` and `-sink` options.

The `-clamp_value` specifies the constant value in the isolation output. Valid values are `0`, `1`, and `latch`. The `-clamp_value` option does not support the value `z`. The latch setting causes the value of the non-isolated port to be latched when the isolation signal becomes active.

The `-elements` option specifies the elements for isolation in cases where there are multiple isolation strategies within a given power domain. The specified elements can be input or output ports on the domain boundary and design instances. The design instances must be the root cells of the power domain. The tool applies the isolation strategy only on domain boundaries and ignores the leaf cell instances.

You can specify an empty argument list for the `-elements` option. If you specify subsequent `set_isolation` commands with the `-update` option, the end result is a combination of the element sets of all `set_isolation` commands. For example, the following is a valid command:

```
dc_shell> set_isolation -domain pd1 -elements {}
```

Later, you can add to the elements list using the `-update` option as follows:

```
dc_shell> set_isolation -domain pd1 -elements e1 -update
```

When you specify the wildcard characters (`*` or `?`) with the `-elements` option, the tool searches for matching ports, pins, or design instances in the current level of hierarchy and applies the isolation strategy to the elements identified as the boundaries of the specified power domain. To restrict the application of the isolation strategy on design instances, set the `upf_isols_allow_instances_in_elements` variable to `false`.

The tool filters the design elements, such as ports, pins, and design instances, when you specify the `-applies_to` option with the `-elements` option. To control the filtering behavior, set the `upf_iso_filter_elements_with_applies_to` variable. The valid values are `ERROR`, `ENABLE`, and `DISABLE`.

- `ERROR`

Generates an error message when you specify the `-applies_to` option with the `-elements` option.

- `ENABLE` (the default)

Filters the elements (pins, ports, and design instances) based on the value that you specify with the `-applies_to` option.

- DISABLE

Ignores the `-applies_to` option and applies the isolation strategy to all the elements specified with the `-elements` option. For the design instance specified with the `-elements` option, the isolation strategy is applied to all the pins of the specified instance.

When you do not specify any of the `-elements`, `-source`, and `-sink` options, the isolation strategy is applied to all the output ports of the power domain.

The `-no_isolation` option specifies that the elements in the `-elements` list should not be isolated. At least one of the `-isolation_power_net` or `-isolation_ground_net` or `-isolation_supply` arguments must be specified unless `-no_isolation` option is used.

Although the power state table can potentially reduce the number of isolation cells required, isolation synthesis is entirely based on directives set using the `set_isolation` and `set_isolation_control` commands.

The tool performs certain optimizations on isolation circuits, that do not affect the functionality. For example, if you have signals going from block A to block B, you specify output isolation on block A (in the parent) and input isolation on block B (in the parent). If the strategy results in two back-to-back isolation cells with no fan out in between, the tool merges the isolation cells. It can merge the isolation cells based on the enable signal, power, or ground signals.

Use the `-name_prefix` or `-name_suffix` options of the `set_isolation` command to specify the naming of the isolation cell instances added during the implementation of a specific isolation strategy.

Use the `-update` option to add information to an existing isolation strategy. You can always use this option with either the `-elements` or `-exclude_elements` option. In addition, you can use the `-update` option with the `-location` option in specific situations. For example, you might initially define a strategy without a location, then update the strategy to add a location.

The following requirements apply to the `set_isolation -location -update` command:

- The `set_isolation -location -update` command must occur before the `insert_mv_cells` or `compile` commands.
- You must use the `set_isolation -location -update` command before the `set_isolation_control` command.
- The isolation strategy must have an isolation control signal. However, you cannot use the `set_isolation -location -update` command after the `set_isolation_control` command. Therefore, you must specify the isolation control signal with the `set_isolation` command.

- You cannot use the `set_isolation -location -update` command if the strategy already has a location specified by an earlier `set_isolation -location` command.
- You can use the `set_isolation -location -update` command only one time.

Isolation Control Signals

The `set_isolation` and `set_isolation_control` commands both have an `-isolation_sense` option, which specifies the logic state of the isolation control signal that places isolation cells in the isolation mode. The possible values for this option are `high` or `low`. The default is `high`.

The isolation signal specified by the `-isolation_signal` option, which is available in both commands, refers to a port, pin, or net with the port or pin having higher precedence. The isolation signal can exist outside the logic hierarchy where the isolation cells are inserted; the synthesis or implementation tools can perform port-punching to make the connection. These punched ports are not considered for isolation or level shifting, even though after port creation, they reside within the coverage of an isolation or level-shifter strategy.

You might have an isolation cell that does not have an enable pin and works as a latch when the input side supply shuts off. To specify this type of cell in the UPF, use the `-isolation_signal {}` option with either the `set_isolation` or `set_isolation_control` command. The UPF should also explicitly indicate whether the cell is a single-rail or dual-rail cell for simulation modeling. You cannot specify the `-isolation_signal {}` option with the `-isolation_sense` option.

Isolation Cells With Multiple Control Signals

The Power Compiler tool supports isolation cells with multiple control signals. However, their usage is limited to within a hard macro.

Use the `set_isolation` command, which follows the IEEE 1801-2018 standard, to specify the multi-control isolation strategy. The command allows a list of values for the `-isolation_signal`, `-isolation_sense`, `-clamp_value`, and `-isolation_supply` options, to enable you to write a strategy for the multi-control isolation cell.

See the following example for the Liberty file modeling the multi-control isolation and the associated UPF strategy:

```
#Liberty model

pin (clock) {
    direction : input;
    is_isolated : true;
    isolation_enable_condition : "!ctrl1||!ctrl2|ctrl3";
}

pin (ctrl1) {
    direction : input;
```

```

        is_isolated : true;
        isolation_enable_condition : "ctrl3";
    }

pin (ctrl12) {
    direction : input;
    is_isolated : true;
    isolation_enable_condition : "ctrl3";
}

pin (ctrl13) {
    direction : input;
    related_power_pin : vddy;
    related_ground_pin : vssy;
}

#UPF multi-control isolation strategy

set_isolation mem_iso -domain PD_MEMORY
    -isolation_signal {ctrl11 ctrl12 ctrl13}
    -isolation_sense {low low high}
    -clamp_value 1
    -location self

```

As per the IEEE 1801-2018 standard on specifying multi-control isolation strategy:

The `-isolation_signal`, `-isolation_sense`, `-clamp_value`, and `-isolation_supply` options are each specified as a single value or a list. If any of these options specify a list, then all the lists specified for these options should be of the same length, and any single value specified is treated as a list of values of the same length. The tuples formed by associating the positional entries from each list are used to define separate isolation requirements for the strategy. These tuples are applied to the isolation cell from the isolation cell's data input port to its data output port in the order in which they appear in each list. The output of the isolation cell is the right-most value in the `-clamp_value` list whose corresponding isolation signal is active.

Note:

- The multi-control isolation strategy is allowed only on hard macro UPF.
 - For RTL verification, the macro instance should be modeled with the `is_hard_macro` or `UPF_is_hard_macro` UPF attribute.
 - For design implementation and gate-level verification, you can use the `is_macro_cell:true` Liberty attribute to identify the cell as a hard macro in the absence of the previously mentioned UPF attribute.
- The `-location` of the corresponding isolation strategy should not be of a value parent, or any other value that can cause this isolation cell to be outside the macro.

- The length of the `-isolation_signal`, `-isolation_sense`, `-clamp_value`, and `-isolation_supply` option values should follow the IEEE 1801-2018 recommendation. That is, if any of these options specify a list, then the value specified by the other options should be of the same length or a single value. If a single value is specified, it is treated as a list of values of the same length as the list specified for another option. See the following example:

```
set_isolation isol1_macro -domain PD_SRAM
  -isolation_signal {ctrl1 ctrl2 ctr3}
  -isolation_sense {high low high}
  -clamp 0
  -isolation_supply {SS1 SS2 SS3}
```

- For any mismatch between the Liberty specified `isolation_enable_condition` expression and the UPF isolation strategy values, the tool overrides the Liberty specification with the UPF isolation strategy definition.

Using the `set_isolation_control` Command

The `set_isolation_control` command specifies the isolation control signal and isolation sense separately. The command identifies an existing isolation strategy and specifies the isolation control signal for that strategy.

Using the location value you specify with the `-location` option of the `set_isolation_control` command, the Power Compiler tool identifies isolation cells in the power domain across the design hierarchy and associates them with UPF strategies. When the value is `self`, the tool starts the search from the port on the boundary of the power domain and traverses inside the power domain until it encounters either a cell, a multiple fanout net, or the boundary of another power domain.

When the location is `parent`, the tool starts the search from the port on the boundary of the power domain, and traverses outside the power domain until it encounters a cell, a multiple fanout net, or the boundary of another power domain. The `-location parent` option also supports heterogeneous fanouts by using strategies defined by the `-sink` and `-diff_supply_only` options.

When the tool encounters an isolation cell that is not already associated with an isolation strategy, it associates the cell with an appropriate isolation strategy. This association is based on the values you specified with the `-clamp_value` option of the `set_isolation` command, and the `-isolation_sense` option of the `set_isolation_control` command. If the cell encountered is not an isolation cell, the tool does not treat the port as an isolation port, and during the next optimization step, the tool inserts an isolation cell.

The `-isolation_sense` option specifies the logic state of the isolation control signal that places isolation cells in the isolation mode. The possible values for this option are 0 or

1. The default is 1. The isolation signal specified by the `-isolation_signal` option can be for a port or pin or a net, with the port/pin having higher precedence. The isolation signal need not exist in the logic hierarchy where the isolation cells are to be inserted; the synthesis or implementation tool can perform port-punching as needed to make the connection. Port-punching means automatically creating a port to make a connection from one hierarchical level to the next. These punched ports are not considered for isolation or level-shifting, even though after the port creation, these ports reside within the coverage of an isolation or level-shifter strategy.

Existing ports are isolated and level-shifted according to the applicable isolation and level-shifter strategy, even if they reside on an always-on path.

Rules for the Location Fanout Option

The `-location fanout` option of the `set_isolation` and `set_isolation_control` commands will be deprecated in a future release. Use the `-location parent` option to support heterogeneous fanout isolation in combination with strategies defined by the `-sink` and `-diff_supply_only` options.

The following rules apply to the `-location fanout` option:

- Do not use the `-isolation_power_net` option of the `set_isolation` command when you use the `-location fanout` option. However, when you use the `-location` option with the value `parent` or `self`, you can use the `-isolation_power_net` option of the `set_isolation` command.
- The `-location fanout` option can be used only when you use one of the `-source`, `-sink`, or `-diff_supply_only` options of the `set_isolation` command. Similarly, when you use the `-elements` option and one of `-source`, `-sink`, or `-diff_supply_only` options, you can only specify `fanout` with the `-location` option.
- The `-no_isolation` option cannot be used when you use the `-elements` option and one of `-source`, `-sink`, or `-diff_supply_only` options of the `set_isolation` command.
- Set the `derived_iso_strategy` attribute using the `set_design_attributes` command before specifying the `-elements` option and one of the `-source`, `-sink`, or `-diff_supply_only` options of the `set_isolation` command. After setting the `derived_iso_strategy` option, if you do not specify the `-elements` option and one of the `-source`, `-sink`, or `-diff_supply_only` options, the tool issues an error message.
- If you set the `derived_iso_strategy` attribute, the only value that you can specify with the `-location` option is `fanout`.

For more details about setting the UPF attributes on ports and hierarchical cells, see [Setting UPF Attributes on Ports and Hierarchical Cells](#)

Order of Precedence of Isolation Strategies

The isolation strategies have the following decreasing order of precedence, irrespective of the order of execution:

1. Strategies that apply to ports, explicitly specified using the `-elements` option.
2. Strategies that apply to ports, implied by specifying an instance using the `-elements` option.
3. Strategies that apply to ports, implied by specifying only the power domain name.
4. Strategies using the `-no_isolation` option.
5. Strategies using the `-source` or `-sink` option. If both options are used, the strategy takes precedence over any strategy where only one option is used.
6. Strategies where the `-diff_supply_only` option is set to `true`.
7. Strategies where the `-diff_supply_only` option is set to `false`.

Resolving Isolation Strategy Conflicts

The Power Compiler tool uses the isolation precedence rules to resolve conflicts when more than one isolation strategy can apply to a port. The tool detects conflicts only after completely reading the UPF files. In general, if a conflict is detected for a port, only the first strategy applies to the port, and the port is removed from the effective element list of subsequent strategies.

Automatically Deriving Isolation Strategies for DFT Paths

While specifying UPF isolation strategies for DFT paths, `-source` and `-sink` ISO strategies are specified in the UPF to target the DFT paths generated during `insert_dft`. The issue with this approach is that the `-source` and `-sink` ISO strategies targeting DFT paths can also be applied to functional paths. This can result in redundant isolation cells on functional paths.

Also, with the mixed use of both `-element` and `-source` and `sink` based isolation strategies in the UPF, for the same path, it is possible to introduce redundant isolation cells.

To resolve this issue, that is, to avoid applying the ISO strategy targeting DFT paths on functional paths, you can enable the tool to apply the `-source` and `-sink` ISO strategies only on DFT paths.

To automatically derive ISO strategies for DFT paths, performing the following steps:

1. Specify a placeholder `-source` and `-sink` ISO strategy in the RTL UPF to target DFT paths, that is, strategy with an empty element:

```
set_isolation iso_dft -domain PD_BOT -source SS_TOP -sink SS_BOT
-elements {} ..
```

2. Post `insert_dft`, use the `generate_mv_constraints -dft_isolation` command to auto update the placeholder ISO strategies in a power domain with DFT paths.

3. Apply the updated isolation strategies and run `insert_mv_cells` to insert relevant isolation cells.

The complete syntax for `generate_mv_constraints -dft_isolation` is the following:

```
generate_mv_constraints -dft_isolation -apply -output filename
```

Where:

- `-dft_isolation`: updates source/sink ISO strategies with DFT crossings
- `-apply`: automatically applies the updated isolation strategies
- `-output filename`: saves the updated isolation strategies in the specified file

During `generate_mv_constraints -dft_isolation`, the tool considers following pins/ports on a domain crossing to update the source/sink ISO strategies:

- Newly punched hierarchical pins during `insert_dft`; you can also query these newly punched DFT pins using the `get_dft_hierarchical_pins` command.
- Any ports or pins on a domain crossing tagged with the predefined attribute `created_during_dft_eco`.
- All ports specified in the `set_dft_signal` command

If an existing ISO strategy is already applicable for a DFT port/pin, the tool skips adding this port/pin while updating the source/sink ISO strategies.

Using Specific Library Cells With Isolation Strategies

When you define an isolation strategy, by default the tool associates the isolation strategy with any suitable isolation cell in the library. When the library does not contain a complete set of isolation cells, you can use some of the basic gates as isolation cells. For more information, see [Multivoltage Design Concepts](#).

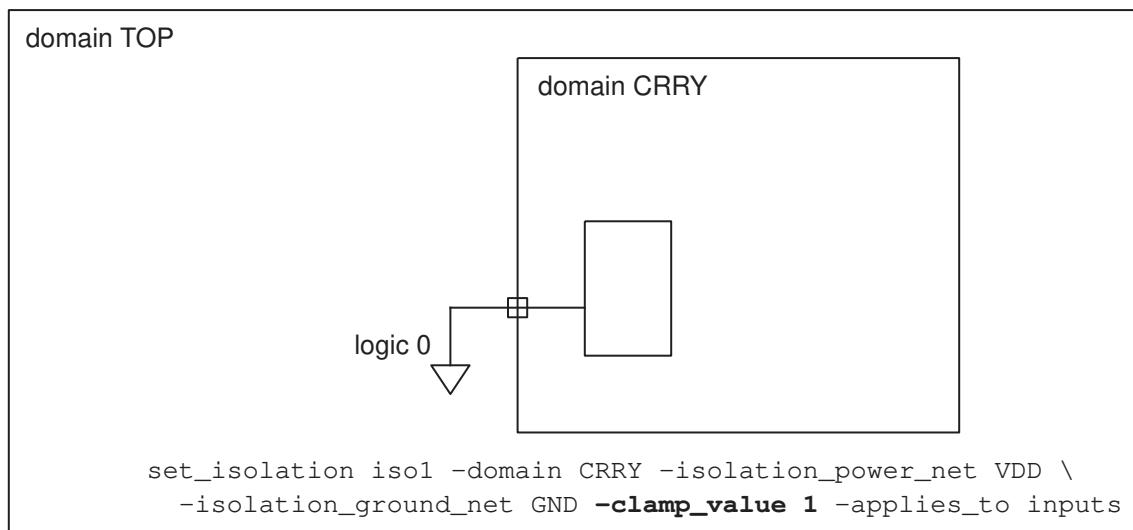
To associate a specific set of library cells with the isolation strategy, use the `use_interface_cell` command. The `use_interface_cell` command can also be used to associate standard cells used as isolation cells with the isolation strategy.

When designs contain instantiated isolation cells that are associated with an isolation strategy, the `use_interface_cell` command remaps these library cells to the cells specified with the `-lib_cells` argument of the command. If the instantiated isolation cells have `dont_touch` attribute set on them, the command does not remap these cells. The command has no impact on the instantiated isolation cells that are not, or cannot be associated with an isolation strategy.

Aligning Isolation Strategies to Constant Drivers

Consider a situation where a driver forces a port to a constant value, either a logic 0 or 1, at a power domain boundary, but the isolation clamp value defined for that port in the UPF file is the opposite value, as shown in the example in [Figure 84](#).

Figure 84 Constant Driver and Conflicting Isolation Strategy



This situation can arise during logic optimization when Power Compiler moves or splits a constant value that crosses domain boundaries. The tool might need to insert an isolation cell to prevent a formal verification error. In such a situation, you might want to modify the isolation strategy to match the constant value so that the isolation cell can be removed during optimization.

To automatically generate new UPF isolation commands that are consistent with the constant driver values, you can use the `generate_mv_constraints` command, as shown in the following example. Suppose that the original UPF commands define an isolation

strategy named iso1, which applies to the inputs of the CRRY power domain and sets a clamp value of 1 as follows:

```
dc_shell> set_isolation iso1 -domain CRRY -isolation_power_net VDD \
           -isolation_ground_net GND -clamp_value 1 -applies_to inputs
dc_shell> set_isolation_control iso1 -domain CRRY \
           -isolation_signal ctrl -isolation_sense low -location self
```

However, the design has a constant driver element driving the u1/cin pin with a value of 0 at the boundary of CRRY power domain, which is a conflict with the isolation strategy. To generate a new isolation strategy to match the constant driver, use this command:

```
dc_shell> generate_mv_constraints -align_isolation_clamp_value \
           -output align.upf
```

The command detects the conflict between the isolation strategy and the constant driver and generates the following new isolation strategy commands to resolve the conflict:

```
# Created by DC Utility for non-matching isolation clamp and driving
# constant value.
# List of new strategies created with clamp value matching the constant
# isolating it, for strategies with no user specified element list
set_isolation iso1_clamp0 -domain CRRY \
               -isolation_power_net VDD -isolation_ground_net GND \
               -elements u1/cin -clamp_value 0 -applies_to inputs
set_isolation_control iso1_clamp0 -domain CRRY \
               -isolation_signal ctrl -isolation_sense low -location self
```

The `generate_mv_constraints` command writes comments and the two new isolation strategy commands to the `align.upf` file. It creates the name of the new isolation strategy by appending the `_clamp0` suffix to the original strategy name. The new `set_isolation` command always uses the `-elements` option to identify the pins to which the strategy applies. The new strategy is more specific than the original strategy, so it has higher priority.

When using the `generate_mv_constraints` command, specify the `-output` option to write the commands into a file, the `-apply` option to execute the new commands, or both options to perform both actions. If you use the `-apply` option, the newly created and modified strategies are applied to the design in memory, and any subsequent usage of the `save_upf` command writes the new isolation commands along with the original UPF commands.

By default, the `generate_mv_constraints -align_isolation_clamp_value` command checks only for conflicts involving general `set_isolation` strategies specified without the `-elements` option, for example, using `-applies_to inputs`. To also fix conflicts with isolation strategies specified with the `-elements` option of the `set_isolation` command, use the `-include_elements` option of the `generate_mv_constraints` command.

For example, suppose that the original UPF commands define an isolation strategy named iso2, which applies to the u1/cin input and u1/carry input pins of CRRZ power domain and sets a clamp value of 1 for these pins, as follows:

```
dc_shell> set_isolation iso2 -domain CRRZ -isolation_power_net VDD \
           -isolation_ground_net GND -elements {u1/cin u1/carry} \
           -clamp_value 1 -applies_to inputs
dc_shell> set_isolation_control iso2 -domain CRRZ \
           -isolation_signal ctrl -isolation_sense low -location self
```

However, the design has a constant driver element driving the u1/cin pin with a value of 0 at the boundary of the CRRZ power domain, which is a conflict with the isolation strategy. To fix the element-level strategy for conflicts with constant drivers, use the generate_mv_constraints command with the -include_elements option as follows:

```
dc_shell> generate_mv_constraints -align_isolation_clamp_value \
           -include_elements -output align.upf -apply
```

The command detects the element-level conflict and generates the following new strategies to resolve the conflict:

```
# Created by DC Utility for non-matching isolation clamp and driving
# constant value.
# List of user strategies modified
set_isolation iso2_modified -domain CRRZ -isolation_power_net VDD \
  -isolation_ground_net GND -elements u1/carry -clamp_value 1 \
  -applies_to inputs
set_isolation_control iso2_modified -domain CRRZ -isolation_signal ctrl \
  -isolation_sense low -location self

# List of new strategies created with clamp value matching the constant
# isolating it, for strategies with user specified element list
set_isolation iso2_clamp0 -domain CRRZ -isolation_power_net VDD \
  -isolation_ground_net GND -elements u1/cin -clamp_value 0 \
  -applies_to inputs
set_isolation_control iso2_clamp0 -domain CRRZ -isolation_signal ctrl \
  -isolation_sense low -location self
```

The generate_mv_constraints command generates two new strategies: one named iso2_modified for the input without a conflict and another one named iso2_clamp0 for the input that has a conflict. When you use the -apply option, the new strategies are applied to the design in memory and the original isolation strategies are updated to remove the elements that are listed in the new isolation strategy; any subsequent usage of the save_upf command writes out the new isolation commands.

If you use the generate_mv_constraints command, you must do so before you compile the design. If the netlist already contains isolation cells, using the generate_mv_constraints command might result in back-to-back isolation cells or loss of association between existing isolation cells and the isolation strategy.

Optimizing Isolation Cell Insertion on Constants

You can prevent the Power Compiler tool from inserting isolation cells on constants (literals, such as 1'b0 and 1'b1) that drive macro cell input and primary output ports when the clamp value of the isolation strategy matches the logic value of the constant. For these cases, the tool propagates the constant across the domain boundary and an isolation cell is not needed.

To prevent isolation cell insertion, set the

`relax_constant_corruption_for_macro_inputs_and_primary_outputs` `design` attribute to `true`, as follows:

```
dc_shell> set_design_attributes -elements {} \
    relax_constant_corruption_for_macro_inputs_and_primary_outputs true
```

If the clamp value does not match the value of the constant, the tool inserts an isolation cell if a strategy is defined.

Preventing Unnecessary Isolation Cell Insertion

You can prevent unnecessary isolation cell insertion by using the `generate_mv_constraints` command. Use this command after loading the design and the UPF files.

When you use the `generate_mv_constraints` command, the tool generates an isolation strategy with the `-no_isolation` option specified. These strategies are generated for power domain boundary pins where an isolation strategy has been specified, but is not needed based on the power states defined in the UPF files. This command has no effect on domain boundary pins where isolation cells have already been inserted. When you use the `-no_isolation` option, you must specify the `-elements` option.

You must specify at least either the `-output` or `-apply` option along with the `-no_isolation` option. If you do not specify the `-apply` option, you must load the output UPF file manually. For example,

```
prompt> generate_mv_constraints -no_isolation -output no_iso.upf
```

The output file `no_iso.upf` contains the following:

```
# Tool-derived commands
set_isolation snps_no_iso_0 -domain PD1 -no_isolation -elements ...
```

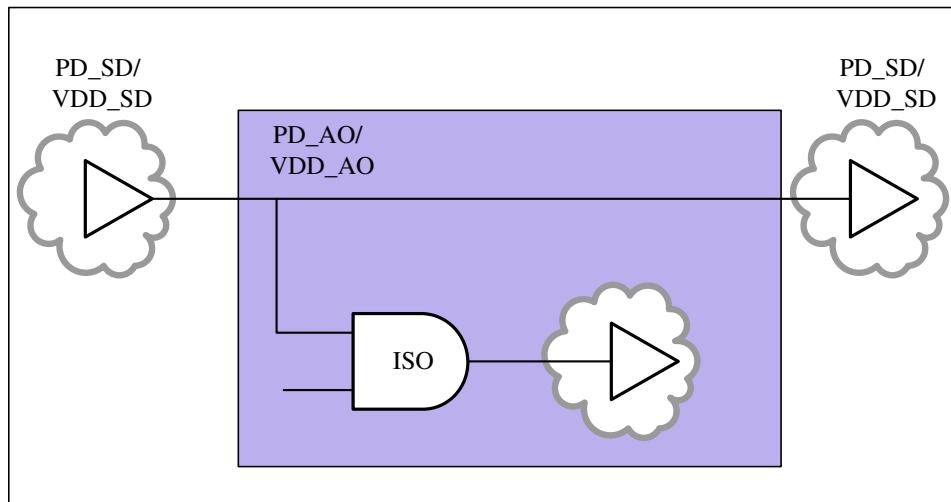
Isolation Cells and Heterogeneous Loads

The `-sink` and `-diff_supply_only` options for isolation cell insertion allow you to specify an isolation strategy based on the supplies used for the driver and the load of a path. You can also use these options for paths with loads that have heterogeneous supplies.

The `-location` option specifies where the isolation cell is placed. To use a heterogeneous fanout isolation strategy with the `-sink` or `-diff_supply_only` option, the `-location` option must be set to `parent` or `self`.

For example, in [Figure 85](#), the isolation cell is inserted on a load with `VDD_AO` supply and on a load with `VDD_SD` supply.

Figure 85 Isolation Cell With Heterogeneous Loads



For this example, the isolation strategy might be defined as follows:

```
set_isolation S1 -domain PD_AO -sink VDD_AO \
-diff_supply_only -location self
```

The tool supports isolation cell insertion with name-based associations and can optimize the source or sink logic.

To report paths that have heterogeneous fanout and their related supplies, use the `report_heterogeneous_fanout` command. The report indicates whether a heterogeneous fanout topology is acceptable for UPF implementation.

For this analysis, the tool does not consider the following to be heterogeneous:

- Equivalent nets, which have both functional and electrical equivalence
- Equivalent supplies, defined as follows:
 - Supply nets connected in the UPF through supply ports or with the `associate_supply_set` command
 - Supply nets defined with the `set_equivalent` or `set_equivalent_function_only` command

By default, the tool analyzes the entire design. You can restrict the scope of the report by using the `-nets` option, in which case the tool analyzes the driver pin of each net in the argument list. Alternatively, you can use the `-pins` option to provide a list of primary ports, leaf cell pins, or hierarchical crossings. The tool treats an output pin as a driver pin. The tool treats an input pin as a load, in which case the tool analyzes its driver pin.

The Power Compiler tool ignores inout pins, elements with multiple drivers, and elements with no load or no driver. The tool issues a warning message only if it encounters one of these objects when the `-nets` or `-pins` option is specified.

If the supply of one fanout load is heterogeneous, the tool reports all of the loads of the same driver. An example of the default report is as follows:

```
dc_shell> report_heterogeneous_fanout

RSN - Related Supply Net
* Indicates heterogeneous fanout found in core of a power domain

Number of drivers with heterogeneous fanout: 2
-----
Driver Pin      Driver Pin RSN      Load Pin      Load Pin RSN
-----          -----          -----
img4/C23/z      VDD_MID          img_aux/C1/A   VDD_AO
                  img_aux/C2/A   VDD_AUX

*img4/U10/z      VDD_MID          img4/C4/A     VDD_AUX
                  img4/U12/A   VDD_MID
                  img_aux/C3/A   VDD_AUX
```

An example of a pin report is as follows:

```
dc_shell> report_heterogeneous_fanout -pins img_core/test_si

RSN - Related Supply Net
* Indicates heterogeneous fanout found in core of a power domain

Number of drivers with heterogeneous fanout: 1
-----
Driver Pin      Driver Pin RSN      Load Pin      Load Pin RSN
-----          -----          -----
test_si         VDD_TOP          img_core/p_fifo/sin  VDD_AUX
                  img_core/o_reg/sin  VDD_MID
```

You can use the `-verbose` option of the `report_heterogeneous_fanout` command to generate a complete report with the following additional information about heterogeneous fanouts in your design:

- Driver and load cell power domain names
- Always-on violation and voltage mismatch (LS violation) on the path between the driver and its load, for the reported topology

- Names of hierarchical pins that are domain crossings between the driver and its load
- Supplies related to the loads in the fanout that are power-state-table (PST) equivalent to the driver's related supply net

Note:

This additional information is shown only for the driver and loads that belong to a heterogeneous fanout topology.

The following two examples show a comparison of the tool reports for the same design, without and with the `-verbose` option:

```
dc_shell> report_heterogeneous_fanout
```

RSN – Related Supply Net

* - Indicates heterogenous fanout found in core of a power domain

Number of drivers with heterogenous fanout: 2

Driver Pin	Driver Pin RSN	Load Pin	Load Pin RSN
u_img_core/C23/Z	VDD_MID	u_img_aux/C1/A u_img_aux/C2/A	VDD_AO VDD_AUX
*u_img_core/U10/z	VDD_MID	u_img_core/C4/A u_img_core/U12/A u_img_aux/C3/A	VDD_AUX VDD_MID VDD_AUX

```
dc_shell> report_heterogeneous_fanout -verbose
```

RSN – Related Supply Net

* - Indicates heterogenous fanout found in core of a power domain

Number of drivers with heterogenous fanout: 2

Driver Pin	Driver PD	Driver Pin RSN	Violations	Load Pin	Load PD	Load Pin RSN	Domain Crossing
u_img_core/C23/Z	MID_PD	VDD_MID	AO Voltage	u_img_aux/C1/A u_img_aux/C2/A	BOT_PD BOT_PD	VDD_AO VDD_AUX	u_img_core/out1 u_img_core/out1
*u_img_core/U10/z	MID_PD	VDD_MID	Voltage AO None	u_img_core/C4/A u_img_core/U12/A u_img_aux/C3/A	MID_PD MID_PD BOT_PD	VDD_AUX VDD_AO VDD_BOT	None None u_img_core/out2
test_si	TOP_PD	VDD_TOP	None Voltage	u_img_core/pdx_fifo/sin u_img_core/o_reg/sin	TOP_PD TOP_PD	VDD_AUX VDD_MID	u_img_core/in1 u_img_core/in1

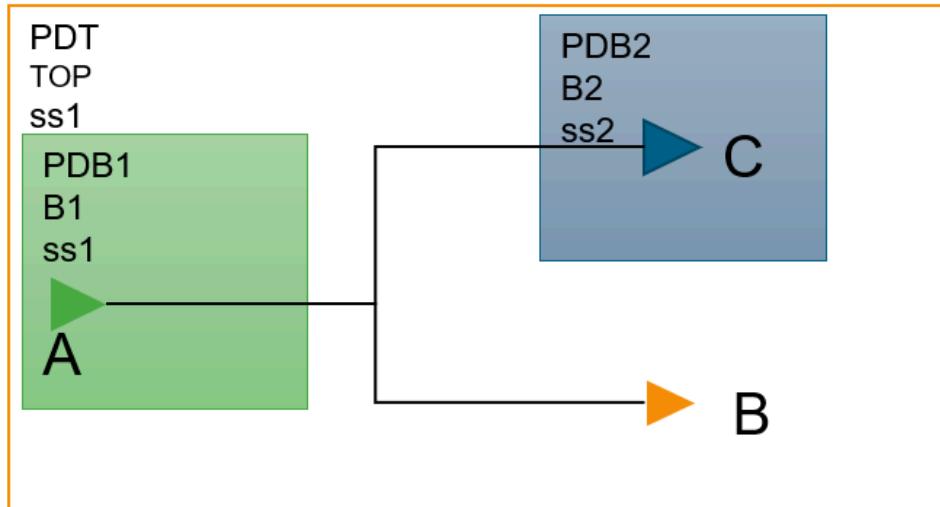
Equivalent Supplies: {VDD_MID VDD_BOT} {VSS_MID VSS_BOT VSS_TOP}

PST-wise equivalent: {VDD_TOP VDD_AUX}

Insertion of Isolation Cells on Heterogeneous Fanout Paths

Heterogeneous fanout is a cross domain net which drives multiple fanouts, where the sink supplies are different. See the following illustration:

Figure 86 Isolation Cell With Heterogeneous Loads



The Power Compiler tool supports isolation cell insertion on heterogeneous fanout paths, by default. So, for path-based isolation cell insertion on heterogeneous fanout paths, you need not explicitly set the `hetero_fanout_isolation` design attribute to `true`.

However, during the `insert_mv_cells` or the `compile_ultra` command execution, if the UPF contains path-based (`-sink` or `-diff_supply_only`) isolation strategies and the `hetero_fanout_isolation` design attribute is not set to `true`, the tool generates an UPF-498 warning message. To avoid this warning message and to ensure that the isolation cell insertion is consistent across all Synopsys verification and implementation tools, it is recommended that you set the `hetero_fanout_isolation` attribute value to `true`.

Note:

The tool can honor a value `true` for
`conservative_diff_supply_only_isolation` only if the
`hetero_fanout_isolation` attribute is explicitly set to `false`. If not, the tool
generates an UPF-846 warning message.

Isolation Handling on Control Signals

Sometimes a port is referenced in a UPF command as the control signal of a UPF strategy or as part of a logic strategy. If the port itself has an isolation strategy, the tool handles isolation cell insertion using certain rules.

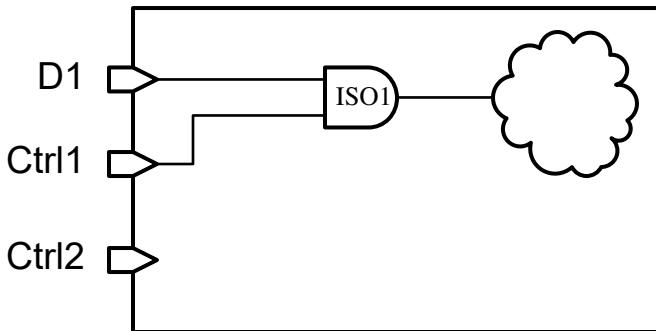
Elements that are not specified as the isolation signal of any isolation strategy are processed first. Then, the tool processes elements specified as the isolation signal of an isolation strategy that is already implemented.

For example, suppose you have the following isolation strategies:

```
set_isolation ISO1 -elements {D1} -isolation_signal Ctrl1
set_isolation ISO2 -elements {Ctrl1} -isolation_signal Ctrl2
set_isolation ISO3 -elements {Ctrl2} -isolation_signal ...
```

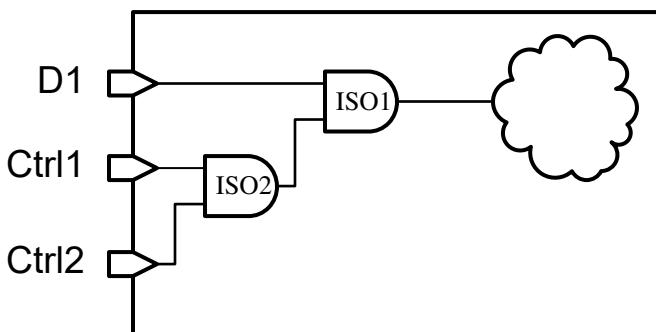
The strategy ISO1 has higher priority since its isolation element is not a control signal for any isolation strategy as shown in [Figure 87](#).

Figure 87 First Step in Isolation Cell Insertion



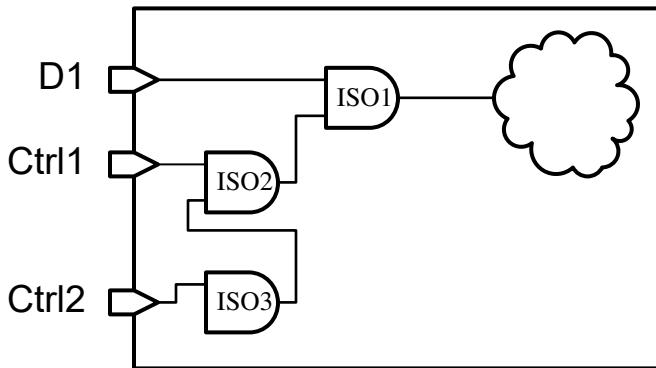
Isolation strategy ISO2 has the next higher priority since its isolation element, Ctrl1, is already implemented as the control signal of ISO1. This is shown in [Figure 88](#).

Figure 88 Second Step in Isolation Cell Insertion



Isolation strategy, ISO3, can be implemented since the isolation element, Ctrl2, is already implemented as the control signal of ISO. This is shown in [Figure 89](#).

Figure 89 Third Step in Isolation Cell Insertion



Smart Derivation of -no_isolation Strategy

Currently, the tool always automatically derives `-no_isolation` strategy on newly-punched hierarchical pins for:

- Retention control (save and restore)
- Power switch control
- Isolation control

This is done to ensure that no existing isolation strategy is applied to newly-punched ports.

However, this approach has the disadvantage that, if the driver of the control pin is less always-on than the control pin, the isolation violation cannot be fixed with the current isolation strategies, because `-no_isolation` has the highest precedence. There is no way to isolate the newly punched ports on the control path, to add new isolation strategy for them, to fix the isolation violation on the control path.

To resolve this issue, you can set the public variable

`upf_smart_derive_iso_strategy_on_new_control_ports` to true, for the tool to intelligently derive the `-no_isolation` strategy for new punched control pins, based on the PST status. See the following syntax:

```
set upf_smart_derive_iso_strategy_on_new_control_ports true
```

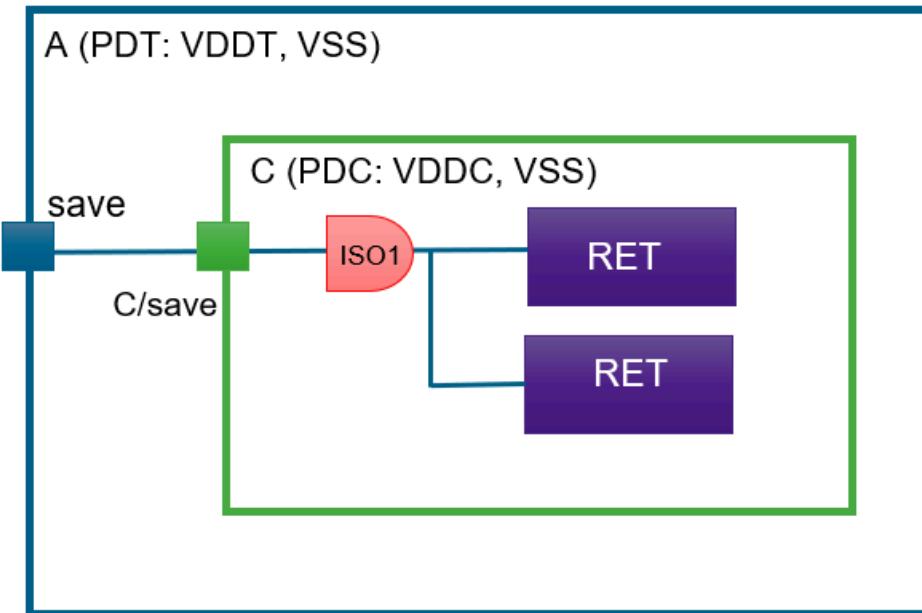
In the smart approach, tool will first check whether the control path has an isolation violation. Only if it finds that there is no isolation violation, the `-no_isolation` strategy will be derived for the new ports.

Note:

This feature is currently supported only for new punched pins on the retention control path.

Example

Consider the following example:



PST:

VDDT		VDDC
Pst1: vdd_on		vddc_on
Pst2: vdd_off		vddc_on
Pst3: vdd_on		vddc_off

```
set_retention RET_PDC -domain PDC -elements {C/ret1 C/ret2 ...}
    -retention_supply PDC.primary \
    -save_signal {save high} -restore_signal {restore high} ...

set_isolation ISO1 -domain PDC -isolation_supply PDC.primary
    -isolation_signal isol1 \
    -isolation_sense low -location self -clamp_value 0 -applies_to inputs
    -diff_supply_only TRUE
```

After `compile`, the tool punches new port for save signal C/save. There is ISO violation on the new port C/save because of Pst2 status. So, with `upf_smart_derive_iso_strategy_on_new_control_ports` to true, the tool will not derive `-no_isolation` strategy on pin C/save, will apply ISO1 strategy, and insert isolation cell on the path of C/save.

Macro Cells With Internal NOR Isolation Cells

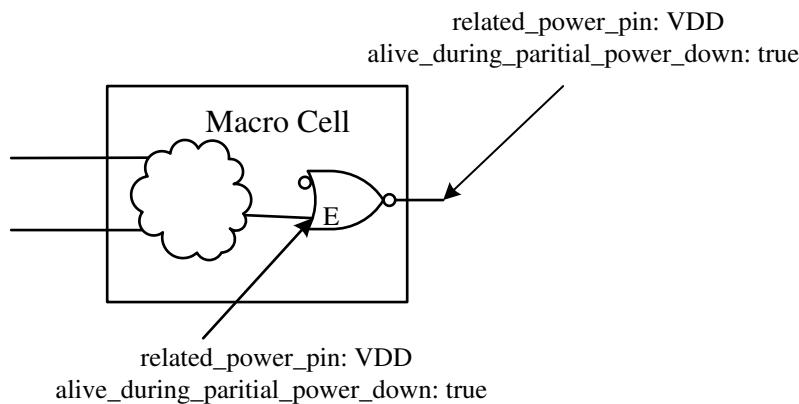
When you model an internally isolated macro cell, a NOR-style isolation cell might appear at the inputs or outputs of the macro cell. If a macro cell contains NOR isolation cells at its output, the Power Compiler tool behaves as if the macro has an `is_isolated : true` Liberty attribute specified at its output pin. The tool does not report isolation violations between the NOR-isolated macro output and the logic at the macro's load (assuming the load logic is powered on more than the macro). For details on macro cell modeling, see *The Library Compiler User Guide*.

The tool supports the following macro models when a NOR isolation cell is at the output of a macro:

- The enable pin of the NOR cell is a single pin and has the same `related_power_pin` attribute as the output pin.

For example, a macro with a standard NOR isolation cell at the output is shown in [Figure 90](#) with the specified Liberty attributes.

Figure 90 Standard NOR Cell and Related Macro Cell Attributes



When you use this model, the tool treats the macro cell as if the output pin had an `is_isolated : true` Liberty attribute set. The tool also assumes the following:

- The enable pin and output pin have the same `related_power_pin` attribute value
- The enable pin and output pin have the `alive_during_partial_power_down` attribute set to `true`
- The output pin must have the `isolated_enable_condition` attribute set to a single primary input
- The `isolation_enable_condition` attribute of the macro cell is an equation of all primary inputs to the macro. The tool assumes the output pin has the `alive_during_partial_power_down` attribute set to `true`.

- The isolation enable condition is missing, but the `alive_during_partial_power_down` attribute is set to `true`. The tool assumes the output is connected to an ideal always-on supply and there are no checks are performed on the output pin.
- If the `alive_during_power_up` attribute is set to `true` at the macro input pin, the tool performs electrical checks at the macro input pin. If this attribute is set to `false`, the checks are not performed.

Voltage Checking

To control the voltage checks that the tool performs, use the following variables:

- `upf_nor_iso_macro_allow_enable_supply_check`

By default, this variable is `true`. If set to `false`, the tool issues a warning message if all the enable pins' supplies are on less than the sink supply.

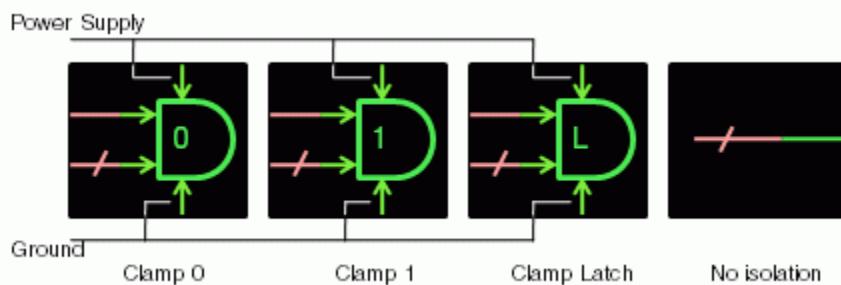
- `upf_allow_is_isolated_output_check`

By default, this variable is `true`. If set to `false`, the tool does not perform voltage checking for isolated output pins with respect to the load supply.

Representing Isolation Strategies in the UPF Diagram View

Figure 91 shows the symbols used to represent an isolation strategy in the UPF diagram view. The symbol used is similar to an AND gate and the clamp value is shown inside the symbol. The symbol also includes pins for power and ground, a segment representing the isolation signal, and a line segment representing the inputs or outputs that the strategy isolates. When the `-no_isolation` option is specified, a straight line is used to show the continuation of the inputs.

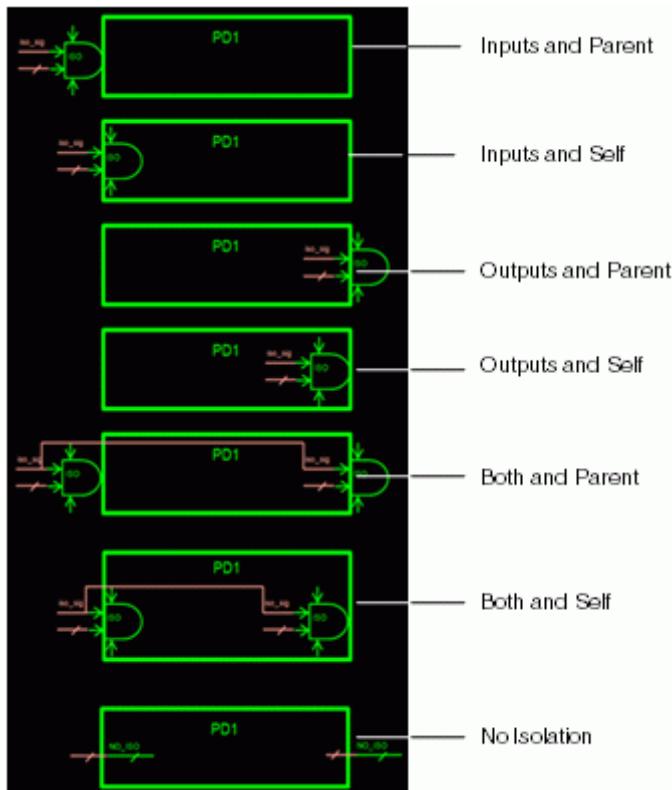
Figure 91 Symbols for Isolation Cells in the UPF Diagram



The symbol is located adjacent to the boundary of its parent power domain. The location also depends on whether the strategy isolates inputs or outputs.

[Figure 92](#) shows all possible combinations of isolation strategy symbols and locations, based on the value of the `-applies_to` option of the `set_isolation` command and the value of the `-location` option of the `set_isolation_control` command used in defining isolation strategy.

Figure 92 Representation of Isolation Strategies in the UPF Diagram



The symbol appears to the left edge of the power domain boundary if the strategy applies to the input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports.

If the strategy applies to both input and output ports, the symbol appears at both left and right edges of the boundary.

While defining the isolation strategy, if you specify the location as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

Note:

If you specify a list of elements using the `set_isolation -elements` command, the UPF diagram ignores the `-applies_to` option and positions the

isolation symbol relative to the left or right edge of the power domain boundary, based on whether the list contains input elements or output elements or both.

Merging and Cloning Multivoltage Cells

Based on the power strategies specified in the input UPF, the Power Compiler tool inserts multivoltage cells, namely, isolation cells, level shifters, and enable level shifters. After multivoltage cell insertion, the tool can perform logic restructuring, that is, merging and cloning of all the isolation, level shifter, and enable level shifter cells, to improve timing and area.

This logic restructuring step is disabled by default. To enable this step, set the `upf_enable_mv_merge_clone` variable to `true`, before running the `compile_ultra` command:

```
dc_shell> set upf_enable_mv_merge_clone true
```

The default value of this variable is `false`.

The tool merges the multivoltage cells if they satisfy all the following conditions:

- The cells have a common local driver.
- The cells logically belong to the same hierarchy.
- The cells are all of the same type, that is, the cells to be merged are all isolation cells, or all level shifter cells, or all enable level shifter cells.
- The cells belong to same strategy and power domain (for example, isolation or enable level shifter).

The Power Compiler tool performs merge and clone during the `compile_ultra` stage. Cloning is also supported when running `compile_ultra` in the incremental mode using the `-incremental` option.

Naming rules apply to the newly merged or cloned multivoltage cells. A keyword is added to the original names of these multivoltage cells that undergo merge or clone, to indicate the actions that are performed. The keyword is either `merge`, or `clone`, or `merge_clone` depending on what actions are performed.

When merge or clone happens, the tool prints informative messages at the end of the corresponding compilation stage, similar to the following messages:

```
Information: 11 enable level shifter cells are merged. (UPF-973)
Information: 1 isolation, 2 enable level shifter, cells are cloned.
(UPF-969)
```

To report the merged or cloned ISO and LS cells, use the `report_isolation_cell` and `report_level_shifter` commands respectively.

Limitations

- The Power Compiler tool supports the merging and cloning of only isolation, level shifter, and enable level shifter cells. For isolation cells, merging and cloning is not supported for none strategy based NOR-style and NAND-style isolation cells. For example, isolation clamp cell of zero-pin retention cannot be merged or cloned by the tool, since there is no explicit isolation strategy for the clamp cell.
- Merging is not supported for non-SPG flow in the Power Compiler tool.

Setting UPF Attributes on Ports and Hierarchical Cells

To specify additional requirements for the ports and hierarchical cells of a power domain, use the `set_port_attributes` and `set_design_attributes` commands.

- [Setting Attributes on Ports](#)
- [Setting Attributes on Macros](#)
- [Setting Design Attributes on Supply Nets and Logic Nets](#)
- [Modeling Unconnected Pins on Macros](#)
- [Specifying Analog Nets](#)
- [Setting Attributes on Hierarchical Cells](#)

Setting Attributes on Ports

To set attributes and their values on the specified ports, use the `set_port_attributes` command. [Table 23](#) lists the attributes and their values for the `set_port_attributes` command.

If you specify the `set_port_attributes` command multiple times on the same object, the last setting overrides the previous settings. If the tool encounters an unrecognized port attribute when reading the UPF, it preserves the attribute in the output UPF.

For the UPF 2.0 syntax, the `set_port_attributes` option accepts arguments as follows:

```
attribute_name attribute_value
```

The UPF 2.1 syntax for the `-attribute` option is as follows:

```
attribute_name {attribute_value}
```

You can mix both styles in a single command. For example,

```
set_port_attributes -elements . \
    -attribute {user_data1 1} -attribute user_data2 false
```

Table 23 UPF Port Attributes

Attribute name	Attribute value	Ports where the attribute can be specified	Use of the attribute
iso_sink	Name of the supply set, DERIVED_DIFF_ONLY, DERIVED_DIVERSE	Output	Identifies the off-chip load of a primary output port
iso_source	Name of the supply set DERIVED_DIFF_ONLY, DERIVED_DIVERSE	Input	Identifies the off-chip driver of a primary input port
related_supply_default_primary	true or not set	Top level input and output ports	Indicates that when the related supply net is not specified, the primary supply of the top-level domain is assumed as the related supply. Used by verification tools so that no assumption is made about the default power supply.
snps_derived	true or false	Input and output supply ports	Indicates that the specified ports have been created by Synopsys tools
repeater_power_net, repeater_ground_net	Name of the supply net	Input and output ports and pins. Cannot be specified on bidirectional ports	Tool inserts a repeater (buffer) to drive the specified port. The inserted buffer is powered by the specified supply net.
UPF_async_clamp_value	0 or 1	Ports or elements	Indicates the clamp value of an asynchronous set or reset pin of an isolation cell

The following example sets the `iso_source` and `iso_sink` attributes on the input and output ports, respectively.

```
prompt> set_port_attributes -ports {in1 in2} -attribute iso_source SS1
prompt> set_port_attributes -ports {out1 out2} -attribute iso_sink SS2
```

You can use the `-driver_supply` and `-receiver_supply` options of the `set_port_attributes` command on bidirectional (`inout`) ports. The tool checks the port status at the `check_mv_design`, `compile_ultra`, and `insert_mv_cells` commands and behaves as follows:

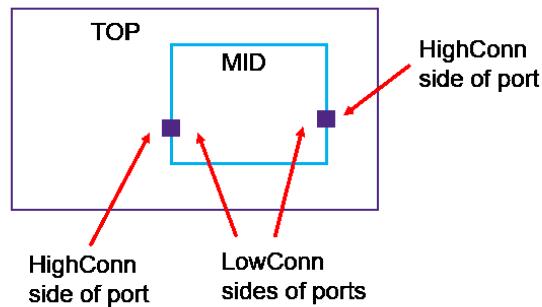
- If you specify only the `-driver_supply` option for an `inout` port, the tool assumes that the receiver supply is the same as the specified driver supply and issues a UPF-494 information message.
- If you specify only the `-receiver_supply` option for an `inout` port, the tool assumes that the driver supply is the same as the specified receiver supply and issues a UPF-494 information message.
- If you specify both options for the same port, the tool checks whether the supplies are electrically equivalent. If the supplies are not equivalent, the tool stops and issues a UPF-485 error message.

Setting Attributes on Macros

Liberty definitions for hard macros might not contain sufficient information to model the interface between the macro and the top-level design.

In UPF terminology, a port has two sides: the HighConn side and the LowConn side, as shown in [Figure 93](#). The HighConn side is visible to the parent of the instance whose interface contains the port. The LowConn side is visible inside the instance. In this example, TOP is the parent and MID is the instance.

Figure 93 Sides of a Port



To specify port properties, use the `set_related_supply_net` command and the `-driver_supply` and `-receiver_supply` options of the `set_port_attributes` command, as follows:

- If you apply the command to a port in the hierarchy at the scope of the command, the attribute applies to the LowConn side of the port. This attribute is used to implement the net connected to the LowConn side.
- If you apply the command to a port in the hierarchy below the scope of the command, the attribute applies to the HighConn side of the port. This attribute is used to implement the net connected to the HighConn side. If there is no attribute set on the HighConn side, the tool uses the attribute set on the LowConn side.

When the tool considers power intent logic insertion outside a hard macro, the receiver supply of an input port on the hard macro is determined in the following order of decreasing priority:

1. A `set_port_attributes -receiver_supply` command specified at the HighConn side of the port
2. A `set_related_supply_net` command specified at the HighConn side of the port
3. Any `related_power_port` information for the pin provided in the Liberty file

When the tool considers power intent logic insertion outside a black box cell, the receiver supply of an input port on the black box cell is determined in the following order of decreasing priority:

1. A `set_port_attributes -receiver_supply` command specified at the HighConn side of the port
2. A `set_related_supply_net` command specified at the HighConn side of the port
3. A `set_port_attributes -receiver_supply` command specified at the LowConn side of the port
4. A `set_related_supply_net` command specified at the LowConn side of the port
5. The primary supply of the power domain

The tool also checks any `set_port_attributes -driver_supply` commands specified at the HighConn side of the port for electrical violations.

If a discrepancy exists between the actual supply and the supply specified in `set_port_attributes` or `set_related_supply_net` commands, the tool issues one or more warning messages.

Setting Design Attributes on Supply Nets and Logic Nets

The Power Compiler tool allows you to use the `set_design_attributes` command to apply user-defined attributes to supply nets and logic nets, which are the only UPF objects allowed in the argument of the `-elements` option.

This usage is primarily for the use of downstream tools. UPF files are used by multiple tools in a design flow, but each tool uses the UPF contents differently.

For example, the following command applies a user-defined attribute to a list of supply nets:

```
dc_shell> set_design_attributes -elements {VDD mid/VDD} \
    -attribute supply_net_attribute true
```

The following command applies a user-defined attribute to a list of logic nets:

```
dc_shell> set_design_attributes -elements {inst1/netA*} \
    -attribute logic_net_attribute true
```

Modeling Unconnected Pins on Macros

If a macro block's pins do not have any related power or ground pins and is not part of a feedthrough path, the tool implicitly assumes that the pins are unconnected. You can explicitly specify that a macro pin is unconnected in the following ways:

- Use the `set_port_attributes -unconnected -ports {...} -model {...}` command
- Set the Liberty `is_unconnected` attribute of the pin

Specifying Analog Nets

A net is an analog net if a pin on the net is an analog pin. You can specify analog pins in two ways:

1. Model the pin as an analog pin using the Liberty `is_analog` pin attribute
2. Use the `set_port_attributes -is_analog` command to specify an analog pin

Isolation, level-shifter, and repeater cells are not inserted on analog nets. No voltage checks for automatically inserted level-shifter cells are performed on analog nets. The tool automatically sets the `dont_touch` attribute on these nets to prevent buffering.

The `check_mv_design` command issues warning messages in the following cases:

- There is a voltage difference between the connected analog pins
- A level-shifter strategy is specified on an analog net
- There is a voltage difference between pins connected to an analog net

Setting Attributes on Hierarchical Cells

To set attributes on a collection of cells, use the `set_design_attributes` command.

Table 24 shows the list of attributes and their values that can be specified on hierarchical cells using the `set_design_attributes` command.

Table 24 *UPF Design Attributes*

Attribute name	Attribute value	Location of the Use of the attribute
<code>derived_external</code> and <code>external_supply_map</code>	Name of the supply set	Hierarchical cell Indicates that the supply sets are reference-only supply sets. These are used with ports with the <code>iso_source</code> and <code>iso_sink</code> attributes. These attributes establish a one-to-one order dependent mapping of the supply sets.
<code>derived_iso_strategy</code>	Name of the isolation strategy	Hierarchical cell To ensure unique strategy name for the derived strategies in the power domain. Used in hierarchical flow to support location fanout.
<code>contains_switches</code>	Power domain	Hierarchical cell Overrides the location of the power switch.
<code>enable_bias</code>	Boolean	Top-level design When set to <code>true</code> , turns on the well bias feature.
<code>lower_domain_boundary</code>	Boolean	Top scope of the design and any hierarchical cell When set to <code>true</code> , the boundary of the power domain extends to the boundary of the power domain below it in the hierarchy.
<code>merge_domain</code>	Boolean	Hierarchical cell Indicates that the two blocks belonging to the same power domain can be merged.
<code>suppress_iss</code>	Power domain	Current design Indicates that supply set handles cannot be created in the power domain.

Table 24 UPF Design Attributes (Continued)

Attribute name	Attribute value	Location of the attribute	Use of the attribute
upf_chip_design	Boolean	Top-level design	Indicates that the design is the TOP block. When the isolation strategy definition contains <code>-location fanout</code> , this attribute causes the primary output ports to be considered as the loads.
correlated_supply_group	Supply net names or wildcard (*) character	Top scope of the design	Indicates that the supply nets of the port state or power state triplets should be considered as correlated voltage range.
legacy_block	Boolean	Hierarchical cell	When set to <code>true</code> , indicates that the block is a legacy block. This is useful while combining two blocks; one defined using domain-dependent supply nets and the other defined using domain-independent supply nets and supply sets. The block defined using domain-dependent supply nets is marked as a legacy block.

For the UPF 2.0 syntax, the `set_design_attributes -attribute` option accepts arguments as follows:

```
{attribute_name attribute_value}* 
```

The UPF 2.1 syntax for the `-attribute` option is as follows:

```
attribute_name attribute_value 
```

You can mix both styles in a single command. For example,

```
set_design_attributes -elements . \
    -attribute {user_data1 1} -attribute user_data2 false 
```

Note:

If the tool encounters an unrecognized design attribute when reading the UPF, it preserves the unrecognized attribute in the output UPF.

Setting Terminal Boundaries

When you set the design attribute `terminal_boundary` to `true` on an instance, the boundary ports of this instance become a terminal boundary. For example, to set a terminal boundary on an instance named U1, do the following:

```
set_design_attributes -elements U1 -attribute terminal_boundary true
```

When you use the `terminal_boundary` attribute, only the root cells of a power domain are allowed in the element list. These cells can be macro cells, block abstracts, black boxes, and other hierarchies.

If you specify a cell that is not a root cell, the tool issues a UPF-210 error message.

When you set the `terminal_boundary` attribute to `true` on a particular block, the tool considers its boundary constraints for performing supply consistency checks. There is no other functional implication of this attribute. The `terminal_boundary` option is used in a bottom-up hierarchical implementation flow. In a flat design, the tool ignores any `terminal_boundary` attribute specified on a nested domain's root cells.

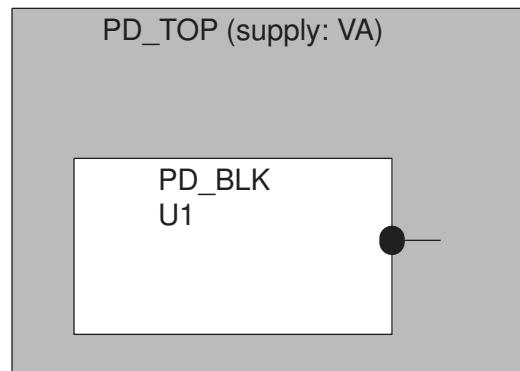
For example, assume you use the following command on the block named U1 shown in Figure 94:

```
set_design_attributes -elements U1 -attribute terminal_boundary true
```

The tool behaves as follows:

- If the block U1 is a .ddc file or a block abstract, during top synthesis, the tool ignores the `terminal_boundary` specified on U1.
- If the block U1 is an ETM or macro, the tool retains the `terminal_boundary` attribute setting on block U1 in the top-level integrated UPF.
- If the block U1, is a black box, the tool retains the `terminal_boundary` attribute setting in the top-level integrated UPF, and it performs a consistency check between the boundary constraint at U1's port and the external driver or load cell supply.

Figure 94 Example of Hierarchical Terminal Boundary



Querying for UPF Design and Port Attributes

To query for UPF predefined attributes of the design objects, that is, attributes that are previously set on the design objects using `set_design_attributes` and `set_port_attributes`, use the `get_upf_design_attribute` and the `get_upf_port_attribute` commands.

The following table provides details on the attributes that can be queried using `get_upf_design_attribute`:

Attribute	Attribute Name	Object Type	Defined If	Value
-attribute {terminal_boundary}	UPF_terminal_bound	instance	Always	TRUE, if the associated instance is marked as terminal boundary
-is_hard_macro	UPF_is_hard_macro	instance	Always	TRUE, if cell derived as a hard macro from Liberty or the <code>set_design_attributes</code> command. FALSE, otherwise
-is_soft_macro	UPF_is_soft_macro	instance	Always	TRUE, if cell is defined soft macro using <code>set_design_attributes</code> . FALSE, otherwise

Attribute	Attribute Name	Object Type	Defined If	Value
enable_bias	UPF_enable_bias	instance	set_design_attributes -attribute enable_bias true is defined at top level	TRUE, if the given instance has enable_bias set to TRUE or if it sits within a block with enable_bias set to TRUE. FALSE, otherwise

The following table provides details on the attributes that can be queried on pin and port objects of the design using `get_upf_port_attributes`:

Attribute	Attribute Name	Defined If	Return Value
-driver_supply	UPF_driver_supply	Object is attributed driver supply	Supply set
		Note: In the case of driver supply where multiple settings can be applied at the same time on an object based on scope, the returned value is the one with higher precedence.	
-receiver_supply	UPF_receiver_supply	Object is attributed with receiver supply	Supply set
		Note: In the case of receiver supply where multiple settings can be applied at the same time on an object based on scope, the returned value is the one with higher precedence.	
-feedthrough	UPF_feedthrough	Object has a valid setting derived from <code>set_port_attributes</code> or Liberty	The collection of shorted pins
-unconnected	UPF_unconnected	Object is valid for <code>set_port_attributes</code> -unconnected setting	TRUE, if object is derived unconnected from <code>set_port_attributes</code> or Liberty. FALSE, otherwise

Attribute	Attribute Name	Defined If	Return Value
-clamp_value	UPF_clamp_value	Object has a valid clamp value setting	The clamp value attributed: 0 1 latch
-is_analog	UPF_is_analog	Always	TRUE, if object is derived is_analog from set_port_attributes or Liberty. FALSE, otherwise
-literal_supply	UPF_literal_supply	Object has a valid set_port_attributes -literal_supply_setting	A supply set, if the setting was supply_set based. Otherwise, a pair of supply nets in the form {power ground}

Assigning Supplies to Pad Ports

A pad cell is identified by the Liberty `is_pad` attribute. Because these cells communicate with the outside world, UPF strategies for multivoltage cell insertion do not apply to pad cells.

When a top-level port connects to a pad cell, the Power Compiler tool does not use the top-level power domain's primary supply for the pad cell pins. Instead, the tool uses the connected pad pin's supply as the related supply of the top-level port.

By default, the Power Compiler tool treats pad cells as follows:

- The tool does not insert power management cells on paths that connect pad cells and top-level ports. If a UPF strategy appears to recommend insertion of a power management cell, the tool issues an information message stating that the insertion is not allowed.
- Multivoltage checking utilities in the tool do not report violations regarding the absence of power management cells on paths that connect pad cells and top-level ports.
- The tool ignores the following commands specified for the top-level port: `set_related_supply_net`, `set_port_attributes -driver_supply`, and `set_port_attributes -receiver_supply`. If these commands specify a supply that is different from the connected pad pin supply, the tool issues an information message.
- If a top-level port connects to both a pin from a cell with the `is_pad` attribute and a pin from a cell without this attribute, the tool applies the pad supply to the top-level port and issues a warning.

- If a top-level port connects to multiple pad cell pins of different voltages, the tool applies a UPF supply to the port instead of any of the pad supplies and issues a warning.
- In some designs, power management cells already exist between a pad cell and the outside world. In this case, the tool does not treat the cell as a pad cell and issues a warning message. In this case, the tool might insert power management cells based on UPF strategies.

You can disable the automatic assignment of the pad supply to the top-level port by setting the `upf_use_driver_receiver_for_io_voltages` variable to `false` (the default is `true`).

Specifying Retention Strategies

The `set_retention` and `set_retention_control` commands specify a strategy for inserting retention cells inside a power-down domain.

The `set_retention` command specifies which registers in the power-down domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality.

The power and ground nets of the retention registers can operate at voltage levels different from the primary and ground supply voltage levels of the power domain where the retention cell is located. Use the `-retention_power_net` and `-retention_ground_net` options to specify the supply nets to be used as the retention power and ground nets.

The retention power and ground nets are automatically connected to the implicit save and restore processes and shadow register. If you specify only the `-retention_power_net` option, the primary ground net is used as the retention ground supply. If you specify only the `-retention_ground_net` option, the primary supply net is used as the retention power supply.

The `-retention_supply` option specifies the supply set whose power and ground functions to use as the retention power and retention ground nets.

If specific objects in the power domain do not require retention capabilities, you can specify them with the `-no_retention` option. The tool maps these objects to library cells that do not have retention capability.

The `-save_condition`, `-restore_condition`, and `-retention_condition` options are intended to capture the clock-dependent retention behavior during simulation. The Power Compiler tool parses these options, but does not use them. However, the tool preserves the options and writes them out at the `save_upf` command if the netlist is not synthesized.

Every retention strategy defined without the `-no_retention` option must have a corresponding `set_retention_control` command. The `set_retention_control` command specifies the retention control signal and retention sense. The command

identifies an existing retention strategy and specifies the save and restore signals and senses for that strategy.

Each control signal can be a port, pin or a net, with a port or pin having higher precedence. The retention signal does not need to exist in the logic hierarchy where the retention cells are to be inserted. The synthesis or implementation tools perform port-punching, as needed, to make the connection. Port-punching automatically creates a port to make a connection from one hierarchical level to the next. These punched ports are not considered for isolation, even though after the port creation, these ports reside within the coverage of an isolation strategy.

The `-assert_r_mutex`, `-assert_s_mutex`, and `-assert_rs_mutex` options of the `set_retention_control` command are intended to capture the clock-dependent retention behavior during simulation. These options are parsed and ignored by the Power Compiler tool.

Specifying Elements to Include in the Retention Strategy

The `-elements` option specifies cells for which the retention strategy applies. In the absence of the `-elements` option, the retention strategy is applied to all sequential cells in the power domain, unless you specify the `-no_retention` option. Note that DesignWare instances are supported when using the `-elements` option with the `set_retention` command. The tool applies the `size_only` attribute on all the elements on which it applies the retention strategy.

The `-update` option allows you to refine the element list of a previously defined retention strategy. When used with the `-elements` option, the set of elements is the union of all elements specified for a strategy. You cannot refine a domain-based retention strategy to an element-based retention strategy with the `-update` option. In the following example, the second `set_retention` command results in an error.

```
create_power_domain MID -elements {mid1 mid2}
set_retention RET1 -domain MID
set_retention_control RET1 ...
map_retention_cell RET1 ...
set_retention RET1 -domain MID -elements {mid1} -update
```

The `set_retention_elements` command defines a list of critical elements that can later be used in a `set_retention` command. The list of elements applies to the scope where the `set_retention_elements` is defined. You must retain all of the elements in the list or none of them. It is an error to have a partially retained list of elements.

The `-exclude_elements` option takes the same types of arguments as the `-elements` option. The specified elements must be part of the domain extent.

Use this option to exclude elements from the retention strategy before implementation, such as when you first create a retention strategy with the `set_retention` command. For

example, the following command applies the RET1 retention strategy to all registers in the PD1 power domain except for the reg1 register:

```
dc_shell> set_retention RET1 -domain PD1 -exclude_elements {reg1}
```

You can also use the `-exclude_elements` option during successive strategy refinement when you use the `set_retention -update` command before implementation. In the following example, strategy RET1 is applied to register u1/reg1 and strategy RET2 is applied to register u1/reg2 and all other registers in hierarchical cell u1:

```
dc_shell> set_retention RET1 -domain PD1 -elements {u1/reg1 u1/reg2}
dc_shell> set_retention RET2 -domain PD1 -elements {u1}
dc_shell> set_retention RET1 -domain PD1 -exclude_elements {u1/reg2}
      -update
```

The Power Compiler tool does not support the following update actions:

- Changing a retention strategy to another type of strategy
- Removing a retention strategy
- Changing another type of strategy to a retention strategy
- Changing one retention strategy to another retention strategy

For these reasons, you cannot use the `-exclude_elements` option after implementation. For example, the following sequence of commands attempts to remove some registers from a retention strategy after implementation. This action is not supported and the tool issues errors and warnings as shown:

```
dc_shell> set_retention RET1 -domain PTOP
dc_shell> compile_ultra
dc_shell> set_retention RET1 -domain PTOP -exclude_elements {reg1 reg2}
Warning: excluded elements will be unassociated from the strategy (Design
Compiler)
dc_shell> compile_ultra -incremental
Error: Retention cells are not associated with any strategy (Design
Compiler)
```

In the following example, the commands attempt to change some retention registers from one strategy to another strategy after implementation. This action is not supported and the tool issues errors and warnings as shown:

```
dc_shell> set_retention RET1 -domain TOP -elements {u1}
dc_shell> compile_ultra
dc_shell> set_retention RET1 -domain TOP -exclude_elements {u1/reg1}
      -update
Warning: excluded elements will be unassociated from the strategy (Design Compiler)
dc_shell> set_retention RET2 -domain TOP -elements {u1/reg1}
dc_shell> compile_ultra -incremental
Error: Retention cells are not associated with any strategy (Design Compiler)
```

Similarly, the tool issues errors if you apply the `-no_retention` option to cells before implementation and subsequently attempt to set a retention strategy on those cells after implementation.

To prevent elements from being disassociated from strategies after implementation, set the `upf_drop_conflict_retention_constraint` variable to `true` (the default is `false`). In this case, the tool issues an error message and stops if it encounters invalid `set_retention` commands after implementation. For example:

```
dc_shell> set_retention RET1 -domain PTOP
dc_shell> compile_ultra
dc_shell> set_retention RET1 -domain PTOP -exclude_elements {reg1}
Error out and drop command because exclude_elements will cause
disassociation of reg1 from RET1
```

If you write a UPF file before executing an action command, the UPF contains the excluded elements even if they are not in the domain extent. After an action command, the UPF contains only those excluded elements that are in the domain extent.

Resolving Retention Strategy Precedence

The following strategies have decreasing order of precedence, irrespective of the order in which they are executed:

1. Strategies that apply to registers, explicitly specified using the `-elements` option.
2. Strategies that apply to registers, implied by specifying a Verilog `process` or `always` block.
3. Strategies that apply to registers, implied by specifying an instance using the `-elements` option.
4. Strategies that apply to registers, implied by specifying only the power domain name.

Strategies with the `-no_retention` option have higher precedence than strategies without this option.

Examples of Retention Strategy Precedence Resolution

The following examples illustrate retention strategy precedence resolution.

Example 41 General Retention Strategy Resolution

```
set_retention RET1 -domain PD1 -elements {inst1} -no_retention
set_retention RET2 -domain PD1 -elements {inst1/reg1} ...
```

In [Example 41](#), RET1 has the granularity of an instance and RET2 has the granularity of a register. For inst1/reg1, even though RET1 has the `-no_retention` option specified, it has less precedence than RET2.

Example 42 Verilog Sample

```
module mid(input clk,d, output reg q1, q2, en);
  always @(posedge clk)
    begin: blk1
      if en ==1'b0
        q1 <= d;
        q3 <= d;
      end
    end
    always @(negedge clk)
    begin: blk2
      q2 <= d;
    end
  endmodule
```

For the example in [Example 42](#), the following strategies are defined:

```
set_retention RET1 -domain MID -elements {mid/blk1}
set_retention RET2 -domain MID -elements {mid/q1}
set_retention RET3 -domain MID -elements {mid/blk1}
set_retention RET4 -domain MID -elements {mid/q1}
```

RET1 and RET3 are in conflict for q3. RET2 and RET4 are in conflict for q1. So, RET3 and RET4 are removed and only the first two strategies are written out. If strategies conflict, the tool retains the strategy created first.

Using the Retention Supply as the Primary Supply

Normally, the primary supply of the domain powers both the retention register and the receiver supply of the retention register's data input. By default, the output driver of the register is also powered by the primary supply of the domain. Therefore, the driver supply of the data output is the primary supply.

However, you can specify that the register and its output is powered by the retention supply by using the `-use_retention_as_primary` option of the `set_retention` command. You cannot update this option. If you use this option, it can affect source and sink analysis during level-shifter, isolation, and repeater cell insertion since the driver or receiver supply uses the retention supply.

When a strategy has the `-use_retention_as_primary` option specified, the tool only uses the library cells specified in the `map_retention_cell` command that have output pins related to the backup PG pin.

Choosing Specific Library Cells With Retention Strategies

Use the `map_retention_cell` and `map_retention_clamp_cell` commands to constrain the library cell choices for retention registers.

The following guidelines apply to the `map_retention_cell` command:

- The argument of the command must be a retention strategy that is defined for the power domain specified by the `-domain` option. The retention strategy and the `-domain` option are mandatory.
- The optional `-lib_cell_type` option directs the tool to select a retention cell that has the specified cell type. However, the value specified with this option does not change the simulation semantics specified by the `set_retention` command.
- The `retention_cell` attribute on the library cells in the target library defines the retention styles of the library cells.
- The optional `-lib_cells` option specifies a list of library cells that can be used for retention cells.
- The optional `-lib_model_name` option specifies the retention register verification model in the input UPF files, which is parsed for syntax. The model information is primarily used by the Formality tool. The model information is not captured in the design database or in the UPF file written after synthesis. When you specify the port mapping information, the tool accepts either the UPF 2.0 or 2.1 syntax, as follows:

```
map_retention_cell -lib_model_name name1 \
    {-port_map port_name net_ref}* \
map_retention_cell -lib_model_name name \
    -port_map {{port_name net_ref}*}
```

The following guidelines apply to the `map_retention_clamp_cell` command:

- The `map_retention_clamp_cell` command is not a UPF command. Therefore it must not be included in UPF files. The tool issues an error message at the `load_upf` command if a UPF file contains the `map_retention_clamp_cell` command. In addition, the tool does not write this command into saved UPF files.

- You can use this command only after loading a UPF file.
- The argument of the command must be a list of retention strategies that are defined for the power domain specified by the `-domain` option. The retention strategies and the `-domain` option are mandatory.
- Use the `-clock_clamp_lib_cells` option to specify the list of library cells that must be used for mapping zero pin retention clamp cells on clock paths.
- Use the `-async_clamp_lib_cells` option to specify the list of library cells that must be used for mapping zero pin retention clamp cells on asynchronous set or reset paths.
- The specified library cells must be isolation or enable level-shifter library cells.

The following guidelines apply to both the `map_retention_cell` and `map_retention_clamp_cell` commands:

- Library cells with `dont_touch` attributes are not used for mapping.
- The operating conditions of the target library cells must match the design environment.
- If none of the specified library cells are suitable, the tool leaves the cells as GTECH isolation cells in the final netlist.
- If a mapping constraint is not specified, the tool selects library cells from the target library in the following order:
 - NOR isolation cells
 - Dual-rail isolation cells
 - GTECH isolation
- If you specify a mapping command more than one time, the tool honors the last successfully accepted command.
- Cell mapping is discarded only if the applicable power domain is removed.

To limit the usage of library cells specified in `map_retention_clamp_cell` for mapping only zero pin retention clamps and no other isolation strategies, set the `upf_iso_map_exclude_zpr_clamp_lib_cells` variable to `true` before running the `map_retention_clamp_cell` command. The default is `false`. However, if you specify the same library cells in both the `map_retention_clamp_cell` and `map_isolation_cell` commands, the `map_isolation_cell` command has higher precedence than the global control provided by the variable.

If the design contains pre-instantiated isolation cells mapped to library cells specified in the `map_retention_clamp_cell` command, the tool remaps them to other library cells during the next mapping step.

To obtain information about retention cell clamp cell constraints, use the `report_retention_clamp_cell -verbose` command and specify the power domain with the `-domain` option.

Zero-Pin Retention Support

The Power Compiler tool supports the inference of zero-pin retention (ZPR) cells and the automatic insertion of retention clamp cells, during synthesis.

During the `compile` flow, the tool iterates through all the ZPR retention cells in the design, detects their ZPR-related attributes and any associated clamp cells, and inserts and maps clamp cells to satisfy the ZPR requirements.

Additionally, when running the `insert_mv_cells` command, run the command with the `-retention_clamp` option, to insert retention clamp cells for pre-existing ZPR cells (if the clamp cells are not already present).

Note:

The `insert_mv_cells -all` command does not insert retention clamp cells. You must explicitly specify the `-retention_clamp` option to insert retention clamp cells.

To disable the retention clamp cell insertion, set the `upf_skip_retention_clamp_insertion` variable to `true`. The default value of this variable is `false`.

Inferring Complex Retention Cells

A complex sequential cell is a cell whose functionality is unknown. You can specify that the tool infer complex retention cells in place of complex nonretention cells during synthesis. Enable this feature by setting the following variable:

```
set_app_var upf_infer_complex_retention_cells true
```

Usually, the Liberty modeling of complex sequential cells have pin names that match the pin names of complex retention sequential cells (except for the save and restore pins).

If you have the following in your UPF:

```
set_retention -elements {complex_nonretention_cell}
map_retention_cell -lib_cells {complex_retention_library_cell}
```

The tool tries to infer a complex retention cell during synthesis based on pin name matching. The tool performs the following steps:

1. Matches the pin names of the nonretention cell to the pin names of the retention library cells specified in the `map_retention_cell` command.
2. Chooses the first retention library cell with names that match exactly. Uses this cell in place of the nonretention cells.
3. If no matching retention library cell is found, then the tool issues a warning message.

With the pin name matching approach, the tool selects the first matching retention library cell from the `map_retention_cell` command. If there are multiple matches and if you want to select a particular retention library cell, set the `retention_equivalent` attribute on the library cells. For example,

```
set_attribute {nonretention_library_cell} \
    retention_equivalent {retention_library_cell}
```

If this attribute is set, then a retention library cell can be used in place of a nonretention library cell if the retention library cell is valid. A valid retention library cell has the `retention_cell` attribute on it, is present in the list of target libraries, and is listed as a library cell in the `map_retention_cell` command. If the library cell is not a valid retention library cell, the tool falls back on pin name matching.

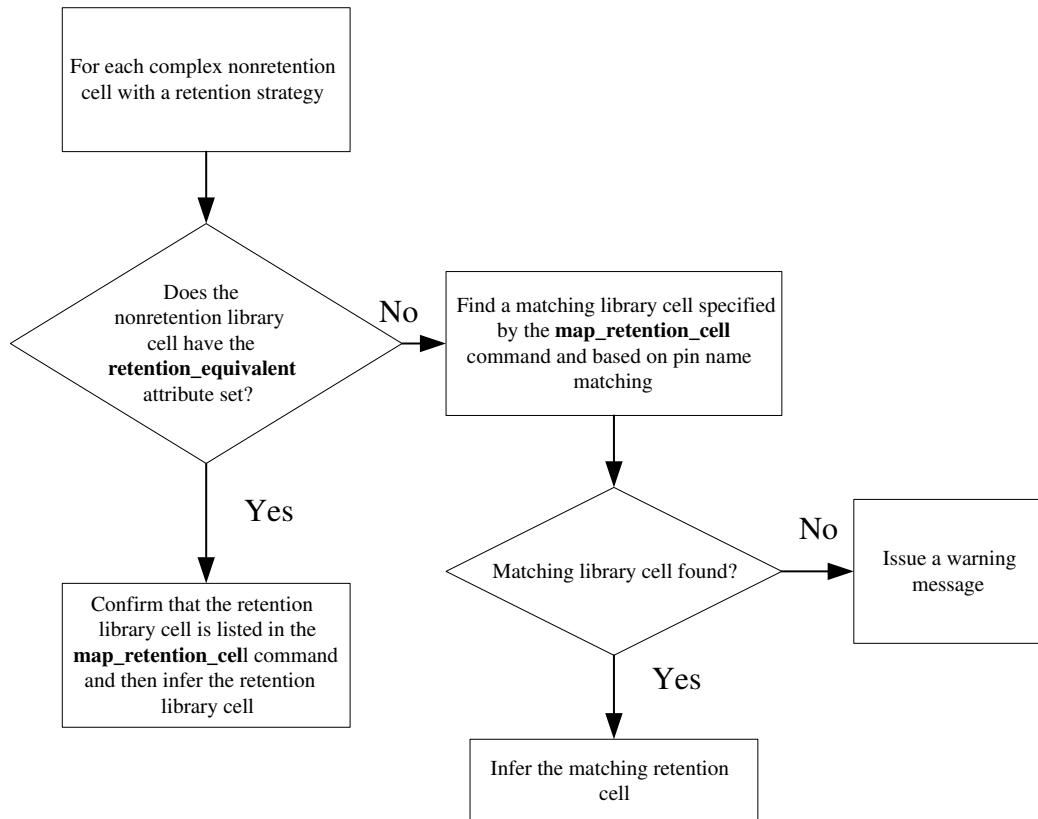
If you have library cells that do not have matching pin names, you can use the same `retention_equivalent` attribute to specify the pin mappings. For example,

```
set_attribute {nonretention_library_cell} \
    retention_equivalent {retention_library_cell {pin1 pin2} \
        {pin3 pin4}}
```

Pin1 of the nonretention library cell corresponds to pin2 of the retention library cell. Pin3 of the nonretention library cell corresponds to pin4 of the retention library cell. If the pin mapping is not valid, then the tool falls back on pin name matching.

[Figure 95](#) shows how the tool infers complex retention cells.

Figure 95 Inferring Complex Retention Cells



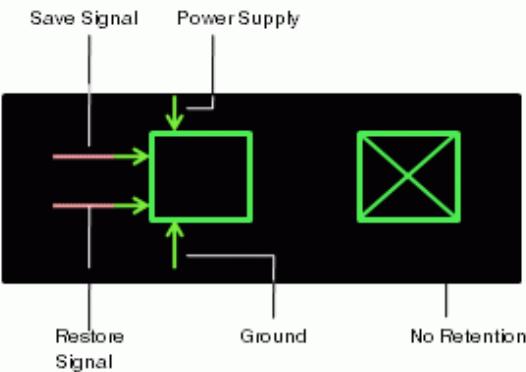
Retention Strategy and Clock-Gating Cells

When you define retention strategy for a power domain, by default, the Power Compiler tool does not apply the retention strategy to the clock-gating cells in the power domain. The tool does not issue warning or information message. However, if you set the `upf_use_additional_db_attributes` variable to `false`, the tool issues a UPF-117 warning message for every power domain defined with a retention strategy and contains clock-gating cells. Formal verification also flags a failure in this situation.

Representing Retention Strategies in the UPF Diagram View

In the UPF diagram view, the retention cell is represented by a green bounding box as shown in Figure 96. The symbol includes pins for power and ground and segments for save and restore signals. The no-retention symbol contains a “X” inside the bounding box.

Figure 96 Representation of Retention Cells in the UPF Diagram



All retention symbols are located at the center of their parent power domains. The diagram displays the supply nets connected to the retention strategy, the domains to which the strategy belongs and their save and restore signals.

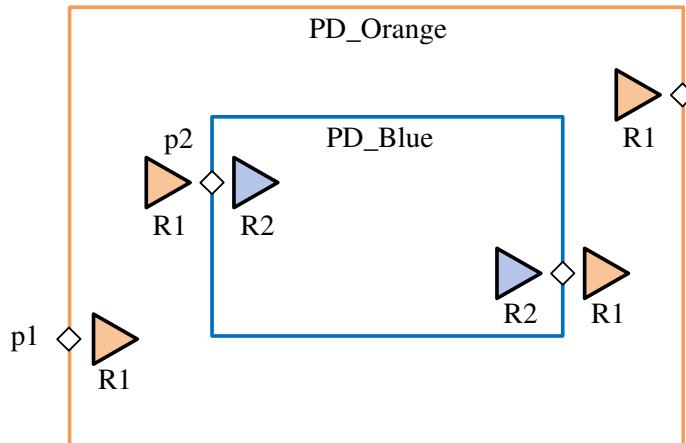
Specifying Repeater Strategies

Repeaters are buffers inserted at regular intervals along the length of a long net to maintain sufficient drive strength along the full length of the net. In the Power Compiler tool, the `set_repeater` command defines a strategy for inserting repeater cells (buffers) on the interface of a power domain. The tool inserts a buffer using a specified power supply.

[Figure 97](#) shows an example of the `set_repeater` command. The corresponding script is as follows:

```
set_design_attributes -elements {.} \
    -attribute lower_domain_boundary true
set_repeater R1 -domain PD_Orange \
    -repeater_supply ss_orange
set_repeater R2 -domain PD_Blue \
    -elements {.} \
    -repeater_supply ss_blue
```

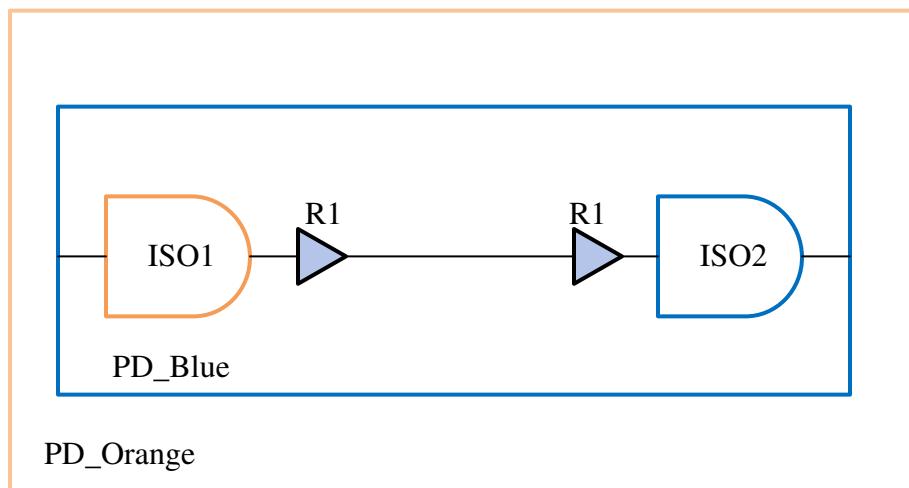
Figure 97 Example Using the set_repeater Command



Repeater cells are inserted before isolation and level-shifter cells are inserted. The presence of repeaters can affect the implementation of isolation and level-shifter strategies that use the `-source` or `-sink` options. Repeaters are considered endpoints for source and sink analysis.

Isolation and level-shifter cells are placed close to the domain boundary, that is, between the domain crossing and the repeater as shown in [Figure 98](#).

Figure 98 Repeaters and Power Management Cells



For the example in [Figure 98](#), the following isolation strategies would apply:

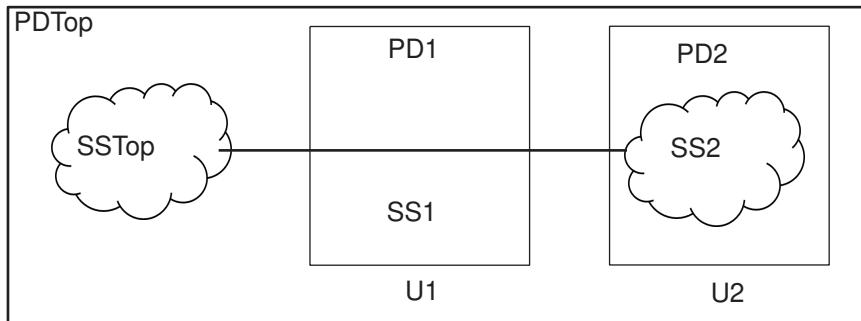
```
set_repeater R1 -domain PD_Blue -repeater_supply SS_Blue
set_isolation ISO1 -domain PD_Blue -isolation_supply SS_Orange \
    -applies_to inputs -sink SS_Blue
set_isolation ISO2 -domain PD_Blue -isolation_supply SS_Blue \
    -applies_to outputs -source SS_Blue
```

Specifying Supplies for Repeaters

Repeaters are buffers inserted at regular intervals along the length of a long net to maintain sufficient drive strength along the full length of the net. In Design Compiler, the `insert_buffer` command lets you specify the number of such buffers to insert into a net by using the `-no_of_cells` option.

If a long net crosses a power domain boundary, such as in the case of a feedthrough net crossing through a power domain, repeater buffers inserted inside the power domain must, by default, maintain the always-on characteristics of the sink domain. For example, in [Figure 99](#), PDTOP power domain is more always-on than PD2 power domain, and PD2 power domain is more always-on than PD1 power domain. The feedthrough path through PD1 power domain must be always-on with respect to PD2 power domain.

Figure 99 Feedthrough Path in PD1 Power Domain Before Buffer Insertion



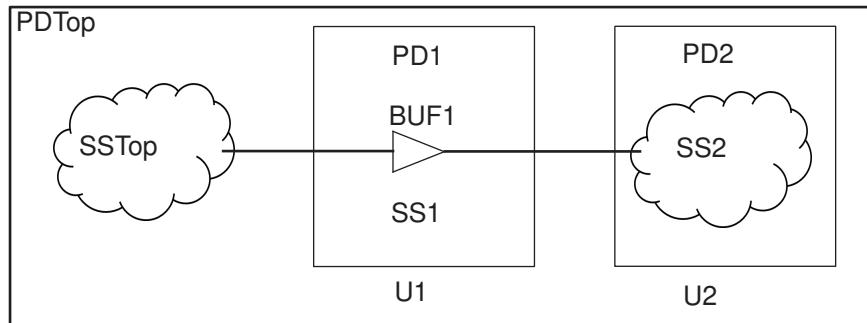
However, the always-on requirement might not be needed in certain cases. For example, if the feedthrough net is a DFT scan signal that is used only when all power domains are active, the inserted buffers can use the PD1 power supply, thereby using less resources. In other cases, depending on the floorplan, the power supplies of the sink domain might not be easily available where the buffer needs to be inserted.

To enforce the insertion of a buffer with a specific power supply on the feedthrough path, use the `-repeater_supply` argument of the `set_port_attributes` command. For example,

```
dc_shell> set_port_attributes -elements {U1} \
    -applies_to outputs -repeater_supply ss1
```

The tool inserts repeater buffers that drive the output port of elements U1 and uses the supply set SS1 to power these buffers. This results in the buffering shown in [Figure 100](#).

Figure 100 Feedthrough Path in PD1 Power Domain With a Buffer Inserted



Insertion of the repeater buffers might cause the need for additional level shifter and isolation cells on the feedthrough path. When you specify the insertion of a repeater buffer to drive a specified port, you must also specify the power supply for the buffer. The supply specified with the `repeater_supply` attribute must be available in the scope of the power domain where the buffer is inserted. You can specify either a supply set using the `-repeater_supply` option as shown in the previous example, or a pair of supply nets (power and ground) using the `-attribute` option, as shown in the following example:

```
dc_shell> set_port_attributes -elements {U1} -applies_to outputs \
    -attribute repeater_power_net VDD \
    -attribute repeater_ground_net VSS
```

You must specify the `-attribute` option two times in the same command, to specify the power and ground nets for the inserted repeater buffers. Multiple occurrences of the `-attribute` option are allowed only for the `repeater_power_net` and `repeater_ground_net` attributes.

The repeater insertion is performed by the `compile_ultra`, `insert_mv_cells` or the `insert_dft` command, before inserting other power management cells. The tool inserts either a single non-inverting repeater (buffer) or a pair of inverters. You cannot specify the type of the repeater to be inserted.

After inserting the repeater, during level-shifter and isolation cell insertion, the tool ensures that the repeater insertions do not cause electrical violations and inserts a level-shifter or an isolation cell to fix the violation. However, you must have defined an isolation strategy for the tool to insert the isolation cell. The `check_mv_design` checks and reports any violation introduced and not fixed by the repeater insertion.

If the repeater does not use the primary supply of the domain as the supply, the `save_upf` command writes the `connect_supply_net` command for the PG pins of the repeater.

Deferring Element Definitions in Power Management Strategies

The Power Compiler tool allows you to specify a power management strategy and defer its applicable element list definition to a later step. You can do this by using an empty element list in the `set_isolation`, `set_level_shifter`, `set_repeater`, and `set_retention` command specifications. See the following examples:

```
dc_shell> set_isolation ISO1 -domain PD1
           -isolation_supply SS -elements {}

dc_shell> set_level_shifter LS1 -domain PD1
           -elements {}

dc_shell> set_retention RET1 -domain PD1
           -retention_supply SS -elements {}

dc_shell> set_repeater REP1 -domain PD1
           -repeater_supply SS -elements {}
```

Note:

The tool treats any power management strategy with only invalid objects in its element list, or only non-power-domain objects in its element list, as an empty element strategy.

Matching Tool and IEEE LRM Defaults

For a few UPF commands of Synopsys implementation tools, for some options that take an enumerated value, the default value of the option differs from its equivalent in the current IEEE LRM 3.1 version. For such options, to change the Power Compiler tool option defaults to match the equivalent LRM defaults, you can use the `lrm_option_defaults` UPF attribute. See the following example:

```
dc_shell> set_design_attributes -elements {.}
           -attribute lrm_option_defaults 3.1
```

The following table lists the UPF commands and their options affected by this design attribute:

UPF Command	Command Option	Synopsys Default	LRM 3.1 Default
<code>set_isolation</code>	<code>-applies_to_boundary</code>	upper	both
<code>set_level_shifter</code>			
<code>set_repeater</code>			

UPF Command	Command Option	Synopsys Default	LRM 3.1 Default
set_isolation	-applies_to	outputs, for domain-level strategies	No default Note: When matching with the LRM default, the tool uses both as the default for this option.
set_isolation	-diff_supply_only	FALSE	TRUE

Here are all the usage rules for this attribute:

- The attribute takes a string value which currently can only be '3.1'. If you specify any value other than '3.1' for this attribute value, the tool reports an UPF-210 error message.
- If you are specifying this design attribute, you must use the `-elements { . }` option. Using the `-model` option is not supported.
- The design should not have any power objects, such as a power domain, supply net, supply port, or supply set, which are already created before you set this design attribute. Otherwise, the tool reports an UPF-210 error message.
- The precedence rule is the following: the user-specified strategy-specific `-applies_to_boundary` option value has the highest precedence. Additionally, the scope-specific `lower_domain_boundary` design attribute has precedence over the `lrm_option_defaults` design attribute.
- In a bottom-up hierarchical flow, if a block has this attribute set to a specific value, then the top-most design must also have the attribute set to the same value.

Creating Power Switches

The `create_power_switch` command creates a virtual instance of a power switch in the scope of the specified power domain. power switch has at least one input supply port and one output supply port. When the switch is off, the output supply port is shut down and has no power.

The `create_power_switch` command lets the tool know that a generic power switch resides in the design at a specific scope or level of hierarchy. The off state of the power switch output is used in the power state table. The Power Compiler tool does not perform

power switch insertion, but the information is passed to the IC Compiler II tool for implementation.

The `map_power_switch` command defines which library cells to use for a specific UPF power switch.

The `-domain` option is available, but not required, for both the `create_power_switch` and `map_power_switch` commands. Without the `-domain` option, the command scope is the current scope. If the `-domain` option is used, the command scope is the scope of the specified power domain.

The following example is a definition of a power switch named SW1.

```
create_power_switch SW1 \
    -domain PD_TOP \
    -output_supply_port {SWOUT VDD1g} \
    -input_supply_port {SWIN1 VDD1} \
    -control_port {CTRL swctl} \
    -on_state {ON VDD1 {!swctl}}
```

The UPF standard requires a simple name for the power switch. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv_input_enforce_simple_names` variable to `false`.

You can override the location of the power switch by using the `contains_switches` attribute of the `set_design_attributes` command. For the following example, even though you are in scope U0, the power switch is placed in scope U2.

```
set scope U0
create_power_switch SW1 ...
set_design_attributes -elements {U2} -attribute contains_switches {S1}
```

You can use the `contains_switches` attribute more than one time on a hierarchical cell. If you do, the power switches are added to the list of switches related to a hierarchical cell. For example,

```
set_design_attributes -elements {U2} -attribute contains_switches {S1}
set_design_attributes -elements {U2} -attribute contains_switches {S2}
```

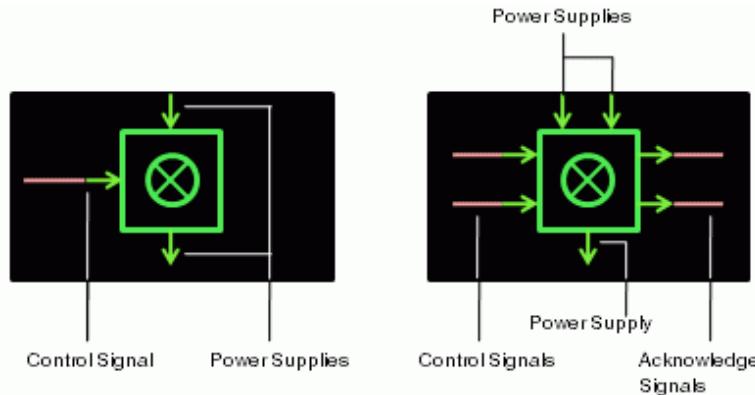
is the same as specifying

```
set_design_attributes -elements {U2} -attribute contains_switches {S1 S2}
```

Representation of Power Switches in the UPF Diagram View

In the UPF diagram view, a power switch is represented by a green circle with an X inside it, as shown in [Figure 101](#).

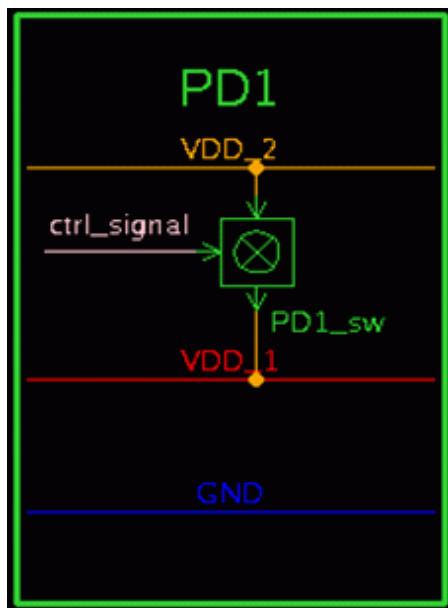
Figure 101 Representation of a Power Switch



The symbol indicates the input and output supply ports, the control ports, and the control signals. The arrows represent the direction of the ports.

As shown in [Figure 101](#), a power switch can have single or multiple control signals. The power switches are located within the boundaries of their parent power domain. Because power switches have supply nets as input and output, they are located between the power supply nets as shown in [Figure 102](#).

Figure 102 Location of the Power Switches in the Power Domain



Power Models

The Power Compiler tool supports the IEEE 1801 construct called a power model. A power model allows you to define the UPF for a macro in a self-contained environment. Each time this macro is instantiated, the power model is loaded from memory instead of calling the `load_upf` and `connect_supply_net` commands for each instantiation.

To define a power model, use the `define_power_model` command. The `apply_power_model` and `add_parameter` commands are also used when defining or specifying the use of power models.

To check your power models, use the `report_power_model` command. This command displays information including where the power models are defined and where they are applied.

Topics covered in this section:

- [Configuring Power Compiler for Power Models](#)
 - [Defining and Applying a Power Model](#)
 - [Excluding Designs From Using Power Models](#)
 - [Hard and Soft Macros](#)
-

Configuring Power Compiler for Power Models

To simplify loading power model definitions, configure the following variables:

- `upf_power_model_library`
Specifies a list of power model UPF files
- `upf_power_model_search_path`
Specifies a list of search paths that the tool uses to find the power model UPF files

Defining and Applying a Power Model

Power models are IEEE 1801 commands encapsulated within the `define_power_model` command. The following is an example of power model definition for a power model named `my_model`:

```
define_power_model my_model -for {lib_cell_A lib_cell_B} {
    add_parameter DOMAIN -default "PD_TOP" -description "top power domain"
    create_supply_net VDD
    create_supply_net VSS
    create_supply_set SS -function {power VDD} -function {ground VSS}
    create_power_domain PD_$DOMAIN -supply {primary SS} -include_scope
}
```

The `add_parameter` command specifies the names of parameters to be defined inside a power model. These parameters can be overridden from the `apply_power_model` command.

When you have defined a power model, you can map the model to a list of cell instances. For example,

```
apply_power_model my_model -elements (u1/macro u2/macro) ...
```

Excluding Designs From Using Power Models

If you want to exclude designs from using power models, do the following:

```
set_design_attributes -models model_list -is_hard_macro false
```

or

```
set_design_attributes -models model_list \
    -attribute UPF_is_hard_macro false
```

Hard and Soft Macros

You can mark specific models as hard or soft macros for hierarchical implementation by using the `set_design_attributes` command. Use the `-models` option to specify the model names along with either the `-is_soft_macro true` or `-is_hard_macro true` option. When these options are specified, the tool defines either the `UPF_is_soft_macro` or `UPF_is_hard_macro` attribute for the specified models. If you specify the model list as `{.}`, the command applies to the model corresponding to the current scope.

You must specify the type of macro before loading the UPF for the macro. Alternatively, you can include the `set_design_attributes` command as the first command in the UPF file provided with the `load_upf -scope` command for the macro. For a hard macro, you can also specify the UPF with the `define_power_model` command.

A hard macro cell typically has a Liberty model that defines its interface, including supply ports and related supplies for its logic ports. A macro defined with the Liberty `is_macro_cell` attribute is treated as a hard macro. A hard macro has one of the following:

- No UPF specification
- A self-contained UPF specification
- A UPF specification that does not define its own top-level domain

A soft macro always has a self-contained UPF. A hard macro might or might not have a UPF because it is not separately implemented by the tool.

For UPF processing, the tool performs checks for terminal boundaries on both hard and soft macros. The tool checks for a self-contained UPF for soft macros.

By default, the `find_objects` command considers all instances of hard and soft macros as leaf cells. The `-traverse_macros` option allows the command to traverse the macro terminal boundaries.

Power State Tables

A power state table defines the legal combination of states that can exist simultaneously during the operation of the design. A power state table is a set of power states of a design in which each power state is represented as an assignment of power states to individual power nets. A power state table of a design captures all the possible operational modes of the design in terms of power supply levels. Given a power state table, a power state relationship (including voltage and relative always-on relations) can be inferred between any two power nets. The power state table is used by the synthesis tool for analysis, synthesis, and optimization of the multivoltage design.

Default Power States

The Power Compiler tool supports the use of the default or predefined power states ON and OFF. You can refer to these states during the early definition stage of the UPF, before the actual supply voltages are defined.

You can use the default power states to define other power states in the `add_power_state` command. Using the `-update` option the first time you refer to the default power states is optional, even though the states already exist. However, the `-update` option is required for all subsequent commands that refer to the default states.

You can choose to use the ON and OFF names for other power state definitions.

The following example uses the default state ON in the definition of the new state NOR:

```
create_power_domain TOP -elements { . } -supply {primary}
add_power_state -domain TOP \
    -state {NOR -logic_expr {TOP.primary==ON && ... }}
```

You must fully define the default ON and OFF states before performing any action commands or checking commands. For example:

```
add_power_state TOP.primary \
    -state {ON -supply_expr {power=={FULL_ON 1.08} \
    && ground=={FULL_ON 0.0}} -update}
```

Power State Propagation

You can control the propagation of power states by setting the `enable_state_propagation_in_add_power_state` design attribute on a top-level or block-level design. The default of this attribute is `false`.

When the attribute is `true`, the following conditions apply:

- The tool propagates the name of the supply set state specified in the `add_power_state` command to the functional nets.
- You can subsequently use the net and the supply set state in the `create_pst` and `add_pst_state` commands.
- You cannot use the `&&` operator in the `-supply_expr` option of the `add_power_state` command.

When the attribute is `false`, the following conditions apply:

- The tool does not propagate the name of the supply set state specified in the `add_power_state` command to the functional nets.
- You cannot use the net and the supply set state in the `create_pst` and `add_pst_state` commands. To create power state tables, you must use the `create_power_state_group` command and refer to the supply set state names in the group definition.
- You can use the `&&` operator in the `-supply_expr` option of the `add_power_state` command.

The Power Compiler tool supports setting the

`enable_state_propagation_in_add_power_state` design attribute to any combination of `true` and `false` for top-level or block-level designs. However, to use the `characterize` command for a design, you must set the attribute to the same value for the top-level design and all of its block-level designs.

Creating Power State Tables

To create a power state table, use the `add_power_state` and `create_power_state_group` commands. Assuming that the supply sets are already defined, the following example defines the power states for three supply sets:

```
add_power_state \
    -supply SS1 -state ON  {-supply_expr {power == {FULL_ON 0.8} && \
                                    {ground == {FULL_ON 0}}}}
add_power_state \
    -supply SS2 -state ON  {-supply_expr {power == {FULL_ON 0.8}}} \
    -state OFF {-supply_expr {{power == {OFF}}}}
add_power_state \
    -supply SS3 -state ON  {-supply_expr {power == {FULL_ON 0.8}}}} \
    -state OFF {-supply_expr {{power == {OFF}}}}
```

Next, create a power state group as follows:

```
create_power_state_group MY_PST
```

Finally, build the power state table using the specified states and the power state group:

```
add_power_state -group MY_PST \
-state RUN12 {-logic_expr {SS1 == ON && SS2 == ON && SS3 == ON}}
-state RUN1  {-logic_expr {SS1 == ON && SS2 == ON && SS3 == OFF}}
-state RUN2  {-logic_expr {SS1 == ON && SS2 == OFF && SS3 == ON}}
-state SLEEP {-logic_expr {SS1 == ON && SS2 == OFF && SS3 == OFF}}
```

The resulting power state table is shown in [Table 25](#).

Table 25 Power State Table for MY_PST

State	SS1	SS2	SS3
RUN12	ON	ON	ON
RUN1	ON	ON	OFF
RUN2	ON	OFF	ON
SLEEP	ON	OFF	OFF

Using internal state names, the tool builds the actual power state table as shown in [Table 26](#).

Table 26 Power State Table Using Internal State Names

State	SS1.ground	SS1.power	SS2.ground	SS2.power
RUN12	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_3	SNPS_INT_ON_5
RUN1	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_3	SNPS_INT_ON_6
RUN2	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_4	SNPS_INT_ON_5
SLEEP	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_4	SNPS_INT_ON_6

Hierarchical Power State Tables

When creating power state tables, you can build them hierarchically using existing power state tables and combining them to form a larger table. Suppose you want to increase the number of supply sets in [Table 25](#) to add supplies SS4 and SS5 as shown in [Table 27](#).

Table 27 Adding Supplies to the Power State Table

State	SS1. ground	SS1. power	SS2. power	SS3. power	SS4. power	SS5. power
RUN12_NORM	GND	ON	ON	ON	ON	ON
RUN_OVD	GND	ON	ON	ON	OVD	ON
RUN1_NORM	GND	ON	ON	OFF	ON	ON
RUN1_UND	GND	ON	ON	OFF	ON	OFF
RUN2_NORM	GND	ON	OFF	ON	ON	ON
RUN2_UND	GND	ON	OFF	ON	ON	OFF
SLEEP_OFF	GND	ON	OFF	OFF	OFF	OFF

The new power state table can be built hierarchically using the existing MY_PST table as follows:

- Create a new power group:

```
create_power_state_group MY_PST2
```

- Create the power states for the supplies in the new power group:

```
add_power_state -group MY_PST2 \
    -state NORM {-logic_expr {SS4==ON && SS5==ON}} \
    -state OVD {-logic_expr {SS4==OVD && SS5==ON}} \
    -state UND {-logic_expr {SS4==ON && SS5==OFF}} \
    -state OFF {-logic_expr {SS4==OFF && SS5==OFF}}
```

These power states combine to form a new power state table as shown in [Table 28](#).

Table 28 Power State Table MY_PST2

State	SS4	SS5
NORM	ON	ON
OVD	OVD	ON
UND	ON	OFF
OFF	OFF	OFF

- Create the new combined power state table:

```
create_power_state_group SYSTEM_PST
add_power_state -group SYSTEM_PST
    -state RUN12_NORM {-logic_expr {MY_PST==RUN12 && MY_PST2==NORM}} \
    -state RUN12_OVD {-logic_expr {MY_PST==RUN12 && MY_PST2==OVD}} \
    -state RUN1_NORM {-logic_expr {MY_PST==RUN1 && MY_PST2==NORM}} \
    -state RUN1_UND {-logic_expr {MY_PST==RUN1 && MY_PST2==UND}} \
    -state RUN2_NORM {-logic_expr {MY_PST==RUN2 && MY_PST2==NORM}} \
    -state RUN2_UND {-logic_expr {MY_PST==RUN2 && MY_PST2==UND}} \
    -state SLEEP_OFF {-logic_expr {MY_PST==SLEEP && MY_PST2==OFF}}
```

The power state table, SYSTEM_PST, is shown in [Table 29](#). This table is equivalent to the power state table in [Table 27](#).

Table 29 Power State Table SYSTEM_PST

State	MY_PST	MY_PST2
RUN12_NORM	RUN12	NORM
RUN12_OVD	RUN12	OVD

State	MY_PST	MY_PST2
RUN1_NORM	RUN1	NORM
RUN1_UND	RUN1	UND
RUN2_NORM	RUN2	NORM
RUN2_UND	RUN2	UND
SLEEP_OFF	SLEEP	OFF

Creating Power State Groups in Hierarchies Having State Propagation Enabled

You can create power state groups in hierarchies having state propagation enabled. That is, when you have a BLOCK with state propagation disabled and a TOP design with state propagation enabled, you can create power state groups in TOP and use these groups to refer to the states in BLOCK, to define the complete power state table (PST).

From the groups created in TOP, you can refer to all the following four types of states defined in BLOCK:

- PST states
- Supply set states
- Domain states
- Other group states

Groups, however, cannot refer to port states added using the `add_port_state` command and net states added using the `add_supply_state` command. You must use `create_pst` to refer to port states or net states.

Defining domain states in TOP is identical to defining group states. You can create domain states too in TOP UPF similar to group states.

Example

The following example shows how to create a power state group in TOP and define the relationship between group state of BLOCK and TOP PST states.

Block has group state GS in group GROUP_B.

Top has PST TOP_PST having power state PS1.

The group GROUP_T in top is referring to block's group state GS and top's PST state PS1 in group state ST1

```

# BLOCK.upf
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON {-supply_expr
    {power=={FULL_ON 1} && ground=={FULL_ON 0}}}
...
create_power_state_group GROUP_B
add_power_state GROUP_B -state GS {-logic_expr {SS==ON && SS1==ON} }

# TOP.upf
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state TRUE
...
create_pst TOP_PST -supplies          {VDD VDD1}
add_pst_state PS1 -pst TOP_PST -state {ON ON}

create_power_state_group GROUP_T
add_power_state GROUP_T -state ST1
    {-logic_expr {TOP_PST==PS1 && BLOCK/GROUP_B==GS} }

```

Bottom-Up Hierarchical Flow

In this tool, use the `propagate_constraints` command to propagate UPF constraints from a synthesized BLOCK to TOP, as part of bottom-up hierarchical flow. In this flow, with state propagation enabled for BLOCK, group states in BLOCK are propagated to TOP with and without domain merging differently.

Without Domain Merging

In this flow, the tool propagates groups created in BLOCK (with state propagation enabled) to TOP. The state propagation value of TOP is irrelevant.

With Domain Merging

Consider a scenario where BLOCK has state propagation enabled. (It is irrelevant whether TOP has state propagation enabled or disabled.) Consider you have created domain states in BLOCK. During domain merging, domain states in BLOCK and the supply sets are deleted when they are merged with TOP domain. In this scenario, to preserve the domain states defined in BLOCK, to ensure that the system PST is not impacted, the tool:

- Creates a new group by name “`<domain_name>_group`” in BLOCK
- Transfers the domain states in BLOCK to the newly created group

Bottom-Up Hierarchical Flow Example

Consider this hierarchical design example with TOP > BLOCK [MID > BOT].

In this example, the hierarchical design TOP has a lower hierarchical BLOCK. TOP has the PDTOP power domain with the `merge_domain` design attribute set to true. The BLOCK

has the PDTOP power domain for BLOCK scope with domain states and another lower scope PST called MID_PST.

```
# MID.upf
set_design_attributes -elements {.}
    -attribute enable_state_propagation_in_add_power_state TRUE
...
create_pst MID_PST
add_pst_state s0 -pst MID_PST
set_scope BOT
set_design_attributes -elements {.}
    -attribute enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON
    {-supply_expr {power=={FULL_ON 1} && ground=={FULL_ON 0}}}
...
set_scope ../
create_power_domain PDTOP
add_power_state PDTOP -state PS1
    {-logic_expr {MID_PST==s0 && BOT/SS==ON} }

# TOP.upf
set_design_attributes -elements {.}
    -attribute enable_state_propagation_in_add_power_state TRUE
create_power_domain PDTOP
set_design_attributes -elements {MID} -attribute merge_domain TRUE
```

During propagate_constraints, the PDTOP domain of BLOCK is merged with the TOP level PDTOP domain and the group states of BLOCK are propagated to TOP.

The domain states of BLOCK's domain are retained in BLOCK by creating another power state group called PDTOP_group.

```
# TOP.upf [full chip UPF]
create_power_domain PDTOP
set_scope MID
set_design_attributes -elements {.}
    -attribute enable_state_propagation_in_add_power_state TRUE
...
create_pst MID_PST
add_pst_state s0 -pst MID_PST
set_scope BOT
set_design_attributes -elements {.}
    -attribute enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON
    {-supply_expr {power=={FULL_ON 1} && ground=={FULL_ON 0}}}
...

# Back to MID scope, new group created in block scope
set_scope ../
create_power_state_group PDTOP_group
```

```
add_power_state PDTOP_group -state PS1
{-logic_expr {MID_PST==s0 && BOT/SS==ON} }
```

Top-Down Hierarchical Flow

Use the `characterize` command to partition the block from a full design. During partitioning, the derivation of the PST in the block UPF depends on the state propagation setting in the design:

State Propagation is Enabled for the Entire Design

In this case, the tool characterizes a derived PST into the block. The derived PST consists of supply port states and supply net states.

State Propagation is Disabled for the Entire Design

In this case, the tool characterizes a derived PST plus a derived group into the block. The derived PST consists of supply port states and supply net states. The derived group consists of supply set states and derived PST states.

Design has a Mixture of State Propagation Values

In this case, the design has state propagation enabled for some blocks and disabled for some other blocks. As in the preceding case, the tool characterizes a derived PST and a derived group into the block. The derived PST consists of supply port states and supply net states. The derived group consists of supply set states and derived PST states.

Top-Down Hierarchical Flow Examples

Here are a few examples of the characterized UPF for three different scenarios. In all these examples, the RTL has three hierarchies or modules: top, mid, and bot. The instance of the mid module is `mid_inst` and the instance of the bot module is `bot_inst`.

Note:

In the following examples, to keep the UPF simple, supply ports and their connections to the supply sets are not shown.

Example: State propagation is disabled for the entire design

Consider the following full-chip UPF. In this UPF, state propagation is disabled for the entire design. There are three power domains (one for each hierarchy) and the primary supply sets of the three domains have one state each.

```
# Full-chip UPF
set_design_attributes -elements {}
-attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST
create_power_domain PDTOP -supply {primary SST}
add_power_state SST -state SST1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
```

```

set_scope mid_inst
create_supply_set SSM
create_power_domain PDMID -supply {primary SSM}
add_power_state SSM -state SSM1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}

set_scope bot_inst
create_supply_set SSB
create_power_domain PDBOT -supply {primary SSB}
add_power_state SSB -state SSB1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
set_scope /

```

Consider that the mid block is characterized. The SST supply set that is defined in the top-most scope is brought into mid (as it is available in mid). The power states of SST are brought into mid as they are. The following is the characterized UPF for the mid block:

```

# Characterized UPF for block mid
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST
add_power_state SST -state SST1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}

create_supply_set SSM
create_power_domain PDMID -supply {primary SSM}
add_power_state SSM -state SSM1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}

set_scope bot_inst
create_supply_set SSB
create_power_domain PDBOT -supply {primary SSB}
add_power_state SSB -state SSB1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
set_scope $CURRENT_TOP_SCOPE

create_power_state_group group
add_power_state -group group -state group_ps_1
    {-logic_expr {SST == SST1 && SSM == SSM1 && bot_inst/SSB == SSB1}}

```

The original full-chip UPF does not have any port or net states. Therefore, the characterized UPF does not have any derived PST. It only has a derived group.

Example: Design has a mixture of state propagation values

Consider the following full-chip UPF. In this UPF, state propagation is enabled for some blocks and disabled for some other blocks. The following are the state propagation values for the different blocks:

- top: state propagation = FALSE
- mid_inst: state propagation = TRUE
- bot_inst: state propagation = FALSE

```
# Full-chip UPF
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST
create_power_domain PDTOP -supply {primary SST}
add_power_state SST -state SST1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
add_power_state SST -state SST2 {-supply_expr {power == {FULL_ON 1.0} &&
                                                ground == {FULL_ON 0.0}}}

set_scope mid_inst
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state TRUE

set_scope bot_inst
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state FALSE
add_power_state SSB -state SSB1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
set_scope /
```

Consider that the mid block is characterized. The SST supply set that is defined in the top-most scope is brought into mid (as it is available in mid). But the power states of SST cannot be brought into mid, as they are (as mid has state propagation enabled). So, the states of SST are brought into mid as internal port states (with prefix SNPS_INT_).

The following is the characterized UPF for the mid block:

```
# Characterized UPF for block mid
set_design_attributes -elements {}
    -attribute enable_state_propagation_in_add_power_state TRUE
set_design_attributes -elements {bot_inst}
    -attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST

set_scope bot_inst
add_power_state SSB -state SSB1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
set_scope $CURRENT_TOP_SCOPE
```

```

create_supply_port SST_power_port -direction in
create_supply_port SST_ground_port -direction in
connect_supply_net SST.power -ports SST_power_port
connect_supply_net SST.ground -ports SST_ground_port
add_port_state SST_power_port -state {SNPS_INT_SST1_3 0.900000}
add_port_state SST_power_port -state {SNPS_INT_SST2_7 1.000000}
add_port_state SST_ground_port -state {SNPS_INT_SST1_4 0.000000}

create_pst pst -supplies [list SST_power_port SST_ground_port]
add_pst_state pst_ps_1 -pst pst -state {SNPS_INT_SST1_3 SNPS_INT_SST1_4}
add_pst_state pst_ps_2 -pst pst -state {SNPS_INT_SST2_7 SNPS_INT_SST1_4}

create_power_state_group group
add_power_state -group group
  -state group_ps_1 {-logic_expr {bot_inst/SSB == SSB1
                                    && pst == pst_ps_1}}
  -state group_ps_2 {-logic_expr {bot_inst/SSB == SSB1
                                    && pst == pst_ps_2}}

```

A derived PST is created for the internal port states and a derived group is created combining the supply set states and the derived PST states.

Example: Design has state propagation disabled and the tool creates derived states

Consider the following full-chip UPF. In this UPF, state propagation is disabled for the entire design.

```

# Full-chip UPF
set_design_attributes -elements {.}
  -attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST
create_power_domain PDTOP -supply {primary SST}
add_power_state SST -state SST1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}

set_scope mid_inst
create_supply_set SSM
create_power_domain PDMID -supply {primary SSM}
add_power_state SSM -state SSM1 {-supply_expr {power == {FULL_ON 0.9} &&
                                                ground == {FULL_ON 0.0}}}
add_power_state SSM -state SSM2 {-supply_expr {power == {FULL_ON 1.0}}}
set_scope /

```

The SSM2 power state has only one function (power). So, this state is interpreted as:

```

add_power_state SSM -state SSM2 {-supply_expr {power == {FULL_ON 1.0}
                                              && ground == *}}

```

Here, “*” expands to all states and voltage values that SSM.ground can take. In the preceding UPF, SSM.ground has only one voltage value defined, which is 0.0. So, in this case, the SSM2 power state expands to:

```
add_power_state SSM -state SSM2 {-supply_expr {power == {FULL_ON 1.0}
&& ground == {FULL_ON 0.0}}}
```

During `characterize`, the tool detects that there is no state in the original UPF that matches this state and the tool creates a derived state for only the ground function, as shown in the following statement:

```
add_power_state SSM -state SNPS_DERIVED_1
{-supply_expr {ground == {FULL_ON 0.0}}}
```

This derived supply set state is then used in the derived group.

The following is the characterized UPF for the mid block:

```
# Characterized UPF for block mid
set_design_attributes -elements {}
-attribute enable_state_propagation_in_add_power_state FALSE

create_supply_set SST
add_power_state SST -state SST1 {-supply_expr {power == {FULL_ON 0.9} &&
                                               ground == {FULL_ON 0.0}}}

create_supply_set SSM
create_power_domain PDMID -supply {primary SSM}
add_power_state SSM -state SSM1 {-supply_expr {power == {FULL_ON 0.9} &&
                                               ground == {FULL_ON 0.0}}}
add_power_state SSM -state SSM2 {-supply_expr {power == {FULL_ON 1.0}}}
add_power_state SSM -state SNPS_DERIVED_1
{-supply_expr {ground == {FULL_ON 0.0}}}

create_power_state_group group
add_power_state -group group
-state group_ps_1 {-logic_expr {SST == SST1 && SSM == SSM1}}
-state group_ps_2 {-logic_expr {SST == SST1 && SSM == SSM2 &&
                               SSM == SNPS_DERIVED_1}}
```

Reconciling Voltages in Power State Tables

In a hierarchical design, the tool creates a final system power state table after checking the consistency between power states and power state tables in the top-level and block-level power state tables. Normally, any inconsistency in voltages and supply states for power state tables results in the tool dropping that state.

However, you can continue with the flow without dropping states by allowing the tool to perform voltage reconciliation, either globally or for specific block boundaries.

To enable voltage reconciliation across all hierarchical boundaries with an infinite voltage matching tolerance, use the following command:

```
dc_shell> set_design_attributes -elements {} \
    -attribute upf_reconciliation_automatic true
```

This specification takes precedence over any block-specific attributes and thresholds. The tool considers every hierarchical boundary to be a reconciliation boundary. In a top-down flow, the power states of the higher scope power state table are pushed to the lower scope power state tables. In a bottom-up flow, the tool checks consistency between attributes pushed from the block level to the top level.

However, you can continue with the tool flow without dropping states by setting the `upf_reconcile_boundary` design attribute to mark a block boundary. To specify that the tool should skip all power state table checking below the boundary hierarchy, set the attribute to `skip`, as follows:

```
dc_shell> set_design_attributes -models IP1 \
    -attribute upf_reconcile_boundary "skip"
```

To specify that the tool should reconcile voltages for all power state tables below the boundary hierarchy, set the attribute to `\`, as follows:

```
dc_shell> set_design_attributes -models IP1 \
    -attribute upf_reconcile_boundary "reconcile_voltages"
```

If you reconcile the voltages, you can set a voltage tolerance that defines a range of acceptable voltages for block-level states. For example, the following command specifies a single tolerance on a specific supply:

```
dc_shell> set_variation -supply {BLK/VDD} -tolerance {0.2}
```

The following command specifies a negative tolerance (the first value) and a positive tolerance (the second value) on a specific supply:

```
dc_shell> set_variation -supply {BLK/VDD} -tolerance {0.2 0.4}
```

The following command specifies a global tolerance on the entire design:

```
dc_shell> set_variation -tolerance {0.2}
```

Example of Voltage Reconciliation

Consider the top-level power state table shown in [Table 30](#) and the block-level power state table shown in [Table 31](#), which have similar supply sets.

Table 30 Top-Level Power State Table

State	VDD1	VDD2	VDD3	VDD4
S0	1.0	1.4	0.75	0.9
S1	1.25	1.4	0.85	1.15
S2	0.85	1.3	0.85	1.15
S3	OFF	1.3	0.85	1.15

Table 31 Block-Level Power State Table

State	VDD1	VDD2	VDD3	VDD4
S0	1.0	1.4	0.75	0.9
S1	1.25	OFF	0.85	1.15
S2	0.9	1.2	0.75	1.25

The following command sets a voltage tolerance of 0.1 for the block-level states:

```
dc_shell> set_variation -tolerance {0.1}
```

The block-level power state table with the resulting voltage ranges is shown in [Table 32](#).

Table 32 Block-Level Power State Table After Range Expansion

State	VDD1	VDD2	VDD3	VDD4
S0	0.9-1.1	1.3-1.5	0.65-0.85	0.8-1.0
S1	1.15-1.35	OFF	0.75-0.95	1.05-1.25
S2	0.8-1.0	1.1-1.3	0.65-0.85	1.15-1.35

The top-level S0 state is aligned with the block-level S0 state. The top-level S1 state is dropped due to a conflict in the VDD2 value of the block-level S1 state. The top-level S2 state is aligned with the block-level S2 state. The top-level S3 state is dropped due to a conflict in VDD1. The final system power state table is shown in [Table 33](#).

Table 33 Final System Power State Table After Voltage Reconciliation

State	VDD1	VDD2	Vdd3	VDD4
S0	1.0	1.4	0.75	0.9
S2	0.85	1.3	0.85	1.15

Reporting Voltage Reconciliation Constraints

To display the voltage tolerances set on the supplies, use the `report_pst -reconcile` command. The report is similar to the following:

```
dc_shell> report_pst -verbose -reconcile
-----
Reconciliation skipped blocks
  : N/A
-----
Reconciliation on blocks
  : mid
-----
Reconciliation thresholds
Global      : (-, -)
Threshold applied on supplies :
SS1.power   : (-0.10, +0.10)
SS1.ground  : (-,-)
SS2.power   : (-0.10, +0.10)
SS2.ground  : (-,-)
-----
Scope       : top
-----
Supply names : SS1, SS1, SS2, SS2
Resulting power states in this scope:

    SS1|    SS1|    SS2|    SS2|
    [p] |    [g] |    [p] |    [g] |
drv : ON1| ON4| ON1| ON5|
drv : ON2| ON4| ON2| ON5|
-----
```

Reporting Power State Tables

The `report_pst` command generates a report of the power states in the current design.

For example, consider a flow that includes the following commands to define power states and associate supply nets with supply set power states:

```
add_power_state SST -state S1 {-supply_expr \
    {power == '{FULL_ON, 0.8, 0.9, 1.0} && ground == '{FULL_ON, 0.0}}} \
add_power_state SSM -state S2 {-supply_expr \
    {power == '{FULL_ON, 0.8, 0.9, 1.0} && ground == '{FULL_ON, 0.0}}} \
add_power_state SSM -state S3 {-supply_expr \
    {power == '{FULL_ON, 0.8, 1.0, 1.2} && ground == '{FULL_ON, 0.0}}}
```

The report generated by the `report_pst -verbose` command is similar to the following:

	SST	SSM
drv:	S1	S2
drv:	S1	S3

The `-verbose` option writes the full list of state names in the report. Without this option, the report shows an asterisk (*) to indicate that all states can be used.

To indicate voltage values in a power state table, use the `-voltage_type` option. The valid arguments are `all` and `nominal`. The report header includes the labels [p], [g], [nw] and [pw] to indicate functional nets defined on the supply sets for power, ground, n-well, and p-well supplies respectively.

For the power state table in the example, the report generated by the `report_pst -verbose -voltage_type nominal` command is similar to the following:

	SST	SSM
	[p][g]	[p][g]
drv:	S1 [0.9][0.0]	S2 [0.9][0.0]
drv:	S1 [0.9][0.0]	S3 [1.0][0.0]

The report generated by the `report_pst -verbose -voltage_type all` command is similar to the following:

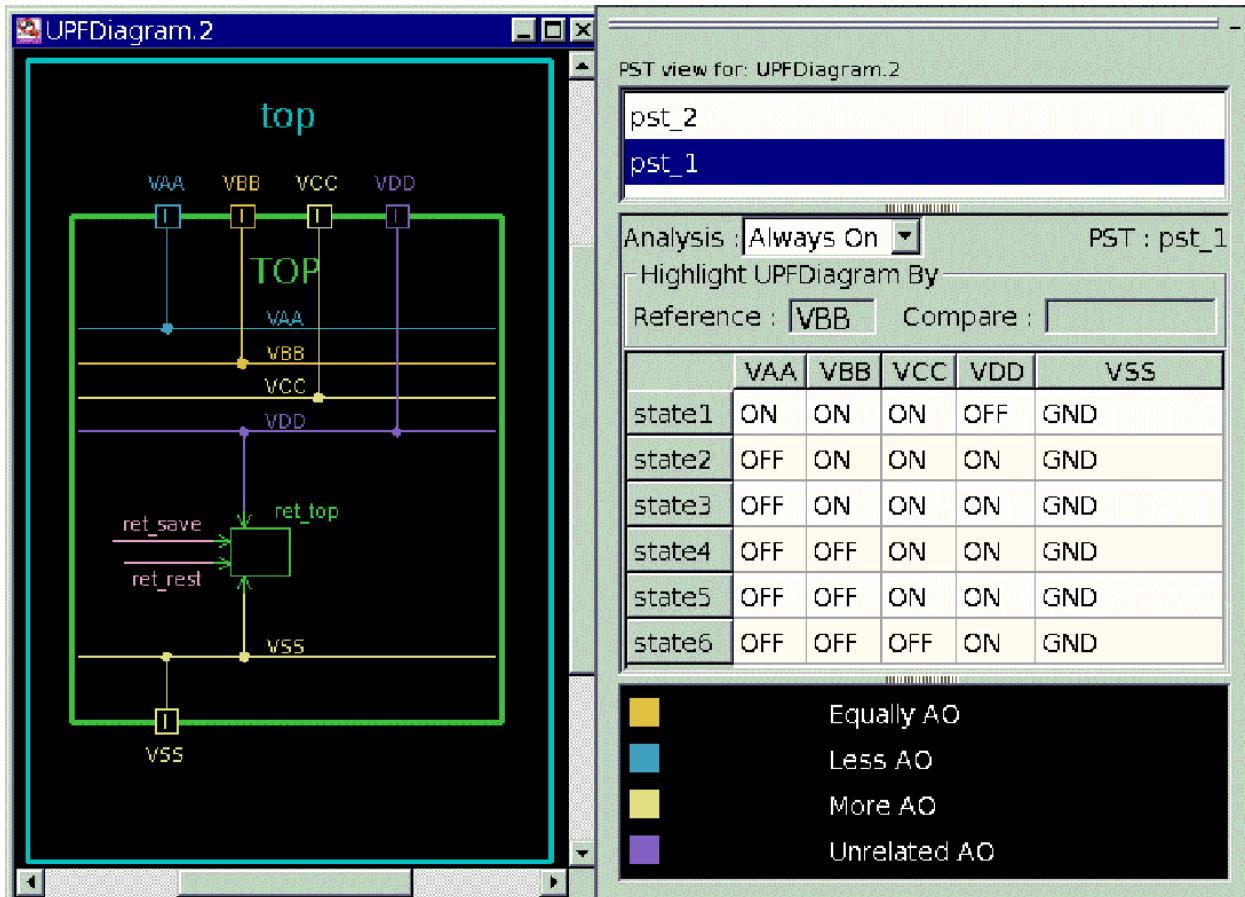
	SST	SSM
	[p][g]	[p][g]
drv:	S1 [0.8, 0.9, 1.0][0.0]	S2 [0.8, 0.9, 1.0][0.0]
drv:	S1 [0.8, 0.9, 1.0][0.0]	S3 [0.8, 1.0, 1.2][0.0]

Visually Analyzing Power State Tables in the UPF Diagram View

To analyze and debug the isolation and the level-shifter strategies in a UPF design, use the UPF diagram view with the Power State Table panel. You can view the power states for each supply in a power state table and examine their relationships in the UPF diagram.

[Figure 103](#) shows an example of the UPF diagram view and Power State Table panel during always-on analysis.

Figure 103 Always-On Analysis Using the Power State Table Panel



The Power State Table panel appears automatically when you open the UPF diagram view. You can hide or display this panel by choosing View > Toolbars > Power State Table.

The Power State Table panel provides the following types of analysis:

- *Always-on analysis* compares the on-off states between any two supplies, including both power and ground supplies
- *Multivoltage level-shifter analysis* compares the voltage relationships between supplies

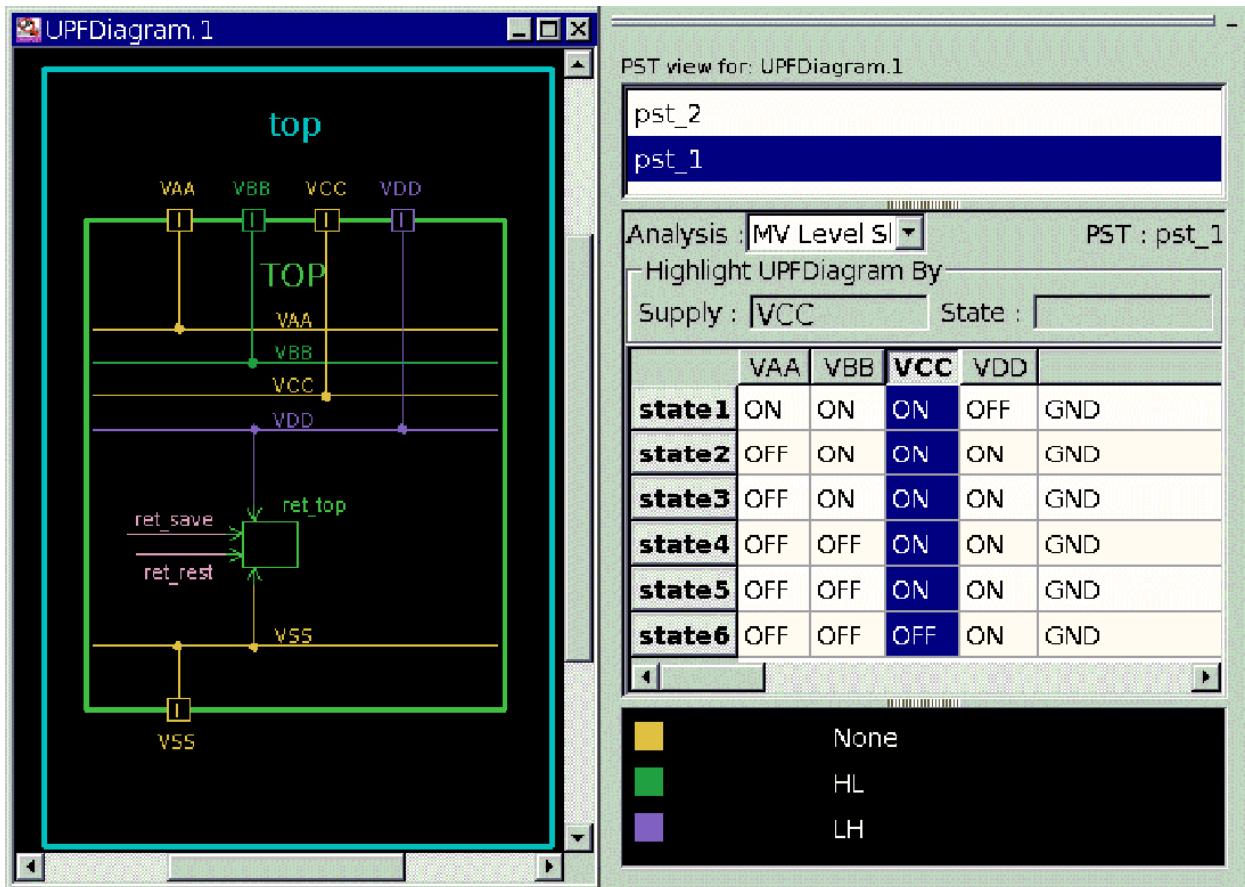
During always-on analysis, you can compare the power and ground supplies in the power state table because the combination of the power and ground supplies defines the always-on relationships between the power domains.

For more details, see, [Analyzing Multivoltage Design Connections in the GUI](#).

During multivoltage level-shifter analysis, to decide if a level shifter is needed between a driver and a load supply, only the power supplies of the power domains need to be compared; the tool supports 0 volts or the off state for the ground supply.

[Figure 104](#) shows an example of the UPF diagram view and Power State Table panel during multivoltage level-shifter analysis.

Figure 104 Multivoltage Level-Shifter Analysis Using the Power State Table Panel



For more information, see the “Visualizing Power State Tables” topic in Design Vision Help.

Support for Well Bias

Some process technologies allow dedicated voltage supplies, instead of normal rail voltages, to be applied to n-well and p-well regions of the chip. Applying a bias voltage to a well changes the threshold voltage for transistors in the well, affecting the performance and leakage current.

The Power Compiler tool offers an optional mode to specify the n-well and p-well bias supply infrastructure using UPF commands. In this mode, the tool automatically makes supply connections to the well bias pins.

To enable the UPF-based well bias mode, set the `enable_bias` design attribute to `true` in the UPF command file as follows:

```
set_design_attributes -elements { . } -attribute enable_bias true
```

With the `enable_bias` design attribute set to `true`, supply sets and supply set handles can be used to specify the bias supply connections. The well bias pins, along with the power supply and ground pins, are connected automatically for standard cells, macros, and other types of cells.

For more information about well bias modeling, see the *Library Compiler User Guide*.

Using a Non-Bias Block in a Bias-Enabled Design

The Power Compiler tool supports the use of non-bias blocks inside a bias-enabled design, without defining the non-bias blocks inside separated scopes. To enable this feature, set the `upf_allow_non_bias_domain_in_bias_scope` variable to `true`:

```
dc_shell> set upf_allow_non_bias_domain_in_bias_scope true
```

This feature allows you to use libraries that have library cells both with and without bias PG-pins, without adding any restriction to them. You can mix supply sets with two functions and four functions inside the same scope, without implementing scoped connections through supply ports for having them in different sections of your design.

Note:

- This feature is disabled by default.
- When using this feature, the top-most domain must be bias enabled. You can define power domains that have bias-disabled blocks as root cells, inside bias-enabled scopes; however, all the root cells must have a matching bias definition either bias on or bias off.
- When using this feature, the tool still creates p-well and n-well supply set functions for the non-bias block scoped in a bias-enabled design. However, you need not define the voltages for these implicit supply sets.
- With the flexibility of inserting library cells with bias PG-pins in non-bias regions, if you have matching operating conditions and matching power and ground voltages between bias and non-bias sections of your design, you could have bias library cells mapped into your non-bias region. To avoid this behavior, use the `set_target_library_subset` command to ensure that the non-bias block is mapped to a proper non-bias PG-pin exposed library cell.

Skipping Bias Checks

In the Power Compiler tool, instantiating cells without bias PG pins, in a bias design, is not allowed. During the execution of the `check_mv_design` command, the tool performs this bias check and, for each cell violating this rule, depending on the type of the cell, prints an appropriate warning message.

However, you might want to use a few non-bias library cells in both bias and non-bias designs. To cater to this requirement, you can specify a set of library cells and cell instances for the Power Compiler tool to skip the bias checks. Use the `set_non_bias_approved_list` command with the `-lib_cells` option to specify the list of library cell reference names for the tool to skip the bias check. Similarly, use the command with the `-cells` option to specify the list of cell instance names for the tool to skip the bias check.

```
dc_shell> set_non_bias_approved_list
           -lib_cells list_of_library_cells

dc_shell> set_non_bias_approved_list
           -cells list_of_cell_instances
```

Note:

- This feature does not skip the bias checks on power management cells namely, isolation, level-shifter, and enable-level-shifter cells, repeaters, retention registers, and always-on buffers and inverters, with the only exception being power switch cells. You can use this feature to skip the bias checks on all other types of cells such as power switch cells.
- If power management cells are specified in the option lists of any of the preceding two options of the `set_non_bias_approved_list` command, the tool does not skip the bias checks on these cells. The tool prints the UPF-975 summary message informing you on the number of library cells and cell instances on which the bias check is not skipped.

Inserting Power Management Cells

Power management cells such as level shifters and isolation cells are not usually part of the original design description. They are inserted during the logic synthesis flow. Buffer-type level shifters can be inserted by the tool as part of compilation. You can also insert them manually by instantiating the cells in RTL or by using specific commands that insert level shifters. Similarly, isolation cells and enable-type level shifters can be instantiated at the RTL level of the design description or inserted by using commands that insert isolation cells.

You can also insert these cells by using the `insert_mv_cells` command. This command uses the strategies defined in the UPF file when inserting these cells. Using options of the `insert_mv_cells` command, you can choose to insert only the isolation cells or only the level shifter cells, or both. By default, the command inserts both isolation and level-shifter cells. You can use this command on both RTL and gate-level designs.

The naming convention of the inserted cells is as follows:

```
<name_prefix>_snps_<power_domain_name>_<isolation_strategy_name>_snps_
<pin_name>_<instance_index>_<name_suffix>
```

For enable level-shifter cells with both a related isolation strategy and a level-shifter strategy, the isolation and level-shifter prefix and suffix are added to the new name. The new naming convention for these enable level-shifter cells is as follows:

```
<levelshift_prefix><isolation_prefix>_snps_<power_domain_name>_
<isolation_strategy_name>_snps_<pin_name>_<instance_index>_
<isolation_suffix><levelshift_suffix>
```

There is no change in the naming of level-shifter and retention cell instances.

The `insert_mv_cells` command inserts the power management cells in the following order:

1. Repeaters or buffers
2. Isolation cells
3. Level-shifter cells
4. Enable level-shifter cells. Based on the requirement, replace the isolation cells by enable level-shifter cells.

Table 34 summarizes the command option and command sequences that can result in the insertion of enable level-shifter cells.

Table 34 Command Sequences and Enable Level-Shifter Cell Insertion

Command option and sequence	Enable level-shifter cell inserted
<code>insert_mv_cells -all</code>	yes
<code>insert_mv_cells -isolation -level_shifter</code>	yes
<code>insert_mv_cells -isolation insert_mv_cells -level_shifter</code>	yes
<code>insert_mv_cells -level_shifter</code> <code>insert_mv_cells -isolation insert_mv_cells -level_shifter</code>	yes

Table 34 Command Sequences and Enable Level-Shifter Cell Insertion (Continued)

Command option and sequence	Enable level-shifter cell inserted
<code>insert_mv_cells -level_shifter</code>	no
<code>insert_mv_cells -isolation</code>	no
<code>insert_mv_cells -level_shifter</code> <code>insert_mv_cells -isolation</code>	no

Note:

You must uniquify your design by using the `uniquify` command before inserting the power management cells. Otherwise, the Power Compiler tool issues an error message.

Relaxing PVT Library Constraints for Power Management Cells

During linking and compilation, if you do not have PVT library cells that match the operating conditions of your power management cells, the tool issues library setup error messages (LIBSETUP-001). To run the synthesis flow without addressing unavailable PVT library cells, use the `set_upf_cell_mismatch` command.

When you use the `set_upf_cell_mismatch` command, you can specify which constraints can be violated for power management cells. This command applies to retention, isolation, and level-shifter cells. If the tool cannot map a cell to one in the target library with matching constraints, it maps the cell to a target library cell that violates the constraint.

After you compile the design, the tool issues information messages to indicate the number of cells that are mapped with relaxed PVT constraints. You can also use the `report_upf_cell_mismatch` command to report which cells are mapped by relaxing constraints.

Reviewing the UPF Specifications

After specifying the power constraints using UPF, you can review the design using the commands or using the GUI.

Commands to Query and Edit Design Objects

To query and edit design objects, use the following commands:

- `find_objects`

The command finds logical hierarchy objects within the specified scope and returns the hierarchical names that match the specified criteria. The command returns a null string when nothing matches the specified search pattern.

- Query commands

To query the UPF objects, use the following commands. The query is resolved when the command is executed, and the result of the query is used by the Power Compiler tool.

- `query_cell_instances`

Returns a list of instance names for all instances of a given reference cell in the current scope of the design.

- `query_cell_mapped`

Returns the reference cell name of a given cell instance.

- `query_net_ports`

Returns a list of ports logically connected to a specified net. By default, the command returns only the ports present at the level of the current scope.

- `query_port_net`

Returns the name of the net logically connected to a specified port, if any such net exists.

- `query_port_state`

Returns information about the port states that have been previously defined using the `add_port_state` command

- `query_pst`

Returns information about the power state tables previously defined with the `create_pst` command

- `query_pst_state`

Returns information about the states that have been previously defined with the `add_pst_state` command

- `query_power_switch`

Returns information about the power switches previously defined with the `create_power_switch` command

- `query_map_power_switch`

Returns information about the power switch library cells previously mapped to the UPF power switches with the `map_power_switch` command

- Editing commands

The editing commands are not written in the UPF file written by the `save_upf` command. However, the changes to the netlist are available in the Verilog and VHDL netlist written by the tool.

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

Reviewing the Power Intent Using the Design Vision GUI

The Power menu in the GUI allows you to specify, modify, and review your power architecture. It also lets you view the UPF diagram and examine the UPF specification defined in your design.

If you have not defined the power intent for your design, see [Defining Power Intent With UPF Commands](#).

If you have already defined the power intent for your design, the Visual UPF dialog box displays the details of your power specification. Using the Power Domains and Power Domain Properties sections, you can edit the power definitions: add new components, redefine the association of the hierarchical cells with the power domains, delete a power domain, and so on.

To open the Visual UPF dialog box:

- Choose Power > Visual UPF

When you open the Visual UPF dialog box, the Visual UPF appears, as shown in the example in [Figure 58](#).

The Visual UPF views that show your power intent are

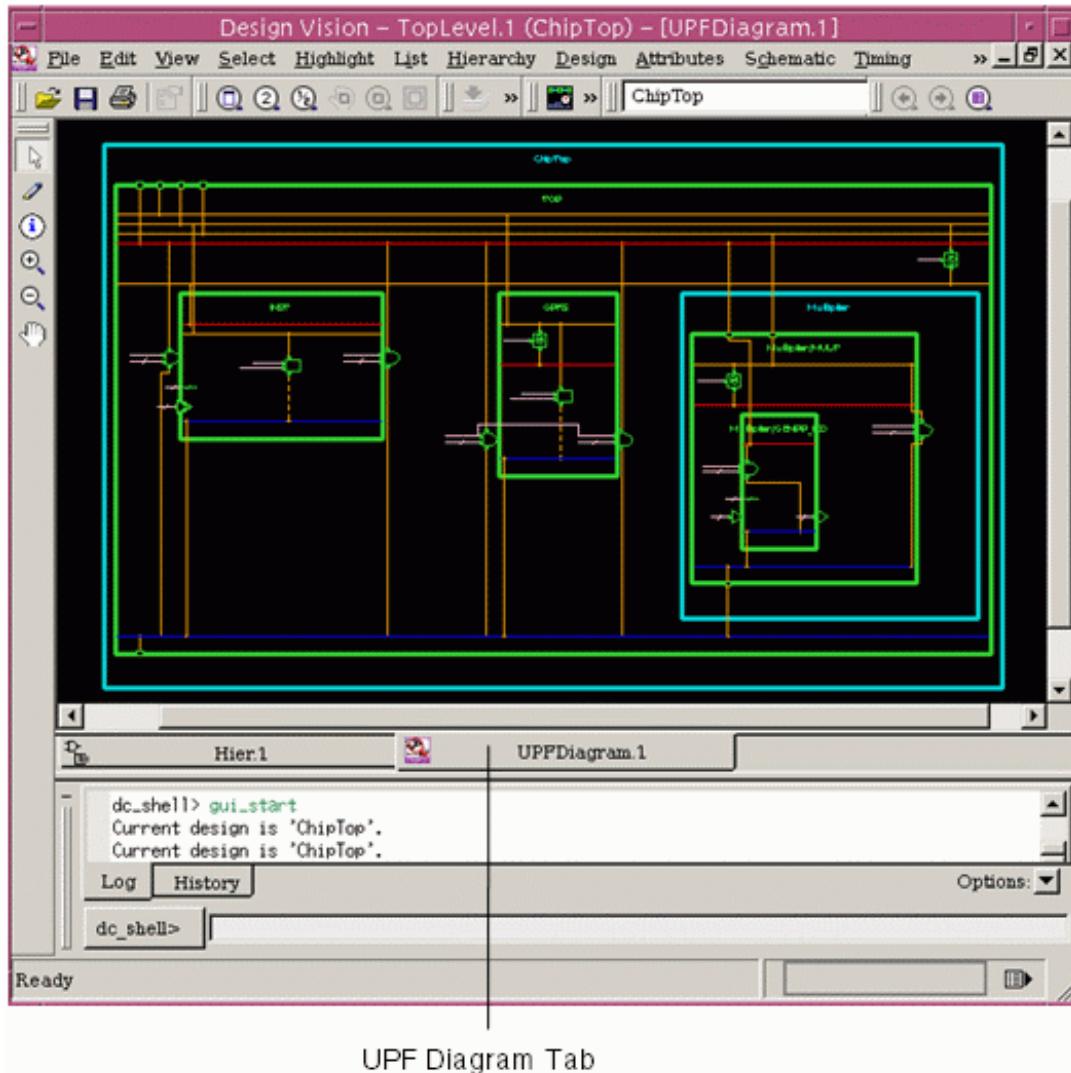
- Design or Logic Hierarchy View

Select the Design Hierarchy tab to view the logic hierarchy of your design, as shown in [Figure 58](#).

- UPF Diagram view

Select the UPF Diagram tab to view the pictorial representation of your power definitions as shown in [Figure 105](#).

Figure 105 UPF Diagram View in the Visual UPF Dialog Box

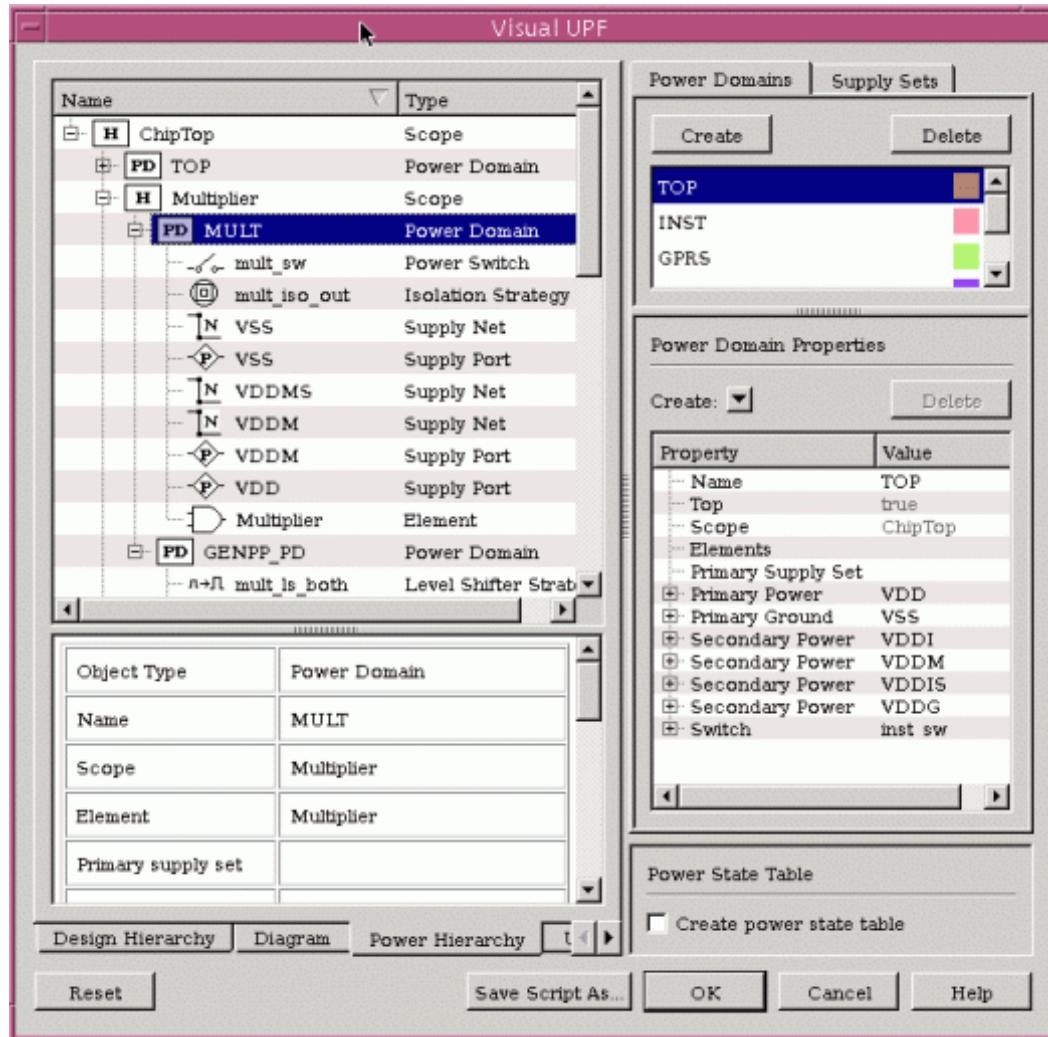


- Power Hierarchy view

Select the Power Hierarchy tab to see the power hierarchy of your design. [Figure 106](#) shows the Power Hierarchy view of a design.

The Power Hierarchy view has two sections. The section on the top shows the hierarchy tree with the connections between different power objects. The section at the bottom shows more details and properties of the object that you select in the top section.

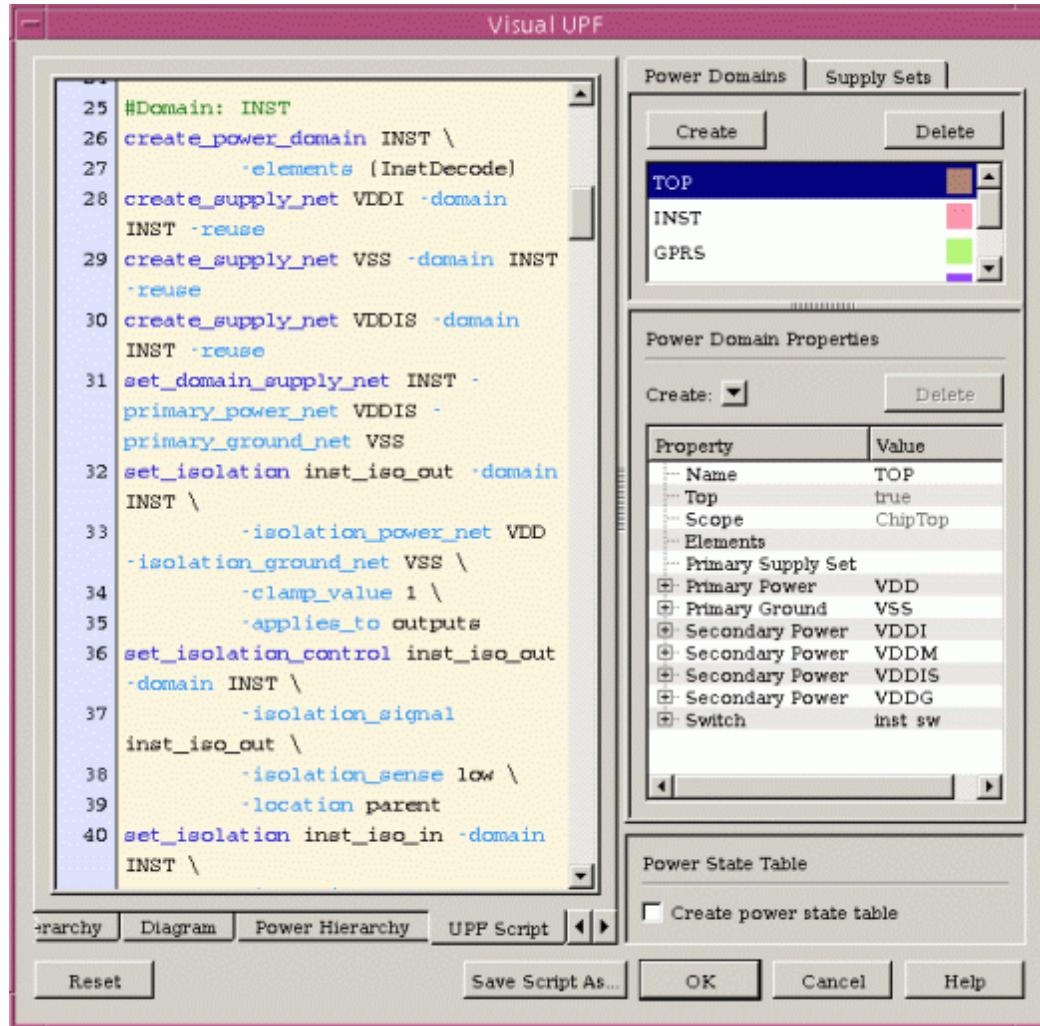
Figure 106 Power Hierarchy View in the Visual UPF Dialog Box



- UPF Script view

Use the UPF script tab to view the UPF script for your power definitions. Figure 107 shows the UPF Script view. The colors used in the script help to differentiate the UPF commands and the power objects.

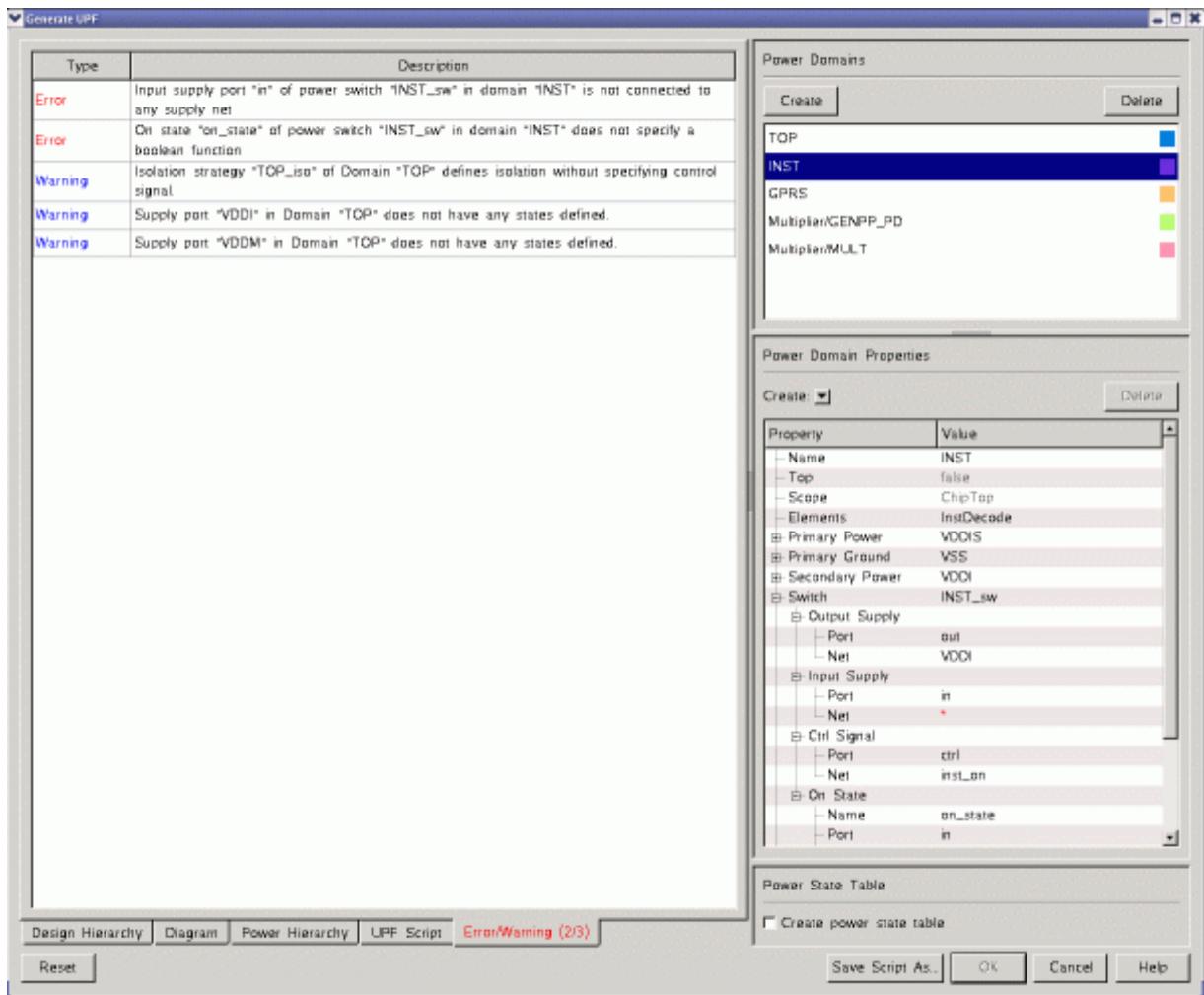
Figure 107 UPF Script View of the Visual UPF



- Error and Warning view

The Error/Warning tab in the Visual UPF view becomes active when your modifications cause errors or warnings, as shown in [Figure 108](#). When there are no errors or warnings, this tab is disabled. You can see the details of the error and warning messages in this view.

Figure 108 Error and Warning View in the Visual UPF Dialog Box



Applying Power Intent Changes

After defining or modifying the power intent, you can do one of the following:

- Save the power intent as a UPF script

Click the Save Script As button to save the power intent script in a file. The file is saved in ASCII format, as a UPF file, but the power intent is not applied to the design database of the tool. You can run this script either in the batch mode or interactively, to apply the power intent.

This feature can be useful when your changes are not yet complete, and you have to save it for a later use. It can also be useful when you have to edit the file before running it. For example, when you create a power state table, all the possible power states are populated in the table. Before running the script, you must edit the script to remove or comment the states that are not required.

- Apply the power intent to the design database

Click OK to apply the power intent in the design database. Until you click OK, power intent specifications are only contained in the Visual UPF dialog box and do not affect the design database.

Examining and Debugging UPF Specifications

The Power Compiler tool provides several commands and display features that analyze and report multivoltage aspects of the design. See the following topics for more information:

- [The analyze_mv_feasibility Command](#)

Use this command to report resolved isolation strategies and determine whether strategies can map successfully to library cells.

- [The check_mv_design Command](#)

Use this command to check for design errors that result in multivoltage constraint or rules violations.

- [The analyze_mv_design Command](#)

Use this command to report path-based details of a multivoltage design.

In addition, the `check_library` command in Library Compiler supports specific checks that are useful in the UPF Flow. For more details, see the Library Checking chapter in the *Library Quality Assurance System User Guide*.

The `analyze_mv_feasibility` Command

The `analyze_mv_feasibility` command can generate reports of resolved power management strategies and cell mapping feasibility. By default, the command generates both types of reports. However, you can specify which analysis to perform by using the following options:

- The `-resolved_strategy` option reports the resolved power management strategies of a design. A resolved strategy is the strategy that the tool selects for implementation based on precedence rules.
- The `-lib_cells` option reports whether the tool can map power management cells with the available libraries.

If you do not specify one of these options, or if you specify both options, the tool generates both types of reports.

For more information, see the following topics:

- [Reporting Resolved Strategies](#)
- [Reporting Cell Mapping Feasibility](#)

Reporting Resolved Strategies

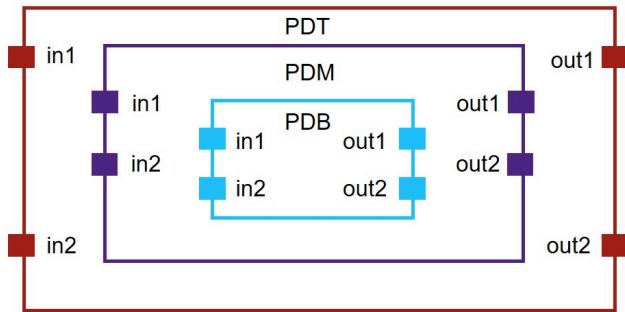
Use the `analyze_mv_feasibility` command with the `-resolved_strategy` option to analyze and report the resolved power management strategies of a design. A resolved strategy is the strategy that the tool selects for implementation based on precedence rules.

The `-resolved_strategy` option generates a text report that includes the following:

- All domain boundaries on which isolation, level-shifter, and repeater strategies are resolved
- All sequential elements on which retention strategies are resolved

For example, consider the design in [Figure 109](#). PDB, the bottom-level power domain, resides within PDM, the middle-level power domain, which is contained within PDT, the top-level power domain.

Figure 109 Example Power Domain



The UPF for this design is as follows:

```

create_power_domain PDT
create_power_domain PDM -elements {mid}
create_power_domain PDB -elements {mid-bot}
set_isolation iso1 -domain PDT -applies_to outputs \
    -applies_to_boundary both
set_isolation iso2 -domain PDM -applies_to inputs \
    -applies_to_boundary upper
set_isolation iso3 -domain PDM -applies_to both \
    -applies_to_boundary lower
set_isolation iso4 -domain PDB -applies_to outputs \
    -applies_to_boundary upper
set_level_shifter ls1 -domain PDT -elements {mid/in1} \
    -applies_to_boundary lower
set_level_shifter ls2 -domain PDM -elements {mid/in1}
set_repeater rptr1 -domain PDM -elements {mid/in2}
set_retention ret1 -domain PDB -elements {mid-bot/q_reg}

```

The default resolved strategy report is similar to the following:

```
dc_shell> analyze_mv_feasibility -resolved_strategy
```

Attributes that prevent power management cell insertion
 dt - dont_touch
 an - analog net
 pad - pad-port path
 pg - power/ground pin

Resolved isolation strategies:

Domain-boundary	HighConn (domain)	LowConn (domain)	Attributes
out1	--	isol1 (PDT)	pad
out2	--	isol1 (PDT)	--
mid/in1	isol1 (PDT)	iso2 (PDM)	--
mid/in2	isol1 (PDT)	iso2 (PDM)	--
mid/bot/in1	iso3 (PDM)	--	--
mid/bot/in2	iso3 (PDM)	--	--
mid/bot/out1	iso3 (PDM)	iso4 (PDB)	--
mid/bot/out2	iso3 (PDM)	iso4 (PDB)	--

Resolved Level Shifter Strategies:

Domain-boundary	HighConn (domain)	LowConn (domain)	Attributes
mid/in1	ls1 (PDT)	ls2 (PDM)	--

Resolved Repeater Strategies:

Domain-boundary	HighConn (domain)	LowConn (domain)	Attributes
mid/in2	--	rptr1 (PDM)	--

Resolved Retention Strategies:

Cell	Strategy (domain)	Attributes
mid/bot/q_reg	ret1 (PDB)	--

You can use additional options to restrict the scope of the report, as follows:

- The `-domain` option restricts the report to include only the cells and the boundary pins of the specified domain. You cannot use the `-domain` option with the `-lib_cells` option.
- The `-strategy` option restricts the report to include only the cells and boundary pins that have a given resolved strategy. The `-strategy` option must be used with the `-domain` option, and as a result, it cannot be used with the `-lib_cells` option.
- The `-elements` option restricts the report to include only the specified elements. You must provide the full hierarchical element names.
- The `-isolation`, `-enable_level_shifter`, `-repeater`, and `-retention` options restrict the report to include only the specified type of strategy.
- The `-disable_clubbing` option changes the report format to list elements in separate rows instead of the default format, which groups hierarchical cells or buses together.

In a design with heterogeneous fanout, a pin might be associated with more than one isolation strategy. In this case, the report contains an additional column that specifies the load on the pin, as shown in the following example:

```
>

Resolved isolation strategies:
-----
Domain-boundary Load      HighConn (domain)  LowConn (domain)  Attributes
-----
A/out          M1/B        --                  isol1 (PDT)       pad
A/out          M1/B2/C     --                  isol1 (PDT)       --
-----
```

Reporting Cell Mapping Feasibility

You can use the `analyze_mv_feasibility` command to analyze whether the Power Compiler tool can map power management cells with the available libraries. If any cells cannot be mapped, the tool generates a report that provides details about every element that cannot be mapped.

To generate a cell mapping report without also generating a resolved strategy report, use the `-lib_cells` option. The following options apply only to the resolved strategy report capability and therefore cannot be used with the `-lib_cells` option: `-domain`, `-strategy`, `-retention`, `-repeater`, and `-disable_clubbing`.

Cell mapping feasibility analysis is available for isolation cells by using the `-isolation` option and for enable level-shifter cells by using the `-enable_level_shifter` option. In the absence of these options, the tool analyzes both types of cells. The tool also reports level-shifter failure reasons when using `analyze_mv_feasibility` without any option.

The recommended procedure is to use the `analyze_mv_feasibility` command after you read the RTL and load the UPF, but before you perform synthesis with the `compile_ultra` command. If you use the `analyze_mv_feasibility` command after power management cell insertion, the tool performs the analysis based on the current status of the netlist. The analysis results might be different in these two use cases because optimizations performed during the synthesis operation might prevent specific library cells from being used.

If any power management cells cannot be mapped, the tool issues a UPF-909 error message and returns a Tcl status of 0. In addition, the tool provides a text report about cells that cannot be mapped. An example of the text report is as follows:

Isolation mapping failures:

Elements(s)	Strategy	Domain	Clamp	Sense	Reasons
mid_inst_1	PD1_ISO1	PD1	0		TLS mismatch (2)

TLS mismatch: Isolation library cell(s) cannot be used as they are not part of target_library_subset constraint.

Error: Not all the isolation and enable level shifter cells can be mapped (UPF-909)

0

Reporting Mapping Feasibility for Level-Shifters

Use the `analyze_mv_feasibility` command with the `-level_shifter` and the `-lib_cells` options to perform a mapping feasibility check for level-shifters and generate a global report of failure reasons on all the paths where level-shifter violation cannot be addressed. The global report, however, does not contain detailed information on the level-shifter insertion paths.

For a detailed path-based report, use the additional option `-elements` with a list of specific pin/net/cell names. This path-based report provides detailed information, including the fanout tree details and the level-shifter failure reasons, on the paths containing the pins/nets specified in the element list. Power state table (PST) information of the driver and load supplies is also reported.

Note:

When using `-elements`, use full hierarchical names of the specific pins, nets, or cells.

Examples

Global Report: Target Library has Level-Shifters

```
dc_shell> analyze_mv_feasibility -lib_cells -level_shifter
```

In this scenario, the preceding command generates a global report which provides information on all the paths where a level-shifter cannot be inserted. The failure reasons are shown in the *Failure reasons* column along with the corresponding driver pin name of the violating path in the *Path drivers* column.

```
*****
Report: analyze_mv_feasibility
Design: bottom
*****|
```

Index	Failure reasons	Path drivers
1	Main rail violation PVT mismatch Unsuitable map_level_shifter	Bot/reg/Q

Path-Based Report: Target Library has Level-Shifters

```
dc_shell> analyze_mv_feasibility -lib_cells -level_shifter \
-elements Bot/reg/Q
```

In this scenario, the target library has level-shifter cells. On running the preceding command, the tool is unable to insert level-shifters due to several reasons. The following path-based report provides a comprehensive picture on why the level-shifter insertion failed on each path, along with the fanout tree details.

Report: analyze_mv_feasibility

Design: bottom

Non-default app-option Info:

No non-default app-options to report

Global driver and load information:

Type	Pin Name	Voltage Range	Power Domain	Related Supplies	No. of fanout
Driver	Bot/reg/Q	0.95-0.95	PD_BOT	(VVDD, VSS)	1
Load	Mem/A [0]	0.95-0.95	PD_TOP	(VDD_RAM, VSS)	1

Driver and Load Related PST:

Supply Names: vdd_top, vdd_mid, vss_top, vss_mid,

Resulting Power States in this Scope:

vdd_top| vdd_mid| vss_top| vss_mid|

drv: ON| ON | ON| ON|

Global reasons:

None

Fanout Tree structure:

[0:Dr] -----[1:PD] -----[2:PD] -----[3:Ld]

Dr: Driver

PD: Power domain boundary

Ld: Load

Fanout Tree details:

Constraints

dtn - don't touch net

dtc - don't touch cell

so - size only cell

ID	Pin	Net	Constraints	Domain	No. of available supplies	Failure reasons	No. of usable library cells
0	Bot/reg/Q	Bot/byte	-	PD_BOT	5	Core logic pin	0
1	Bot/A	Bot/byte	-	PD_BOT	5	Main rail violation, PVT mismatch	0
2	Bot/A	byte	LS1	PD_TOP	5	Unsuitable map_level_shifter	12
3	Mem/A [0]	byte	-	PD_TOP	5	Core Logic pin	0

For a description of the table columns in the report, see the following table:

Column	Description
ID	ID of the pin name as per the fanout tree structure
PIN	Pin name
Net	Net name connected to the pin
Constraints	Constraints set on the pin or net. This includes dont_touch on net or cell, size only cell, and level-shifter strategy name.
Domain	Domain name of the net segment
No. of available supplies	Count of available supplies
Failure reasons	Local failure reasons
No. of usable library cell	Number of library cell that could have been used

Generating HTML Cell Mapping Reports

The Power Compiler tool optionally provides detailed information about unmapped power management cells in HTML reports. HTML reporting is off by default. You can enable this feature as follows:

- To generate HTML reports at the `insert_mv_cells`, `insert_dft`, and `compile_ultra` commands, set the `upf_generate_pm_cell_html` variable to `true` (the default is `false`).
- To generate HTML reports at the `analyze_mv_feasibility` command, use the `-format html` option. The tool honors this option regardless of the setting of the `upf_generate_pm_cell_html` variable.

When HTML reporting is enabled and the design has unmapped power management cells, the tool creates a directory named `pm_cells_map_failure.x` in the run directory. The directory name `x` is an integer index that starts with 0 for the first report and increments by 1 for each additional report. Each execution of any of the listed commands generates a new report, which results in a new directory for that report.

In each `pm_cells_map_failure.x` directory, the tool writes HTML files named `index.html`, `iso.html`, and `els.html`. The `index.html` file is a top-level summary page with links to the detail HTML files. All commands generate detail files for isolation cell mapping failures (`iso.html`) and enable level-shifter cell mapping failures (`els.html`). The `compile_ultra` and `insert_dft` commands also generate a detail file for retention cell mapping failures (`retention.html`).

Figure 110 is an example of the top-level summary page and **Figure 111** is an example of the isolation cell summary page.

Figure 110 Top-Level HTML Summary Page

Power Management cells map failure summary

[Isolation cell map failures](#)
[Enable Level Shifter cell map failures](#)
[Retention cell map failures](#)

Sun Sep 27 06:11:50 2020

Figure 111 Isolation Cell Summary Page

Isolation cell map failure reasons

	Element(s)	Cell name(s)	Strategy	Domain	Clamp	Iso sense	Additional properties	Failure reasons
1	mid_inst_1/d (bus)	Cell name(s)	PD1_ISO1	PD1	0	high	Properties	target_library_subset_mismatch (2 lib cells)
2	mid_inst_1/clk	Cell name(s)	PD1_ISO1	PD1	0	high	Properties	target_library_subset_mismatch (2 lib cells)
3	mid_inst_1/bot_inst_2 (cell)	Cell name(s)	PD1_ISO1	PD1	0	high	Properties	target_library_subset_mismatch (2 lib cells)

Sun Sep 27 06:11:45 2020

Many entries in the table are links to additional information. For example, **Figure 112** shows expanded information about failure reasons.

Figure 112 Isolation Cell Detail Page

Isolation cell map failure reasons

Element(s)	Cell name(s)	Strategy	Domain	Clamp	Iso sense	Additional properties	Failure reasons			
1 mid_inst_1/d (bus)	Cell name(s)	PD1_ISO1	PD1	0	high	Properties <ul style="list-style-type: none"> ◦ Supply of the domain where isolation cell resides (power, ground, N-well, P-well): (PD1.primary.power, PD1.primary.ground, (nil), (nil)) ◦ Isolation cell strategy supply (power, ground, N-well, P-well): (PD1.primary.power, PD1.primary.ground, (nil), (nil)) ◦ Isolation cell driver supply (power, ground): (PD1.primary.power, PD1.primary.ground) ◦ Isolation cell load supply (power, ground): (PD1.primary.power, PD1.primary.ground) ◦ Isolation strategy location: self ◦ PVT: (P = 1.00, V (primary, backup): (1.20V, 1.20V), T = 25.00) ◦ Scaling library group: Not defined ◦ Bias: disabled ◦ target_library_subset = -dont_use { TG3AND2XCLPC } ◦ link_library_subset = (nil) 	target_library_subset_mismatch (2 lib cells) Following library cells cannot be used as they violate target_library_subset constraint <table border="1" style="margin-left: 20px;"> <tr><td>Isolation library cells</td></tr> <tr><td>Isolation_TYP_mvt_1.2V_u/TG3AND2XCLPC</td></tr> <tr><td>Isolation_TYP_mvt_0.74V_u/TG3AND2XCLPC</td></tr> </table>	Isolation library cells	Isolation_TYP_mvt_1.2V_u/TG3AND2XCLPC	Isolation_TYP_mvt_0.74V_u/TG3AND2XCLPC
Isolation library cells										
Isolation_TYP_mvt_1.2V_u/TG3AND2XCLPC										
Isolation_TYP_mvt_0.74V_u/TG3AND2XCLPC										
2 mid_inst_1/clk	Cell name(s)	PD1_ISO1	PD1	0	high	Properties	target_library_subset_mismatch (2 lib cells)			
3 mid_inst_1/bot_inst_2 (cell)	Cell name(s)	PD1_ISO1	PD1	0	high	Properties	target_library_subset_mismatch (2 lib cells)			

Sun Sep 27 06:11:45 2020

The `check_mv_design` Command

To check for design errors, including multivoltage constraint violations, electrical isolation violations, connection rules violations, and operating condition mismatches, use the `check_mv_design` command.

Multivoltage Design Violations in the GUI

The MV Advisor violation browser provides a visual analysis and debugging environment for multivoltage design violations. You can check the design for issues such as multivoltage constraint violations, electrical isolation violations, connection rule violations, and operating condition mismatches. After checking a design, you can use the violation browser to examine the violation report.

To open the MV Advisor violation browser, choose Power > MV Advisor.

The violation browser automatically loads the current violation report if a valid report is available for the current state of the design. If a valid report does not exist, the violation browser provides links that you can use to load a saved report or generate a new report, as shown in [Figure 113](#).

Figure 113 Violation Report When a Valid Report is Not Available



The violation browser groups the violations based on specific properties, displays detailed information about the violations, and guidance for investigating and fixing them. When you select a violation, the violation browser displays details such as an explanation of the warning or error message and suggestions for fixing the violation.

The violation browser also provides access to context-aware reports and other analysis tools. You can perform the following actions:

- Select pin names and view information about the pins
- Display man pages (in the man page viewer) for warning and error messages
- Visually inspect a violation by displaying it in a schematic view

You can display the report for an individual violation in a new instance of the Design Vision window that serves as a debugging work environment.

You can check the design for violations before or after you open the violation browser. To check the design before opening the violation browser, use the `check_mv_design` command. When the violation browser is open, you can use the Check MV Design dialog box to check the design.

Generating Design Violation Reports

You can analyze multivoltage design problems by checking the design for errors and generating a violation report that you can view in the console log view and save in a file. You can check the design at any time by using the `check_mv_design` command. For more information, see the man page. When the MV Advisor violation browser is open, you can check the design by using the Check MV Design dialog box.

When you check the design, the tool creates or updates the current violation report by default. If you do not specify a file name, the tool stores the current report information in a temporary file until the end of the current session. If you specify a file name, the tool saves the report in an XML file and also creates an XSLT file with the same name with a .xslt extension. The XSLT file contains auxiliary information that is required when you view the violation report in a Web browser.

The MV Advisor violation browser supports the violations listed in [Table 35](#). The `check_mv_design -output` command identifies and reports several other issues and these are not supported by the GUI for reporting and fixing details.

Table 35 Violations Supported by MV Advisor

LIBSETUP-001	LIBSETUP-001a	LIBSETUP-001b	MV-038	MV-044	MV-076	MV-078
MV-166	MV-168	MV-168b	MV-231	MV-231a	MV-231b	MV-232c
MV-232l	MV-237	MV-252	MV-513	MV-514	MV-514a	MV-514b
MV-516	MV-529	MV-534	MV-545	UPF-067	UPF-103	

Violation Groups

The violation browser groups the messages based on the source domain-sink domain pair in the first level and the driver pin in the next level, as shown in [Table 36](#).

Table 36 Messages Grouped by Source Domain and Sink Domain Pair

Message ID	Description
MV-231 MV-231a MV-231b	Level-shifter violations
MV-237	Paths with voltage violations

Table 36 *Messages Grouped by Source Domain and Sink Domain Pair (Continued)*

Message ID	Description
MV-252	Missing level-shifting violations specified with the <code>-no_shift</code> option
MV-513	Redundant isolation
MV-514 MV-514a MV-514b	Missing isolation violations
MV-545	Missing isolation violations with the <code>-no_isolation</code> option

The violation browser groups the messages shown in [Table 37](#) based on the power domains where the cells or nets are located.

Table 37 *Messages Grouped by Power Domain of the Violating Cells or Nets*

Message ID	Description
LIBSETUP-001 LIBSETUP-001a LIBSETUP-001b	Cells with mismatched operating conditions
MV-044	Isolation cells used as a core cell
MV-076	Always-on nets driven by a normal cell
MV-078	Always-on cells driving a normal net
MV-166	Retention cells without a strategy
MV-168, MV-168b	Isolation cells without a strategy
MV-232c MV-232l	Level-shifter with violations
MV-516	Back-to-back isolation cells violation
MV-529	Unused power management cells
UPF-067	Undetermined PG pin connection

In addition, the violation browser groups messages as follows:

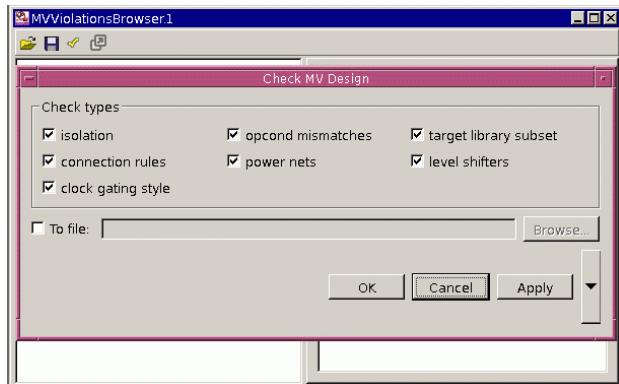
- MV-038 warning messages appear under one title and without multiple levels of groups.
- MV-534 messages are based on the driver pins.

- MV-232c messages are based on the power domains in the first level and the power supplies in the next level.
- MV-516 messages appear under one title and without multiple levels of groups.
- UPF-103 messages are based on the ignored strategy.

Use the following steps to generate or update the current violation report:

1. Click the  button to run the `check_mv_design` command and load the report in the MV Advisor violation browser. The Check MV Design dialog box appears as shown in [Figure 114](#).
2. Select or deselect the check type options as needed, to perform the required checks. The tool performs all the checks by default.
3. Select the “To file” option.
4. (Optional) Specify a file name, to save the report in a file.
5. Click OK or Apply.

Figure 114 Check MV Design Dialog Box



The Web browser report groups the violations in the same way that the MV Advisor violation browser groups them. However, the Web browser displays only the violation list, the information in the violation category tree in the violation browser. It does not display the detailed report information for each violation that the violation browser displays.

To open a violation report in your Web browser, open the XML file in a browser window. The XSLT file must be present in the same directory.

Examining Design Violations in the MV Advisor Violation Browser

The MV Advisor violation browser provides a visual analysis and debugging environment for design violations in a multivoltage design. You can search for and view information about several types of multivoltage design violations. When you select a violation, the

violation browser displays details about the violation and guidance about how to proceed in investigating the violation.

The violation browser view window consists of a button bar at the top and two panes: a violation category tree on the left and a report view on the right.

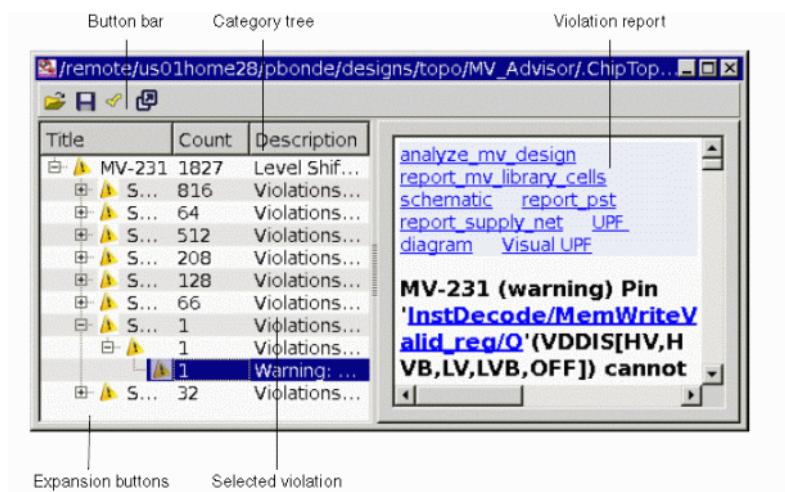
- The violation category tree groups the violations into types, categories, and subcategories.

You can use the expansion buttons in the category tree to expand or collapse individual types, categories, and subcategories.

- The report view displays information about the type, category, or violation that you select in the category tree.

[Figure 115](#) shows an example of the MV Advisor Violation Browser with a violation selected in the category tree and the violation report in the report view.

Figure 115 The MV Advisor Violation Browser



The buttons on the button bar at the top of the view window allow you to

- Open another violation report
- Save the violation report you are viewing
- Generate or update the current violation report
- Display the report for the selected violation in a new Design Vision window, where you can use other analysis tools to debug the violation

To help you to evaluate the status of violations through the design flow or to compare the reports from different design checks, you can open more than one report at the same time. When you open a report file, the violation browser compares the design name and the

number of cells with the netlist in the current design, and displays a message if it finds any inconsistencies.

Note:

The Design Vision GUI supports detailed reporting of how to debug and fix violations for a subset of the issues identified and reported by the `check_mv_design -output` command. For information about the checks supported in the MV Advisor violation browser, see [Generating Design Violation Reports](#).

Exploring the Violations

To analyze and debug a violation, you can open the report on the Browser panel in a new Design Vision window. By default, the Browser panel is attached to the left side of the window. You can use the workspace area in this window to debug the violation by clicking links at the top of the violation report to generate and display other reports and open analysis views such as a schematic or UPF diagram view.

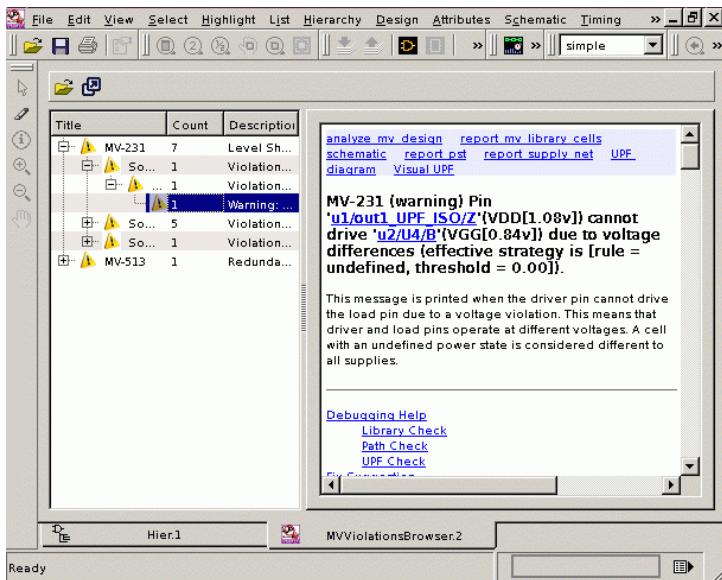
To explore the violation categories and view the violations within each category, do the following:

1. Expand violation categories, showing the subcategories or violations at the next level in the category tree.

To expand or collapse a violation category, double-click its line in the category tree or click its expansion button in the Type column.

As shown in [Figure 116](#), a plus sign on the expansion button means the category is collapsed. A minus sign on the expansion button means the category is expanded.

Figure 116 Detailed Report in the Report View



2. Select a violation type, category, or subcategory tree to display information about it in the report view.

When you select a violation type, the report view displays the generic violation message and the number of violations of that type found in the design as shown in [Figure 116](#).

When you select a violation category or subcategory, the report view displays the generic violation message, the number of violations in the category, and the location of the violations.

3. Select a violation in the category tree to display a detailed report about it in report view, as shown in [Figure 116](#).

When you select a violation, the report view displays the warning or error message, a brief explanation of the message, and a detailed description of the violation that includes debugging information and suggestions for fixing the violation.

The links at the top of the report view allow you to run report commands and other features of the tool.

- To run the `analyze_mv_design` command and display the level shifter report in a report view, click the `analyze_mv_design` link.
- To run the `report_mv_library_cells` command and display the library cells report in a report view, click the `report_mv_library_cells` link.

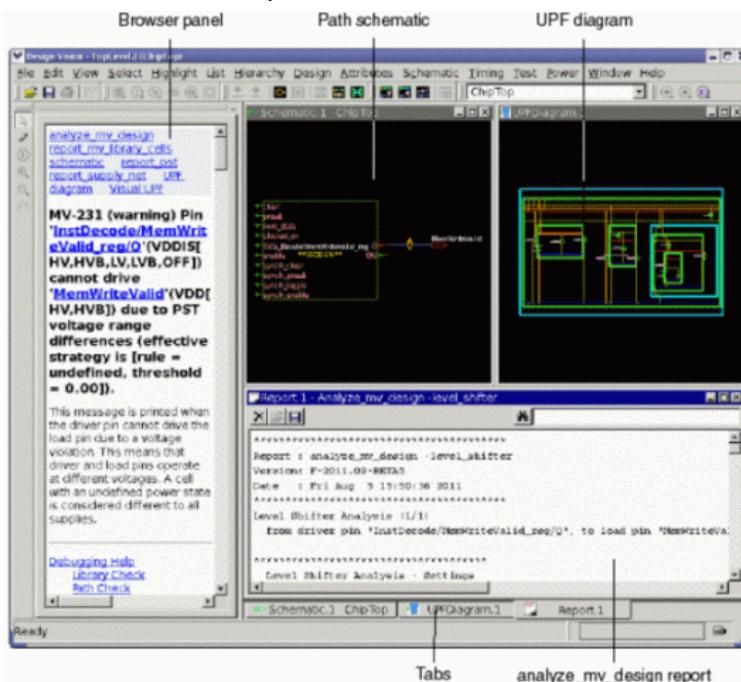
- To run the `report_pst` command and display the power state table report in a report view, click the `report_pst` link.
- To run the `report_supply_net` command and display the supply net report in a report view, click the `report_supply_net` link.
- To view the violation in a schematic, click the schematic link.
- To open a UPF diagram view, click the UPF Diagram link.
- To open the Visual UPF dialog box, click the Visual UPF link.

In the warning or error message, click the pin name to select the pin. In the detailed report, you can click the links to run commands and access useful features of the tool.

You can use the arrow keys on the keyboard to scroll through the report view. To scroll up or down, press the Up Arrow key or the Down Arrow key. To move to the top or bottom of the view, press the Page Up key or the Page Down key.

[Figure 117](#) shows an example of the debugging environment provided by the Browser panel in a new Design Vision window. For example, after displaying the violation report on the Browser panel, you can click links at the top of the report to display the violation in a schematic, open the UPF diagram view, and see the reports in report views.

Figure 117 Violation Report on Browser Panel in a New Window

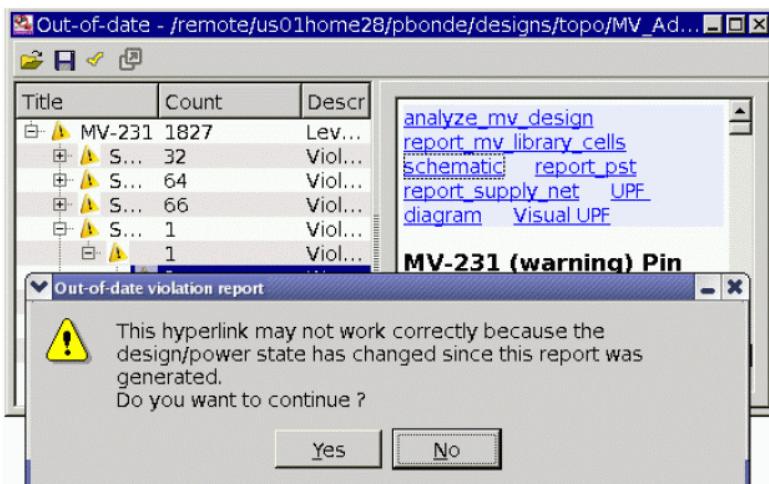


The tool maintains a single, current repository for the multivoltage design violations that you can view in the MV Advisor violation browser. If you update the violation report that you are viewing, the tool automatically refreshes the report information in the violation browser.

If you view an out-of-date report in the violation browser, or if a change that you make in the design invalidates the report that you have open in a violation browser window, the term “Out-of-date” appears in the window title bar. In addition, the violation browser restricts hyperlinks in an out-of-date report when a link action might update the design or manipulate design objects. If you click a restricted hyperlink, the tool displays a warning and prompts you to continue or cancel the link action, as shown in Figure 118.

Reports, schematics, the UPF diagram, and the Visual UPF dialog box all work with the up-to-date design. Commands that operate directly on an element reported in a violation, such as the `report_net` command, can cause an error if the element no longer exists in the design due to a previous action. Hyperlinks that are not restricted include internal HTML jumps, man page links, and links to preview commands.

Figure 118 Out-Of-Date Violation Report Warning



The `analyze_mv_design` Command

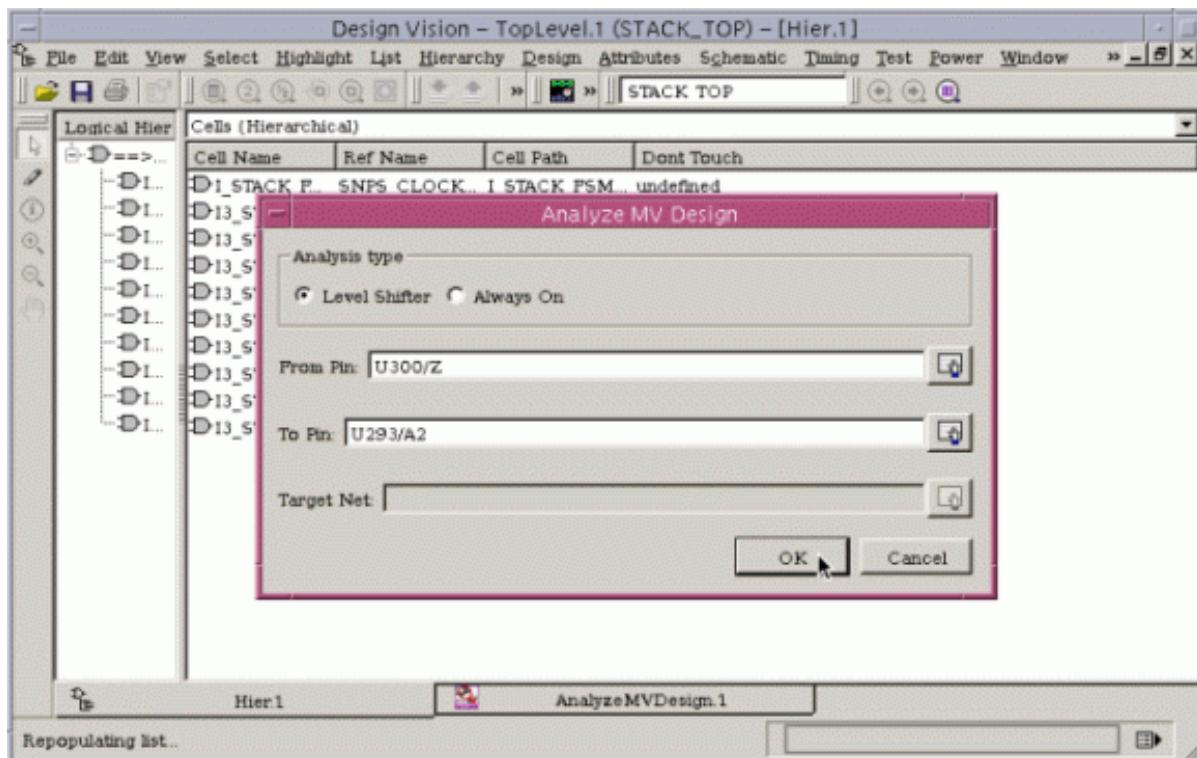
The `analyze_mv_design` command reports path-based design details of a multivoltage design that can be useful in debugging multivoltage design issues. The report contains details of the variable settings for level-shifter insertion and always-on buffering, relevant power state tables, the driver-to-load pin connections, the pin-to-pin information on specified paths, the target libraries used for insertion of power management cells, and other useful debugging information. You can also run this command in the GUI and see the issues in the schematic. For details, see [Analyzing Multivoltage Design Connections in the GUI](#).

Analyzing Multivoltage Design Connections in the GUI

You can use the Analyze MV Design dialog box to analyze your design for multivoltage-specific connectivity issues. The `analyze_mv_design` command runs internally and displays the result in a new view.

To open the Analyze MV Design view, choose Power > Debugging > Analyze MV Design. The Analyze MV Design dialog box appears as shown in [Figure 119](#).

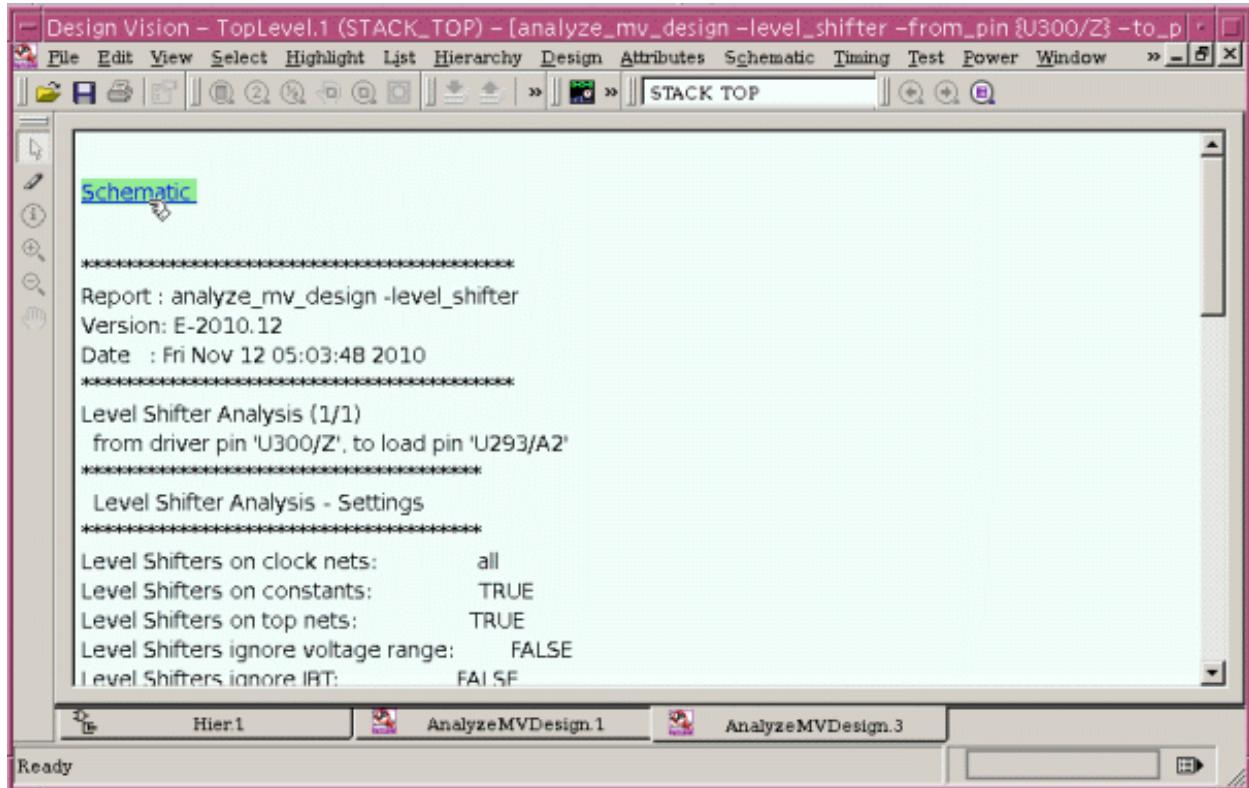
Figure 119 Analyze MV Design Window



Use the dialog box to choose the type of analysis to perform, either level shifter or always on. You can also specify the From Pin and the To Pin, where the checks have to be performed. When you click OK, the tool runs the `analyze_mv_design` command.

The report of the `analyze_mv_design` command is displayed in a new view, as shown in [Figure 120](#). The report contains details of level-shifter violations.

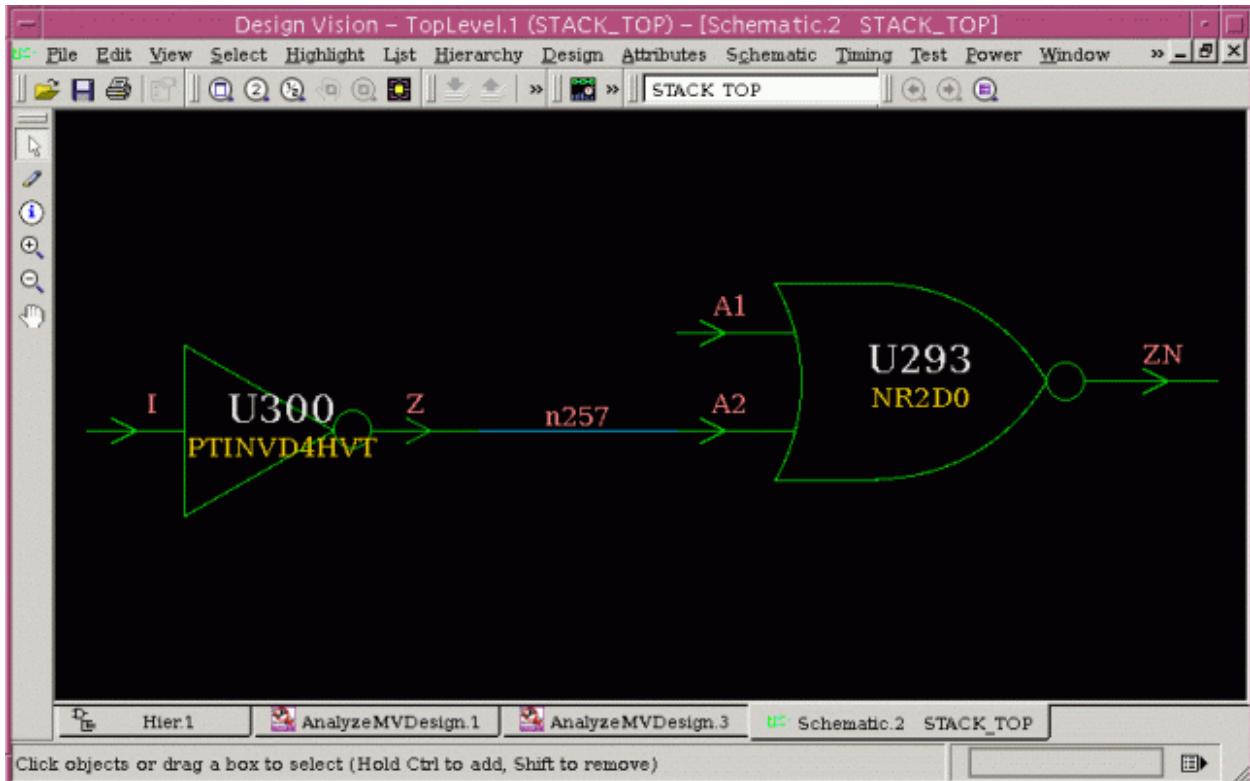
Figure 120 Report View of the Analyze MV Design Window



The report also contains a hyperlink to the schematic; when you follow the link, the schematic shows design objects that are specific to the reported issue, as shown in Figure 121. In the schematic, you can

- Create a collection of the power supply nets connected to one or more pins
- View a list of the ground supply net connections for one or more pins
- View a report of power pin information for one or more cells
- View a report of PG pin library information for one or more cells

Figure 121 Schematic View of Analyze MV Design Window



Writing the Power Information

The power information updated during synthesis can be written out with the `save_upf` command. This UPF file written by Design Compiler is referred to as the UPF' file to distinguish it from the UPF file that you use to synthesize the design. The UPF' file is used as input to downstream tools such as the IC Compiler II, PrimeTime, PrimePower, and Formality tools.

The additional information in the UPF' file are as follows:

- A comment on the first line, as shown in the following example:

```
#Generated by Design Compiler(E-2010.12) on Thu Oct 28 14:26:58 2010
```
- Explicit power connections to special cells such as level shifters and dual supply cells
- Additional UPF commands specified at the command prompt in the Design Compiler session

If you specify UPF commands at the command prompt, along with the RTL file, update the UPF file with these commands. This update is required for Formality to verify the design successfully.

The UPF standard requires a simple name for the argument of certain UPF commands. By default, the tool checks this requirement. To remove this requirement in the UPF file written by the `save_upf` command, set the `mv_output_enforce_simple_names` variable to `false`.

Preserving the Command Order in the UPF' File

To improve the clarity of the UPF' file, the tool

- Writes the user-specified UPF commands and tool inserted UPF commands in separate sections
- Lists the commands in the user-specified section in the order they were specified

To distinguish the user-specified UPF command section from the tool-generated UPF command section, the sections are separated by the `derived_upf` variable setting.

The beginning of the tool-generated section is marked by the following setting:

```
set derived_upf true
#Design Compiler added commands
```

The end of the tool-generated section is marked by the following variable setting:

```
set derived_upf false
```

Do not explicitly set the `derived_upf` variable to either `true` or `false`. Use of this variable is restricted to the tool.

If the RTL design contains PG information, the UPF commands generated by the `convert_pg` command are listed in a separate section, marked by the following comments:

```
# Commands created by "convert_pg"
...
# End of commands created by "convert_pg"
```

When a UPF' file is read into the Power Compiler tool and another UPF' file is written, the command order is preserved in the newer UPF' file. The file contains the user-specified UPF command and the tool generated UPF commands in separate sections.

Note:

This feature is not supported in a hierarchical flow. If you use this feature in the hierarchical flow, the tool issues the `UPF-401` information message.

[Example 43](#) shows the UPF' file written out by the Power Compiler tool.

Example 43 UPF' File

```
#Generated by Design Compiler

create_power_domain PDT
create_supply_net SN1 -domain PDT
create_supply_net SN2 -domain PDT

create_power_domain PDC -elements {ABC}
create_supply_net SN3 -domain PDC

set derived_upf true
#Design Compiler added commands
connect_supply_net SN1 -ports {PORT1}
connect_supply_net SN3 -ports {PORT2}
set derived_upf false
```

Controlling the Line Width in the UPF' File

When the `save_upf` command writes the commands into an output file, by default, commands with several arguments are not split into multiple lines. To simplify the format of the output file for ease of use, you can control the line width of the file by using the following variables:

- `mv_output_upf_line_width`

Use this variable to control the line width in the UPF file written by the `save_upf` command. The default of this variable is 0, indicating that the `save_upf` command does not split long lines over multiple lines.

Set this variable to a positive integer to specify the width of each line. Lines that are longer than the specified value are split and written on multiple lines. When a line is split over multiple lines, the end of each incomplete line that continues on the next line is marked with a backslash character (\), as shown in [Example 44](#).

When the line to be split does not have a space up to the specified limit, the tool allows the line to exceed the limit; the line is split at the first space after the specified limit.

- `mv_output_upf_line_indent`

Use this variable to specify the number of spaces to indent at the beginning of each line continuation. The default of this variable is 2 which means, the `save_upf` command indents 2 spaces at the beginning of each continuing line.

Example 44 shows the UPF file written by the `save_upf` command, when the `mv_output_upf_line_width` variable is set to 30 and the `mv_output_upf_line_indent` variable is set to 3.

Example 44 UPF' File Written With Line-Splitting Enabled

```
set mv_output_upf_line_width 30
set mv_output_upf_line_indent 3
set_port_attributes -ports \
    {instA/power_port} \
    -attribute snps_derived
```

Note:

The tool issues error messages if you set any of these variables to a value less than 0.

Writing and Reading Verilog Netlists With Power and Ground Information

The Power Compiler tool supports the writing and reading of Verilog netlists containing the complete power and ground (PG) supply connection information, including supply connections to the leaf-level library cells.

To write out a Verilog netlist with the complete PG supply connection information, use the `-pg` option of the `write_file` command. For example,

```
dc_shell> write_file -format verilog -pg output top_with_pg.v
```

When you read in a Verilog netlist with the complete PG supply connection information using the `read_verilog -netlist` command, the tool automatically recognizes and restores the PG supply connection information stored in the netlist. If there are any conflicts between the Verilog netlist and the `connect_supply_net` commands in the UPF file, the tool reports the differences as errors.

Power and Ground Supply Connection Syntax

When you specify the power and ground connections in a Verilog module, the Verilog netlist, PG supply connections use the same syntax as the logic signal connections. For example,

```
module test ( A, Z, VDD, VSS );
    input A, Z;
    input VDD;
    input VSS;
    ...

```

When power and ground nets need to cross over levels of the Verilog hierarchy that do not have corresponding crossovers in the UPF hierarchy, the tool punches ports to carry the PG nets into the lower levels of the hierarchy, as in the following example.

```
// VDDINT is not used in any leaf cells in 'top'
module top ( A, Z, VDD, VSS, VDDINT );
    input A;
    input VDD;
    input VSS;
    input VDDINT;
    output Z;
    ...
    mid M1 ( .A( A ), .Z( Z ), .VDDINT( VDDINT ), .VSS( VSS ) );
module mid ( A, Z, VDDINT, VSS );
    ...

```

The tool does not punch ports at the top level. If a supply net drives a cell at the top level, but the net is not explicitly connected to a supply port or a supply set function defined in the UPF file, the supply inputs to those cells are undriven wires.

Each module net has the same name as the UPF supply net corresponding to the PG supply net. Because the names of the supply net does not need to match the name of the supply port, the tool uses the following syntax for the supply net:

```
// corresponds to UPF statement:
// connect_supply_net VDDINT -ports VDD
module test ( A, Z, .VDD( VDDINT ), VSS );
```

Each instance of a cell with PG pins shows the PG connections for the instance, including the pin name and the name of the supply net connected to the pin. The supply pins appear last, after the signal pins, as shown in the following example:

```
...
SIMPLE_PG_CELL U1 (.a(A), .z(Z), .VDD(VDD0), .VSS(VSS));

COMPLEX_PG_CELL U1 (.a(A), .z(Z), .VDD(VDD0),
                     .VDDBACKUP(SNPS_ss_SS_P1$primary$power),
                     .VSS(VSS));
...
...
```

Supply Sets

If the UPF file uses supply sets or supply set handles, and these supplies are not resolved to supply nets, the tool writes out the supply set definitions in the Verilog netlist. The supply set definitions contain the `SNPS_ss_` prefix and use the dollar character \$ in place of each period character that delimits the fields within a supply set identifier. For example,

```
module test ( A, Z, .VDD1( SNPS_ss_ss1$power ),
              .VDD2( SNPS_ss_pd1$primary$power ),
              .VSS( SNPS_ss_pd1$primary$ground ) );
    input A, SNPS_ss_ss1$power,
          SNPS_ss_pd1$primary$power,
          SNPS_ss_pd1$primary$ground;
    output Z;
    wire VDDS; // switched supply
    ...

```

Power Switches

The Power Compiler tool does not instantiate power switch cells. However, power switches have output supply nets that can power other cells. The tool writes out these supply nets without drivers. For example,

```
module top(a, z, VDDA, VDDB, VSS);
    input a;
    input VDDA;
    input VDDB;
    input VSS;
    output z;

    mid M1(.a(a), .z(z), .VDDA(VDDA), .VDDB(VDDB), .VSS(VSS));
    ...
    ...

Module mid(a, z, VDDA, VDDB, VSS);
    input a;
    input VDDA;
    input VDDB;
```

```

input VSS;
wire VDD_SW;
wire a1;

SIMPLE_PG_CELL U1 (.a(a), .z(a1), .VDD(VDD_SW), .VSS(VSS));
// The ports in the bot design are punched as follows:
bot B1 (.a(a1), .z(z), .VDDA(VDDA), .VDBB(VDBB), .VSS(VSS), \
.VDD_SW(VDD_SW));
...

```

The tool does not write out the PG nets and ports that do not drive any PG pins of leaf-level cells. The power switch behavior is derived from the UPF description provided with the PG netlist.

Reading Verilog Netlists With Power and Ground Supply Connections

The Power Compiler tool requires PG connections to be represented in UPF format before processing the design. When the tool reads in a Verilog netlist containing power and ground information, it matches the PG supply connections in the netlist to the UPF supplies and converts these connections into UPF commands. The tool issues error messages if it finds any conflicts.

The Verilog netlist must satisfy the following requirements:

- The netlist must contain a complete set of PG connections for all cells in the design, including standard cells.
- The netlist must be created by a tool with a valid UPF infrastructure.
- The PG connections must be intact as written by the tool, and not modified by the user.

Meeting these requirements ensures that every PG connection in the input netlist corresponds to a UPF supply net as specified in the UPF files, and no new supply nets, supply ports, or supply connections are needed.

The Formality and Verdi NLP tools use the PG Verilog netlist alone to represent the PG connections of the design. These tools do not need to match the PG netlist to UPF supplies, although they might extract other power infrastructure information from the UPF files.

Specifying Design Instances Using SystemVerilog Elements

When using the `-elements` option with certain UPF commands, you can reference SystemVerilog vector and structure elements using their RTL vector or structure name. To enable this feature, run the following command:

```
set_upf_query_options -bus_struct_mode true
```

Setting the `-bus_struct_mode` option affects the `set_retention`, `set_isolation`, `set_level_shifter`, and `map_retention_cell` commands. To use this feature, you must have the following variables set before reading the UPF:

- `bus_naming_style = "%s[%d]" (default)`
- `bus_dimension_separator_style = "][" (default)`
- `hdlin_enable_upf_compatible_naming true`

This option should only be enabled when loading the initial RTL UPF and then should be disabled after reading the RTL, as shown:

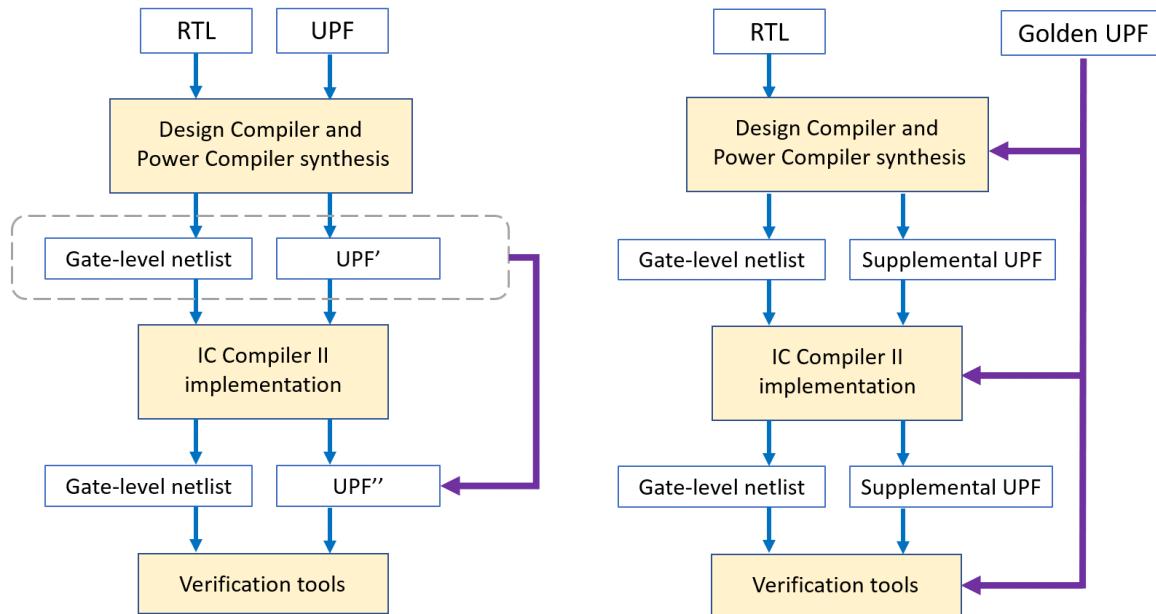
```
set_upf_query_options -bus_struct_mode true
load_upf block1.upf
set_upf_query_options -bus_struct_mode false
```

Note that the `-bus_struct_mode` option should be `false` (the default) except when reading in the RTL UPF before any netlist editing happens. Otherwise, any vector or structure references might return incorrect results.

The Golden UPF Flow

The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Design Compiler and IC Compiler II tools. [Figure 122](#) compares the traditional UPF flow with the golden UPF flow.

Figure 122 UPF-Prime (Traditional) and Golden UPF Flows



The golden UPF flow maintains and uses the original “golden” UPF file unchanged throughout the flow. The Design Compiler and IC Compiler II tools write power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF’ or UPF” file.

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.
- You can optionally use the Verilog netlist to store all the PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

To use the golden UPF flow, you must enable it by setting the following variable:

```
dc_shell> set_app_var enable_golden_upf true
```

After you enable this mode, to execute any UPF commands other than query commands, you must put the commands into a script and execute them using the `load_upf` command. You cannot run the commands individually on the command line or by using the `source` command.

For more information about using the golden UPF mode, see [SolvNetPlus article 1412864, "Golden UPF Flow Application Note."](#)

Reporting Commands for the UPF Flow

The following reporting commands are supported in the Power Compiler tool. These are not UPF standard commands.

- `report_dont_touch`
- `report_power_domain`
- `report_level_shifter`
- `report_power_switch`
- `report_pst`
- `report_isolation_cell`
- `report_retention_cell`
- `report_supply_net`
- `report_supply_port`
- `report_target_library_subset`
- `report_mv_library_cells`

For more details, see the command man pages.

UPF-Based Hierarchical Multivoltage Flow Methodology

Design Compiler topographical mode supports flat, top-down, and bottom-up hierarchical UPF design flows. These flows are also supported by Synopsys tools such as the IC Compiler II, PrimeTime, and Formality tools. This section describes the UPF portion of the hierarchical design methodology. For basic information about the hierarchical design methodology, see the *Design Compiler User Guide*.

When you synthesize your design using the UPF-based hierarchical flow, specify the voltage for each supply net. Also specify the timing constraints as recommended in the

[SolvNetPlus article 026172, “IEEE \(1801\) UPF Based Design Compiler Topographical Technology and IC Compiler Hierarchical Design Methodology.”](#)

In the hierarchical implementation of a design, you first determine the physical partition. Follow these guidelines while partitioning your design:

- The scopes of all power domains within a partition must be contained inside the partition.
- For all supply connections inside a partition, supply nets must be specified within the partition.
- The partitions should not be nested.

Hierarchical UPF Design Methodology

The following topics describe procedures for hierarchical designs.

- [Block-Level Implementation](#)
- [Top-Level Implementation](#)
- [Assembling the Design](#)

Block-Level Implementation

Creating the Blocks

Create the block-level and top-level UPF files for the design. To create the blocks, you can use either the top-down approach or the bottom-up approach. The bottom-up approach is preferable because this determines the smallest block that can be compiled independently.

When the individual blocks and the top are synthesized, you can assemble the design in either the Design Compiler or IC Compiler II tools. To assemble the design using the IC Compiler II tool, the Power Compiler tool requires the complete design database for the design planning stage. For more details, see [Assembling the Design](#).

Generating the Block-Level UPF Constraints

To use the hierarchical UPF methodology, your constraint specification in the UPF file must also be hierarchical. You can choose one of the following two ways to create the block-level and top-level UPF files:

- Write the power intent manually in the UPF file for all the blocks, including the top. If required, write the boundary constraints for the blocks.
- Use the `characterize` command to create the block-level UPF constraints as well as the boundary constraints from the full chip UPF description. It is important to remember

the following points when you use the `characterize` command to generate the block-level UPF constraints:

- If your design does not have the control signals at the block-level interfaces and you cannot modify your block level interfaces, you must use the `characterize` command to generate the block-level UPF constraints.
- By default, the `characterize` command propagates the UPF constraints in the top design to the subblock.

However, if you use this approach, you can perform equivalence checking only on the entire design and not on each hierarchical block.

Note:

All necessary power management control signals should be created manually. They also have to be manually brought into the appropriate block-level interfaces. This is the recommended approach.

Using Manually Created Block-Level UPF Files

When you create the blocks manually, each block and its power intent in the UPF file must be written such that each block can be simulated and synthesized independently. You might have to write the boundary constraints for the blocks to capture any port that does not operate at the same voltage as the rest of the block. If a block contains a power domain, the UPF constraints refer to objects and power supplies only within the block.

Using ETMs and Macros for Block-Level UPF Files

An ETM (Extracted Timing Model) is the Liberty model representation of a design. An ETM captures the UPF information using relevant Liberty attributes. A macro design might be represented using the Liberty model or it might be an IP provided by a vendor. The tool treats macros and ETMs in the same way and does not distinguish between the two types of implementation.

The tool supports ETMs and macros in the UPF hierarchical flow as follows:

- Only the UPF of the interface logic is required for the hierarchical UPF implementation
- You can also provide the full block UPF of the macro or ETM for the tool to automatically extract the interface power intent
- The UPF intent of the macro or ETM design's top power domain is used for design integration

The UPF constructs can be created and referenced at the top-most scope of a macro or ETM. Set the `upf_suppress_etm_model_checking` variable to `true` (default is `false`)

if you want to bypass ETM and macro checking when loading a UPF file. For example, if ublock is modeled as an ETM, you would do the following:

```
set_app_var upf_suppress_etm_model_checking true
load_upf block.upf -scope ublock
```

Note that UPF constructs defined at the scope of nested logic within a macro or ETM are read and ignored.

Using Design Compiler Generated Block-Level UPF Files

If you use the top-down approach to write your design or if your UPF file is nonmodular, Design Compiler can generate the block-level UPF using the `characterize` command. For the tool to correctly generate the block-level UPF file, your power domain definition and partitioning should comply with the guidelines mentioned in [UPF-Based Hierarchical Multivoltage Flow Methodology](#). The UPF objects in the block should not refer to any object that is above the block in the hierarchy. You should follow these steps to synthesize your design using the hierarchical UPF design methodology:

1. Read the design and the UPF constraints for the entire design.
2. Specify the operating voltages for the supply nets and specify the timing constraints.
3. For each subblock in the design, perform the following tasks:

- a. Run the `characterize` command.

This command pushes the appropriate timing and power constraints from the top-level to the specified block. The block-level power constraints and the boundary constraints that are specified by the `set_related_supply_net` command are set on the specified block. For more details, see [Characterization of Supply Sets and Supply Nets](#).

The `characterize` command can also automatically set the related supply net on the ports of the block-partition. To avoid voltage violations at the boundary, that can be caused by the automatic setting of related supply net, you must define level-shifter strategies at the block-partition boundary. If you do not want certain ports to be level shifted, use the `set_level_shifter -no_shift` command. For more details see [Automatic Inference of Related Supply Nets](#).

While setting the related supply net, additional checks are performed for voltage violations, availability of the supply net, and so on, and appropriate error and warning messages are issued.

- b. Save the characterized block and the design data.

Set the characterized block as the current instance and use the `write` command to save the characterized block. The command sequence is shown in the following example:

```
characterize BlockA
set current_instance BlockA
write -format ddc -hierarchy -output BlockA.characterized.ddc
```

- c. Remove the block from the top level using the `remove_design -hierarchy` command. When you remove the block, the UPF constraints associated with the block are also removed.
4. When all the subblocks have been characterized, saved in .ddc format, and removed, save the top-level design in .ddc format.

Synthesizing the Blocks

To synthesize each subblock of the hierarchical design, read the design in one of the following two methods:

- The RTL file and the manually written UPF file for each block.
- GTECH netlist in the .ddc file for each block, written after the characterization step.

The difference between the two is the clarity of the block-level UPF and the automatic inclusion of boundary constraints when you use the .ddc file generated after the characterization step and the ability to perform hierarchical verification using Formality. The power intent created by the `characterize` command is the same as the manually created UPF file. If you use the RTL design and the manually written UPF file, you should create appropriate boundary constraints.

You then use either the top-down or bottom-up synthesis flow options supported in Design Compiler topographical mode to perform block-level synthesis. For more information, see [SolvNetPlus article 021034, “Hierarchical Flow Support in Design Compiler Topographical Mode.”](#)

Top-Level Implementation

Follow these steps to perform the top-level synthesis:

1. Read the block-level designs.

The block-level design can be any one of the following types:

- A synthesized block-level design
- A block abstraction created in either the Design Compiler or IC Compiler II tool

Specify the blocks using the `set_top_implementation_options` command.

2. Read the top-level design in either of the following formats:

- RTL design and UPF files; use the `load_upf` command to read the UPF file
- GTECH netlist in .ddc file format, obtained after removing all the characterized subblocks

Note: When reading the top-level UPF file before the block-level UPF file, you must set the `upf_allow_refer_before_define` variable to `true` to allow loading a top-only UPF file with references to unlinked subblocks within the design. By default, this variable is set to `false`.

3. Run the `propagate_constraints -power_supply_data` command.

This command propagates all the block-level constraints to the top-level, including the block abstractions created in either the Design Compiler or IC Compiler II tool that contain UPF data.

4. Synthesize the top-level design.

For the block abstractions created in Design Compiler topographical mode, the tool performs size-only optimization on the block interface logic, including the power management cells.

5. Save the synthesized design and the UPF constraints.

When you save the design in .ddc file format, the UPF constraints are also saved in the file. To save the UPF constraints separately, use the `save_upf` command. To save the complete UPF information, use the `save_upf -full_upf` command. To save only the top-only UPF, use the `save_upf` command. You can also set the `upf_block_partition` variable to specify the name of a block or a list of blocks whose

UPF information you do not want to save. For instance, if you have a subblock named block1 and you want to save the top UPF without block1, do the following:

```
set_app_var upf_block_partition block1
save_upf
```

Note that if block1 is an ETM, macro, or any other black box, you do not need to specify it in the `upf_block_partition` list since the `save_upf` command automatically skips these blocks.

Completing these steps completes the synthesis of your design using the Design Compiler hierarchical UPF flow. Using the synthesized design, you can continue the flow in the IC Compiler II tool. For more details on assembling your design for the subsequent steps in the IC Compiler II tool, see [Assembling the Design](#).

Assembling the Design

To continue with the hierarchical flow in the IC Compiler II tool, you can assemble your design in either the Design Compiler or IC Compiler II tool. Note that you must explicitly ensure that the block-level UPF constraints are available in the top-level design during the optimization step of the top-level. You do this using the `propagate_constraints -power_supply_data` command. Use the following steps to assemble your design in the Design Compiler tool for use in the IC Compiler II tool:

1. Read all the synthesized subblocks.
2. Set the top-level design as the current design.
3. Link the design using the `link` command.
4. Use the `propagate_constraints -power_supply_data` command for all the block-level UPF constraints to be available at the top-level.
5. Save the design. This saved design is the full-chip design database that you can use to start the design planning step in the IC Compiler II tool.

For more information, see [SolvNetPlus article 026172, “IEEE 1801 \(UPF\) Based Design Compiler Topographical Technology and IC Compiler Hierarchical Design Methodology.”](#)

Characterization of Supply Sets and Supply Nets

A supply set or a domain-independent supply net of a block is characterized when it is any of the following:

- The primary, default retention, default isolation supply of the power domain of the block
- The supplies specified in the retention or isolation strategies of the power domain of the block

- A supply that is specified for the power switch of the power domain of the block
- An exception supply that is connected to the cells in the power domain of the block
- An extra supply of the power domain, defined by using the `extra_supplies_#` keyword
- A supply set that is connected to the supply ports that are defined inside the block
- A supply set that is the related supply for the ports of the block

Note:

In this case, the supplies are characterized even if they are the restricted supplies in the top-level power domain of the block being characterized. This is because, the block can contain an unrestricted feedthrough supply that passes through power domains.

While partitioning a block, the tool moves supply sets defined in the block and in lower levels of hierarchy to the block. The supply sets and domain-independent supply nets are handled similarly because supply sets are also inherently domain-independent.

When a `repeater_supply` attribute is specified in the path of an isolation strategy defined using the `-source`, `-sink`, or `-diff_supply_only` option, the value of the `repeater_supply` attribute is used to derive the value of the `iso_source` and `iso_sink` attributes at the boundary of the block.

During characterization, at the block level,

- Two supply ports and a supply set are created. The supply ports are connected to the power and ground functions of the supply set.
- To distinguish the supply ports created by the `characterize` command, the newly-created supply ports are marked with the `smps_derived` UPF attribute. So, each supply port created by the `characterize` command has an associated `set_port_attributes` command in the block-level UPF file.
- If you have defined power states for the supply sets for the block-level, using the `add_power_state` command, during characterization, the tool writes the `add_port_state` command for the created port.

At the top level, and in the UPF file for the top level, two ports are created, which are connected to the power and ground functions of the supply set.

Automatic Inference of Related Supply Nets

In the top-down hierarchical flow, when you characterize a block, the block-level power constraints as well as the boundary constraints that are specified by the `set_related_supply_net` command are set on the specified block.

The `characterize` command can also automatically set the related supply net on the ports of the block-partition, using the following criteria:

- The direction of the port.
- The location constraint of the isolation and level-shifter strategies.
- Related supply net of the driver or the load cells.
- The `-driver_supply` and `-receiver_supply` options specified with the `set_port_attributes` command.

The `characterize` command can also infer the driver or load to be inserted at the boundaries.

Note:

For the `characterize` command to appropriately infer and set the related supply net, you must explicitly define the level-shifter and isolation strategies before running the `characterize` command, if you have voltage violations.

[Table 38](#) shows the related supply net inferred by the Power Compiler tool when you define only the level-shifter strategy, and not the isolation strategy, to overcome the voltage violations at the boundary pins.

Table 38 Related Supply Net With Level-Shifter Only Strategy

Port direction	Level-shifter strategy	Related supply net inferred by the tool
Input	self	Outside or driver supply net. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	Inside or load supply net.
Output	self	Outside or load supply net. If supply is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	Inside or driver supply net.
Input or Output	none or auto	Not supported. UPF-206 error message is issued.

[Table 39](#) shows the related supply net inferred by the Power Compiler tool when you define both level-shifter and isolation strategies.

Table 39 Related Supply Net With Level-Shifter and Isolation Strategies

Port direction	Level-shifter strategy	Isolation strategy	Related supply net inferred by the tool
Input	self	self	Outside or driver supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	self	parent	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	self	Related supply net is not set and UPF-207 error message is issued.
Input	parent	parent	Inside or load supply net.
Input	none or auto	self or parent	Not supported. UPF-206 error message is issued.
Output	self	self	Outside or load supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	self	parent	Related supply net is not set and UPF-207 error message is issued
Output	parent	self	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	parent	Inside or driver supply.
Output	none/auto	self or parent	Not supported. UPF-206 error message is issued.

Table 40 shows the related supply net inferred by the Power Compiler tool when there are no voltage violations at the boundary pins.

Table 40 Related Supply Net With No Voltage Violations at the Boundary Pins

Port direction	Isolation strategy	Related supply net inferred by the tool
Input	self	Outside or driver supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	Isolation power supply. If supply net is not available, use the inside or load supply.
Input	none	Outside or driver supply. If supply net is not available, use the inside or load supply.

Table 40 Related Supply Net With No Voltage Violations at the Boundary Pins (Continued)

Port direction	Isolation strategy	Related supply net inferred by the tool
Output	self	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	Inside or driver supply.
Output	none	Outside or load supply. If supply net is not available, use the inside or load supply.

Note:

If voltage violations are across two blocks that have to be characterized, define the level-shifter strategies for both the blocks. To avoid level-shifter redundancy, use the `-no_shift` option of the `set_level_shifter` command. If the violations are across multiple blocks, specify the list of pins while defining the level shifter strategy with the `-no_shift` option.

Top-Level Design Integration

After the blocks are characterized, these blocks can be integrated into the top-level designs, multiple times. Use the `propagate_constraints` command to integrate the characterized blocks to the top level.

Power Domain Merging

While merging the power domain to the top level, the `propagate_constraints` command ensures that equivalent supply sets, nets, and ports are present at the top level. In addition, their connectivity should be equivalent at the top level. The tool issues the UPF-168 error message when equivalence is not found.

During integration, the block-level ports that have the `smps_derived` UPF attributes are substituted by their equivalent top-level ports and supply nets or supply sets.

Switch Cell Matching

When a power switch cell exists in the blocks, a matching switch cell must exist at the top level for the domains to be merged. It is an error to match a switch cell from the block to a switch cell in the top level, that is already matched.

When the tool merges domains that contain switch cells, the following rules apply:

- A switch cell in the top level should have a unique switch cell in the domain being merged. The switch cells being merged should also have equivalent
 - Input supply nets
 - Control and acknowledge signals separated by buffer or inverter pairs
 - Voltage setting on the output supply nets
 - Port states, including the state names, state value, primary domain, and so on
 Also, the output supply nets must have similar connectivity with the other supply nets in the design.
- When the domain has multiple equivalent switch cells, the first matching switch cell is used.

Legacy Blocks

A legacy block is a block designed before the introduction of supply sets, using only domain-dependent supply nets. A legacy block does not use or define any domain-independent supply nets or supply sets.

A conflict can arise when a legacy block is used in a design with domain-independent supply nets. To prevent such a conflict, set the block's `legacy_block` design attribute to `true`. This converts all power domains of the legacy block to be fully restricted, so the legacy block can no longer use any domain-independent supply nets declared in the scopes above the block.

A domain-independent supply net is a supply net that is available to any power domain defined at or below the scope of the supply net, as long such domains are not restricted. In other words, the supply net was created by a `create_supply_net` command without the `-domain` option. For example,

```
dc_shell> create_supply_net SN1
```

Conversely, a domain-dependent supply net is a supply net that is available only to the domain for which it is defined. In other words, the supply net was created by a `create_supply_net` command with the `-domain` option. For example,

```
dc_shell> create_supply_net SN2 -domain PD2
```

The supply net is created in the PD2 power domain and cannot be used in other domains.

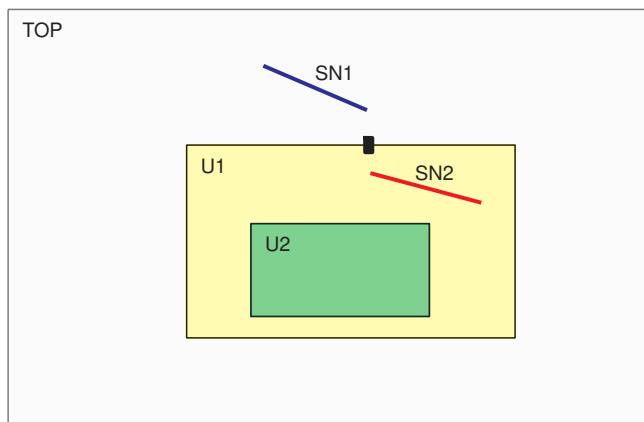
A restricted power domain is a power domain that is restricted to use only certain supply sets. This restriction results from usage of the `extra_supplies` keyword with the `-supply` option of the `create_power_domain` command. For example,

```
dc_shell> create_power_domain PD3 -elements U3 \
           -supply {extra_supplies_0 SS1}
```

The PD3 power domain is restricted to using only the SS1 supply set.

When a legacy block is used in a newer design containing supply sets, a conflict can arise with a situation like the one shown in [Figure 123](#).

Figure 123 Legacy Block Used in a Top-Level Design



In this diagram, U1 is a legacy block with a domain-dependent supply net, SN2. An instance of this block is used in the top-level design, TOP, which has a domain-independent supply net, SN1. U1 contains a lower-level block, U2. Because the SN1 supply net is domain-independent, it is available for use in all of the domains. On the other hand, because the SN2 supply net is domain-dependent, it is available for use only in the domain of block U1.

If the two supply nets are connected through a supply port on U1, the availability of the combined net in U2 is undefined. This might lead to the usage of the supply net in U2, which would be incorrect.

To clearly specify that this type of connection is not allowed, declare U2 to be a legacy block by using the following command:

```
dc_shell> set_design_attributes -elements U1 \
           -attribute legacy_block true
```

This command converts all power domains of the U1 block to fully restricted domains so that those domains can no longer use the domain-independent supply nets declared in

scopes above the block. The tool achieves this effect by using the `-supply` option of the `create_power_domain` command when the block is read.

For example, the tool changes the following command:

```
create_power_domain PD -elements U2
```

to the following command:

```
create_power_domain PD -elements U2 -supply {extra_supplies ""}
```

Because there are no supply sets listed between the quotation marks in the `-supply` option, the domain becomes fully restricted and does not allow the usage of any supply sets defined at higher levels of the design, thereby preventing any supply set availability conflict from arising.

No domain-independent supply nets or supply sets can be defined or used inside a legacy block or any of its lower-level blocks, and no supply set handles can be used. Any lower-level blocks below a legacy block must also be legacy blocks.

The `propagate_constraints` command supports the usage of legacy blocks. The following script shows an example of the flow.

```
read_verilog top_only.verilog
load_upf top_only.upf
read_verilog my_legacy_block.verilog
load_upf my_legacy_block.upf
current_design top
#Mark the block as legacy block
set_design_attributes -elements {U2} -attribute legacy_block TRUE
#propagate_constraints makes all the domains in the block restricted
propagate_constraints -design my_legacy_block
```

Note:

The `characterize` command is not supported for legacy blocks.

12

Library Setup for Power Optimization

To perform power optimization on a multivoltage design using the Power Compiler tool, the target libraries you use must conform to the Liberty open library rules.

For more information about library setup for power optimization, see the following topics:

- [Basic Library Requirements for Multivoltage Designs](#)
 - [Library Usage in Multicorner-Multimode Designs](#)
 - [Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells](#)
-

Basic Library Requirements for Multivoltage Designs

To synthesize a multivoltage design using the Power Compiler tool, the target libraries you use must conform to the Liberty open library rules. The target libraries should also support special cells such as clock-gating cells, level-shifters, isolation cells, retention registers, and always-on buffers and inverters. To support synthesis of multivoltage designs, the tool also supports multiple libraries characterized at different voltages.

Power and Ground Pin Syntax

If the target library that you specify complies with the power and ground (PG) pin Liberty library syntax, the Power Compiler tool uses this information during the synthesis process. However, if your target library does not contain PG pin information, you can convert it into PG pin library format. For more information, see [Converting Libraries to PG Pin Library Format](#).

Converting Libraries to PG Pin Library Format

If the libraries that you specify do not contain PG pin information, you can define them in the library to conform to PG pin Liberty syntax.

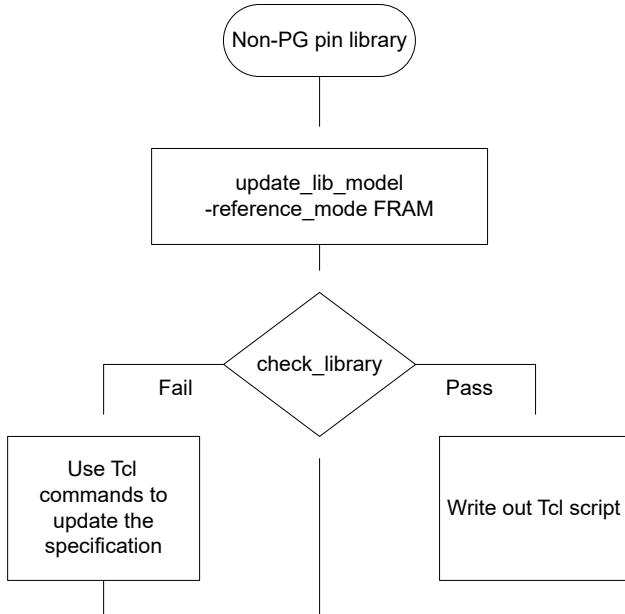
- [Using the FRAM View](#)
 - [Using Tcl Commands](#)
 - [Tcl Commands for Low-Power Library Specification](#)
-

For more information, see [SolvNetPlus article 029641, “On-the-Fly Low-Power Library Specification.”](#)

Using the FRAM View

In the Design Compiler topographical mode, you can use the FRAM view as the reference for converting your library to the PG pin library format. You must set the `mw_reference_library` variable to the location of the Milkyway reference libraries. Use the `update_lib_model` command to convert your library to the PG pin library format. The tool uses the PG pin definitions available in the FRAM view of the Milkyway library for the conversion. This is the default behavior. [Figure 124](#) shows the steps involved in converting non-PG pin library to a PG pin library.

Figure 124 Conversion of a Non-PG Pin Library to a PG Pin Library Using FRAM View



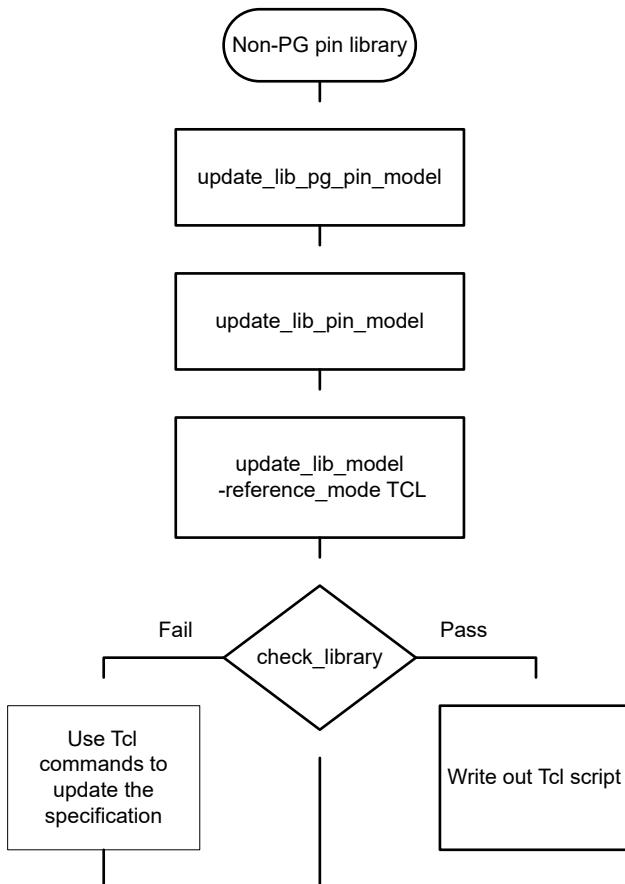
To ensure that the PG pin library that is created is complete, use the `check_library` and `report_mv_library_cells` commands. If the PG pin library is not complete, run the library specification Tcl commands to complete the library creation. For more information, see [Tcl Commands for Low-Power Library Specification](#).

Using Tcl Commands

When your library files are not in the PG pin library syntax and you do not have the FRAM view of Milkyway library, you can use the following Tcl commands to specify the necessary information required for deriving the PG pin details, as shown in [Figure 125](#).

- `update_lib_voltage_model`
 This command sets the voltage map for the specified library.
- `update_lib_pg_pin_model`
 This command sets the PG pin map for the specified library cell.
- `update_lib_pin_model`
 This command sets the pin map for the specified library cell.

Figure 125 Conversion of Non-PG Pin Library to PG Pin Library Using Tcl Commands



These Tcl commands specify the library requirements that are used while converting the libraries to PG pin format.

Run the `update_lib_model -reference_mode TCL` command to convert your libraries to PG pin library format. To check if your newly created PG pin library is complete, run the `check_library` command. If your newly created PG pin library contains conflicts or

is incomplete, you can run the library specification Tcl commands to complete the library specification. For more details, see [Tcl Commands for Low-Power Library Specification](#).

Tcl Commands for Low-Power Library Specification

When you convert your library to PG pin format, if the newly created library file is complete, you can start using the library for the low-power implementation of your design. However, if your library contains power management cells and the modeling is not complete, you can use the following Tcl commands to complete your library specifications. These commands specify the library voltage and PG pin characteristics.

- `set_voltage_model`

This command sets the voltage model on the specified library by updating the voltage map in the library.

- `set_pg_pin_model`

This command defines the PG pins for the specified cell.

- `set_pin_model`

This command defines the related power, ground, or bias pins of the specified pin of the library.

For more details, see the command man page and the Library Checking Chapter in the *Library Quality Assurance System User Guide*.

Macro Cells With Fine-Grained Switches

The Power Compiler tool supports macro cells with fine-grained switches, which have the following attribute settings in the PG pin definition in the library:

- The `direction` attribute is `internal`.
- The `pg_type` attribute is either `internal_power` or `internal_ground`.
- The `pg_function` attribute is defined.
- The `switch_function` attribute is defined.
- The `switch_cell_type` attribute of the macro is `fine_grain`.
- The `switch_pin` attribute is set to `true` for the control port.

Use the `connect_supply_net` command to connect to the internal PG pins of these macro cells. However, supply nets connected only to the internal PG pins of these macro

cells cannot be used for level-shifter insertion and always-on synthesis, unless the following conditions are true:

- The supply net is the primary supply of the power domain.
- The supply net is specified by the isolation strategy of the power domain.
- The supply net is specified by the retention strategy of the power domain.
- The supply net is defined or reused as a domain-dependent supply net of the power domain.
- The supply net is defined with the `extra_supplies_#` keyword.

You can use the `set_voltage` command to set the operating voltage on the internal PG pins of the macro cells with fine-grained switches. If you do not set the voltage on the internal PG pin of the macro cell, the value of the `voltage_name` attribute of the PG pin is used as the operating voltage.

Library Usage in Multicorner-Multimode Designs

The following topics discuss how to handle libraries properly in multicorner-multimode designs:

- [Link Libraries With Equal Nominal PVT Values](#)
- [Distinct PVT Requirements](#)
- [Automatic Detection of Driving Cell Library](#)
- [Relating the Minimum Library to the Maximum Library](#)
- [Unique Identification of Libraries Based on File Names](#)

Link Libraries With Equal Nominal PVT Values

The link library lists all of the libraries that are to be used for linking the design for all scenarios. Furthermore, because several libraries are often intended for use with a particular scenario, such as a standard cell library and a macro library, Design Compiler automatically groups the libraries in the link library list into sets and identifies which set must be linked with each scenario.

Library grouping is based on their PVT values. Libraries with the same PVT values are grouped into the same set. The tool uses the PVT value of a scenario's maximum operating condition to select the appropriate set for the scenario.

If the tool finds no suitable cell in any of the specified libraries, an error is reported, as shown in the following example:

```
Error: cell TEST_BUFBn_BUFA/Z (inx4) is not characterized
for 0.950000V, process 1.000000, temperature -40.000000. (LIBSETUP-001)
```

You should verify the operating conditions and library setup. If you do not correct this error, optimization is not performed.

Link Library Example

Table 41 shows the libraries in the link library list, their nominal PVT values, and the operating condition (if any) specified in each library. The design has instances of combinational, sequential, and macro cells.

Table 41 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Combo_cells_slow.db	1/0.85/130	WORST (1/0.85/130)
Sequentials_fast.db	1/1.30/100	None
Macros_fast.db	1/1.30/100	None
Macros_slow.db	1/0.85/130	None
Combo_cells_fast.db	1/1.30/100	BEST (1/1.3/100)
Sequentials_slow.db	1/0.85/130	None

To create a scenario s1 with the cell instances linked to the Combo_cells_slow, Macros_slow, and Sequential_slow libraries, you run:

```
dc_shell-topo> create_scenario s1
dc_shell-topo> set_operating_conditions -max WORST -library slow.db:slow
```

Note:

Specifying the `-library` option with the `set_operating_conditions` command helps the tool identify the correct PVT for the operating conditions. The PVT of the maximum operating condition is used to find the correct matches in the link library list during linking.

Using this linking scheme, you can link libraries that do not have operating condition definitions. The scheme also provides the flexibility of having multiple library files (for example, one for standard cells, another for macros).

Inconsistent Libraries Warning

When you use multiple libraries, if the library cells with the same name are not functionally identical or do not have identical sets of library pins (same name and order), a warning is issued, stating that the libraries are inconsistent.

You should run the `check_library` command before running a multicorner-multimode flow, as shown in the following example:

```
set_check_library_options -mcmm
check_library -logic_library_name {a.db b.db}
```

When you use the `-mcmm` option with the `set_check_library_options` command, the `check_library` command performs multicorner-multimode specific checks such as the operating condition or power-down consistencies. When inconsistencies are detected, the tool generates a report. In addition, the tool also issues the following information message:

```
Information: Logic library consistency check FAILED for MCMM.
(LIBCHK-360)
```

To overcome the LIBCHK-360 messages, you must check the libraries and the report to identify the cause for the inconsistency. The man page of the LIBCHK-360 information message describes possible causes for the library inconsistencies.

Setting the `dont_use` Attribute on Library Cells

When you set the `dont_use` attribute on a library cell, the multicorner-multimode feature requires that all characterizations of this cell have the `dont_use` attribute. Otherwise, the tool might consider the libraries to be inconsistent. You can use the wildcard character to set the `dont_use` attribute as follows:

```
set_dont_use */AN2
```

When library cells with a `dont_use` attribute have a pin order that does not match exactly in the libraries of different corners, the tool continues with the flow without issuing any error or warning messages. If you remove the `dont_use` attribute of these cells, the tool issues the MV-087 error messages.

Note:

You do not have to issue the command multiple times to set the `dont_use` attribute on all characterizations of a library cell.

Distinct PVT Requirements

If the maximum libraries associated with each corner (scenario) do not have distinct PVT values, the cell instances are linked incorrectly, which results in incorrect timing values. This happens because the nominal PVT values that are used to group the link libraries into sets, group the maximum libraries of different corners into one set. Consequently,

the cell instances are linked to the first cell with a matching type in that set (for example, the first AND2_4), even though the `-library` option is specified for each of the scenario-specific `set_operating_conditions` commands. That is, the `-library` option locates the operating condition and its PVT values, but not the library to link.

For example, consider **Table 42**, which shows libraries in a link library, listed in order, their nominal PVT values, and the operating condition specified in each library.

Table 42 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Ftyp.db	1/1.30/100	WORST (1/1.30/100)
Typ.db	1/0.85/100	WORST (1/0.85/100)
TypHV.db	1/1.30/100	WORST (1/1.30/100)
Holdtyp.db	1/0.85/100	BEST (1/0.85/100)

Table 43 shows the operating condition for each of the scenarios.

Table 43 Scenarios and Their Operating Conditions

	Scenarios			
	s1	s2	s3	s4
Maximum Operating Condition (Library)	WORST (Typ.db)	WORST (TypHV.db)	WORST (Ftyp.db)	WORST (Typ.db)
Minimum Operating Condition (Library)	None	None	None	BEST (HoldTyp.db)

The following commands are applied:

```
create_scenario s1
set_operating_conditions WORST -library Typ.db:Typ
create_scenario s2
set_operating_conditions WORST -library TypHV.db:TypHV
create_scenario s3
set_operating_conditions WORST -library Ftyp.db:Ftyp
create_scenario s4
set_operating_conditions \
```

```
-max WORST -max_library Typ.db:Typ \
-min BEST -min_library HoldTyp.db:HoldTyp
```

The tool groups the Ftyp.db, and TypHV.db libraries into a set with Ftyp.db as the first library in the set. Therefore, the cell instances in scenario s2 are not linked to the library cells in TypHV.db, as intended. Instead, they are linked to the library cells in the Ftyp.db library, assuming that all the libraries include the library cells required to link the design.

When you use multiple libraries, if any of the libraries with same-name cells have the same nominal PVT, a warning is issued, stating that the libraries are ambiguous. The warning also states which libraries are being used and which are being ignored.

Automatic Detection of Driving Cell Library

In multicorner-multimode flow, the operating condition setting is different for different scenarios. To build the timing arc for the driving cell, different libraries are used for different scenarios. You can specify the library using the `-library` option of the `set_driving_cell` command. However, specifying the library is optional because the tool can automatically detect the driving cell library.

When you specify the library using the `-library` option of the `set_driving_cell` command, the tool searches for the specified library in the link library set. If the specified library exists, it is used. If the specified library does not exist in the link library, the tool issues the UID-993 error message as follows:

```
Error: Cannot find the specified driving cell in memory. (UID-993)
```

When you do not use the `-library` option of the `set_driving_cell` command, the tool searches all the libraries for the matching operating conditions. The first library in the link library set, that matches the operating condition is used. If no library in the link library set matches the operating condition, the first library in the link library set, that contains the matching library cell is used. If no library in the link library set contains the matching library cell, the tool issues the UID-993 error message.

Relating the Minimum Library to the Maximum Library

The `set_min_library` command is not scenario-specific. This implies that if you use this command to relate a minimum library to a particular maximum library, that relationship applies to all scenarios.

Table 44 *Unsupported Multiple Minimum Library Configuration*

Scenarios	
s1	s2
Maximum library	Slow.db
Minimum library	Fast_0yr.db

For example, you could not relate two different minimum libraries—for example, Fast_0yr.db and Fast_10yr.db—with the maximum library, Slow.db, in two separate scenarios. The first minimum library that you specify would apply to both scenarios. **Table 44** shows the *unsupported* configuration.

Note, however, that a minimum library can be associated with multiple maximum libraries. As shown in the example in [Table 45](#), the minimum library Fast_0yr.db is paired with both the maximum library Slow.db of scenario 1 and the maximum library SlowHV.db of scenario 2.

Table 45 *Supported Minimum-Maximum Library Configuration*

Scenarios	
s1	s2
Maximum Library	Slow.db
Minimum Library	Fast_0yr.db

Unique Identification of Libraries Based on File Names

Two libraries with the same name can be uniquely identified if their file names, which precede the library names, which are colon-separated, are unique. For example, the library ABC.db:stdcell (where ABC.db is the library file name and stdcell is the library name) is identifiable with respect to the library DEF.db:stdcell.

However, two libraries that have the same file name and library name but reside in different directories are not uniquely distinguishable. The following two libraries are not uniquely distinguishable:

```
/remote/snps/testcase/LIB/fast/ABC.db
```

```
/remote/snps/testcase/LIB/slow/ABC.db
```

Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells

In multivoltage and multicorner-multimode designs, as designs increase in size and complexity, manually specifying the operating conditions and linking them with the appropriate library cells with matching operating conditions becomes difficult. For these types of designs, it is useful to automatically infer the operating conditions, especially for special cells such as multirail pad cells, macro cells and switch cells. When the operating condition set on the design does not match the operating condition of the cell rails or when the design operating condition does not have rails, the tool issues a LIBSETUP-001 error message.

You can disable the automatic operating conditions inference by explicitly setting the operating conditions.

Note:

The Power Compiler tool does not perform automatic operating condition inference for standard cells. The operating conditions of the standard cells should match exactly with the operating conditions of the design.

Using the `set_opcond_inference` Command

Use the `set_opcond_inference` command to specify the operating condition.

Use the `-level` option specifies the degree to which the inferred operating condition can deviate from the operating condition of the design. The value that you can specify with this option are EXACT, UNIQUE_RESOLVED, CLOSEST_RESOLVED, or CLOSEST_UNRESOLVED. When you do not specify this option, the default is CLOSEST_RESOLVED. For more information, see [Deviating From the Inferred Operating Condition and Its Impact](#).

You must use one of `-level` and `-match_process_temperature` options. The tool issues a LIBSETUP-751 information message when the operating conditions are successfully inferred on a cell instance.

For multicorner-multimode designs, the `set_opcond_inference` command applies to all corners and scenarios of the design. To report the settings specified for the operating condition inference, use the `report_opcond_inference` command.

Deviating From the Inferred Operating Condition and Its Impact

The level value you specify with the `-level` option of the `set_opcond_inference` command determines how much the inferred operating condition can deviate from the operating condition of the design. When you set a higher deviation, the probability of automatic operating condition inference is higher, resulting in a smaller number of

LIBSETUP-001 error messages. This also implies less accurate timing and power results. The following table summarizes the level values that you can specify with the `-level` option and its impact on the automatic operating condition inference:

Level value specified with the <code>-level</code> option	Degree of deviation in the inferred operating condition and its impact
EXACT	Operating condition inferred is exact. This results in no inference at all. Timing is exact.
UNIQUE_RESOLVED	Operating condition is inferred for the library cell whose name matches exactly with the cell reference name in the design. You cannot choose a different library cell. Timing can be incorrect. You do not encounter LIBSETUP-001 error messages.
CLOSEST_RESOLVED	This is the default. If multiple library cells are available, library cell with a matching reference name whose operating condition is closest to the design is chosen. Choosing this operating condition can cause inaccurate timing.
CLOSEST_UNRESOLVED	The library cell whose operating condition is closest to the design is chosen. The library cell name need not match exactly with the cell reference name in the design.

The details of the behavior of the tool when you set a specific level value with the `-level` option of the `set_opcond_inference` command are described in this section:

- EXACT

When you set the level value to EXACT, the automatic operating condition inference is not performed.

- UNIQUE_RESOLVED

The tool performs a name based search in the target libraries. If multiple library cells match with the cell name, the tool does not perform the inference. However, if the cell is present in a unique library file and no other library contains the cell, the operating condition is inferred. Otherwise, operating condition is not inferred on the cell and a LIBSETUP-001 error message is issued.

- CLOSEST_RESOLVED

This is the default, when you do not specify the `-level` option of the `set_opcond_inference` command.

For each macro cell, pad cell, or switch cell instance, the tool finds the set of library cells with the same name. If multiple library cells with the same name are found, the tool chooses a single library cell based on the matching PVT values. The cells with

exception connections, whose supply net voltage does not match the rail voltages in the library, are also considered for operating conditions inference.

For cells with exception connections, the tool chooses the library cell with maximum number of rail voltages that match the supply net voltage of the instance. If there are multiple library cells with maximum number of rail voltages that match the supply net voltage of the instance, the inference fails and the tool issues a LIBSETUP-001 error message.

The pad cells in the library whose rail voltages do not match the supply voltage on the port because of the settings of the `set_port_attributes` or the `set_related_supply_net` command are eliminated from operating conditions inference. However, when the tool finds that such eliminations can cause potential LIBSETUP-001 errors, it reconsiders the eliminated cells for operating conditions inference.

Within this set of library cells that are considered for inference, the tool groups the library cells in the following order of priority:

1. The PVT values of the library cell match the PVT values of the design.
2. The process, temperature, and voltage values from one of the rail voltages match the PVT values of the design.
3. The temperature and voltage values of the library cell match the temperature and voltage values of the design.
4. The temperature and voltage from one of the rail voltages match the PVT values of the design.
5. The process and voltage values of the library cell match the process and voltage values of the design.
6. The process and voltage from one of the rail voltages match the PVT values of the design.
7. The voltage value of the library cell matches the voltage value of the design.
8. The voltage value from one of the rail voltages match the voltage value of the design.
9. The process and temperature values of the library cell matches the process and temperature values of the design.
10. None of the process, voltage, and temperature values of the library cell match the process, voltage, and temperature values of the design.

After the library cells are grouped, the tool inspects each group in the order mentioned previously. The inference is terminated for the following situations:

- None of the groups contain exactly one cell.
- None of the groups contain any library cell.

When the tool finds a group that contains exactly one cell, the tool chooses the library cell and uses the PVT values of that cell as the operating condition of the associated macro, pad, or switch cell.

- CLOSEST_UNRESOLVED

The tool groups the library cells based on the matching names, as in CLOSEST_RESOLVED. The tool then picks the first library cell from the first non-empty group of library cells. It then sets the operating condition of the library cell on the specific cell instance and links the cell instance to the library cell.

13

Power Optimization in Multicorner-Multimode Designs

Designs that can be synthesized using multiple operating conditions and in multiple modes are called multicorner-multimode designs. The Design Compiler Graphical tool extends the topographical technology to analyze and optimize these designs.

For more information about synthesis tool support for multicorner-multimode technology, see the following topics:

- [Optimizing Multicorner-Multimode Designs](#)
 - [Reporting Commands](#)
 - [Script Example for Multicorner-Multimode Flow](#)
-

Optimizing Multicorner-Multimode Designs

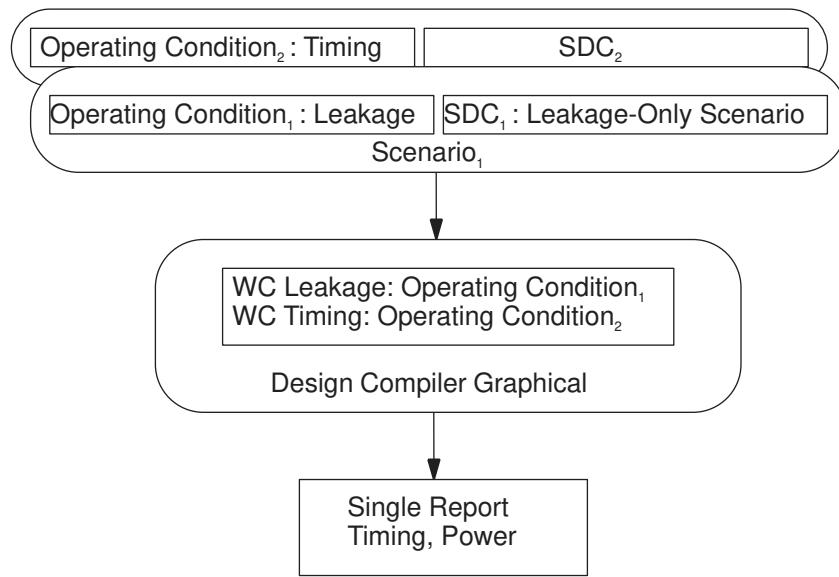
Designs that can be synthesized using multiple operating conditions and in multiple modes are called multicorner-multimode designs. Design Compiler Graphical extends the topographical technology to analyze and optimize these designs. The multicorner-multimode feature also provides ease-of-use and compatibility between flows in the Design Compiler and IC Compiler II tools.

For general information about multicorner-multimode concepts and features, see the *Design Compiler User Guide* and *IC Compiler II Implementation User Guide*.

Optimizing for Leakage Power

[Figure 126](#) shows how to set constraints on different scenarios of a multicorner-multimode design.

Figure 126 Setting Different Constraints on Different Scenarios



Typically, in a multicorner-multimode design, leakage power optimization and timing optimization are done on different corners. Therefore, the worst case leakage corner can be different from the worst case timing corner. To perform leakage power optimization on specific corners, set the leakage power option on specific scenarios of the multicorner-multimode design by using the `set_scenario_options` command as follows:

```
set_scenario_options -scenarios S1 \
    -setup false \
    -hold false \
    -leakage_power true
```

When you optimize for leakage power in multicorner-multimode designs,

- Define the leakage power option on specific scenarios targeted for leakage power optimization.
- Leakage and timing optimizations can be performed concurrently across multiple scenarios.
- The worst case leakage corner is different from the worst case timing corner.
- Do not use the `set_leakage_optimization` command inside a scenario. This command is supported only for non multicorner-multimode designs.

If no leakage scenario is defined, the average leakage value of all the scenarios is used for leakage optimization.

When you use the `set_multi_vth_constraint` command, you must specify a leakage corner using the `set_scenario_options -scenarios` command.

The following example shows how leakage power is specified on a multicorner-multimode design. In this example, leakage power optimization is performed only for scenario_1 and scenario_3 because the `-leakage_power` option is true:

```
set_scenario_options -scenarios {scenarios_1, scenarios_3} \
    -leakage_power true
set_scenario_options -scenarios {scenarios_2, scenarios_4} \
    -leakage_power false
```

Example 45 shows how to create a scenario and set the leakage power option on the scenario:

Example 45 Leakage Power Optimization in a Multicorner-Multimode Design

```
read_verilog top.v
current_design top
link
create_scenario s1
set_operating_conditions WCCOM -library slow.db:slow
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s1.sdc
set_switching_activity -toggle_rate 0.25 \
    -base_clock p_Clk -static_probability 0.015 -type inputs
set_scenario_options -scenarios s1 -setup false -hold false \
    -leakage_power true

create_scenario s2
set_operating_conditions BCCOM -library fast.db:fast
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s2.sdc

create_scenario s3
set_operating_conditions TCCOM -library typ.db:typ
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s3.sdc

create_scenario s4
set_operating_conditions NCCOM -library typ2.db:typ2
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s4.sdc
set_scenario_options -scenarios s4 -setup false -hold false \
    -leakage_power true

report_scenarios
compile_ultra -scan -gate_clock
report_power -scenarios [all_scenarios]
report_timing -scenarios [all_scenarios]
report_scenarios
report_qor
report_saif
```

Optimizing for Dynamic Power Using Low-Power Placement

To perform dynamic power optimization in a multicorner-multimode design, use the `set_scenario_options -dynamic_power true -setup true` command. This command performs scenario-specific dynamic power optimization in a multicorner-multimode design. For multiple dynamic power scenarios, the tool uses the average switching activity calculated from data in the SAIF files when performing optimization.

In the Synopsys physical guidance flow, when you enable the low-power placement feature, the tool performs dynamic power optimization for multicorner-multimode designs. To enable this feature, set the `power_low_power_placement` variable to `true` and specify the dynamic power and setup constraints for the scenario. [Example 46](#) shows a script to perform dynamic power optimization in multicorner-multimode designs in the Synopsys physical guidance flow.

Example 46 Dynamic Power Optimization in a Multicorner-multimode Design

```
set power_low_power_placement true
current_scenario S1
read_saif -input S1.saif
set_scenario_options -dynamic_power true -setup true
compile_ultra -spg
```

For more information about dynamic power optimization see [Dynamic Power Optimization](#).

Reporting Commands

This section describes the commands that you can use for reporting multicorner-multimode designs.

report_scenarios Command

The `report_scenarios` command reports the scenario setup information for multicorner-multimode designs. The scenario specific information includes the logic library used, the operating condition, and TLUPlus files.

The following example shows a report generated by the `report_scenarios` command:

```
*****
Report : scenarios
Design : DESIGN1
scenario(s) : SCN1
Version: ...
Date   : ...
*****
```

All scenarios (Total=4): SCN1 SCN2 SCN3 SCN4

```
All Active scenarios (Total=1): SCN1
Current scenario      : SCN1

Scenario #0: SCN1 is active.
Scenario options:
Has timing derate: No
Library(s) Used:
  technology library name (File: library.db)

Operating condition(s) Used:
  Analysis Type      : bc_wc
  Max Operating Condition: library:WCCOM
  Max Process        : 1.00
  Max Voltage        : 1.08
  Max Temperature: 125.00
  Min Operating Condition: library:BCCOM
  Min Process        : 1.00
  Min Voltage        : 1.32
  Min Temperature: 0.00

Tlu Plus Files Used:
  Max TLU+ file: tlu_plus_file.tf
  Tech2ITF mapping file: tf2itf.map
```

Reporting Examples for Multicorner-Multimode Designs

This section contains report examples for some of the report commands used in multicorner-multimode designs.

report_scenarios

The `report_scenarios` command reports the scenario setup information for multicorner-multimode designs. This command reports all the defined scenarios. The scenario-specific information includes the logic library used, the operating condition, and the TLUPlus files.

The following example shows a report generated by the `report_scenarios` command:

```
*****
Report : scenarios
Design : DESIGN1
scenario(s) : SCN1
Version: ...
Date   : ...
*****

All scenarios (Total=4): SCN1 SCN2 SCN3 SCN4
All Active scenarios (Total=1): SCN1
Current scenario      : SCN1

Scenario #0: SCN1 is active.
Scenario options:
```

```

Has timing derate: No
Library(s) Used:
    technology library name (File: library.db)

Operating condition(s) Used:
    Analysis Type      : bc_wc
    Max Operating Condition: library:WCCOM
    Max Process        : 1.00
    Max Voltage         : 1.08
    Max Temperature: 125.00
    Min Operating Condition: library:BCCOM
    Min Process        : 1.00
    Min Voltage         : 1.32
    Min Temperature: 0.00

Tlu Plus Files Used:
    Max TLU+ file: tlu_plus_file.tf
    Tech2ITF mapping file: tf2itf.map

```

report_power

The `report_power` command supports the `-scenarios` option. Without the `-scenarios` option, only the current scenario is reported. To report power information for all scenarios, use the `report_power -scenarios [all_scenarios]` command.

Note:

In the multicorner-multimode flow, the `report_power` command does not perform clock tree estimation. The command reports only the netlist power in this flow.

The following example shows the report generated by the `report_power -scenarios` command.

```

*****
Report : power
Design : Design_1
Scenario(s): s1
Version: ...
Date   : ...
*****

Library(s) Used: slow (File: slow.db)

Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW      (derived from V,C,T units)
    Leakage Power Units = Unitless

```

Warning: Could not find correlated power. (PWR-725)

Power Breakdown

Cell	Driven Internal Power (mW)	Net Switching Power (mW)	Tot Power (% Cell/Tot)	Dyn Power (mW)	Cell Leakage Power (nW)
Netlist Power	4.8709	1.2889	6.160e+00 (79%)	1.351e+05	
Estimated Clock Tree Power	N/A	N/A	(N/A)		N/A

Script Example for Multicorner-Multimode Flow

[Example 47](#) shows a basic script example for the multicorner-multimode flow.

Example 47 Basic Script to Run a Multicorner-Multimode Flow

```
#.....path settings.....
set search_path ". $DESIGN_ROOT $lib_path/dbs \
$lib_path/mwlibs/macros/LM"
set target_library "stdcell.setup.ftyp.db \
stdcell.setup.typ.db stdcell.setup.typhv.db"
set link_library [concat * $target_library \
setup.ftyp.130v.100c.db setup.typhv.130v.100c.db \
setup.typ.130v.100c.db]
set_min_library stdcell.setup.typ.db -min_version stdcell.hold.typ.db

#.....MW setup.....
#.....load design.....


create_scenario s1
set_operating_conditions WORST -library stdcell.setup.typ.db:stdcell_typ
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s1.sdc
set_scenario_options -scenarios s1-setup false -hold false \
-leakage_power true

create_scenario s2
set_operating_conditions BEST -library stdcell.setup.ftyp.db:stdcell_ftyp
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s2.sdc

create_scenario s3
set_operating_conditions NOM -library stdcell.setup.ftyp.db:stdcell_ftyp
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s3.sdc

set_active_scenarios {s1 s2}
report_scenarios
```

```
compile_ultra -scan -gate_clock
report_qor
report_constraint
report_timing -scenarios [all_scenarios]
.
.
.
insert_dft
.
.
.
compile_ultra -incremental
```

The multicorner-multimode design in [Figure 127](#) and the subsequent example scripts in [Example 48](#) and [Example 49](#) show how you define your power intent in the UPF file and define the scenarios for a multicorner-multimode multivoltage design.

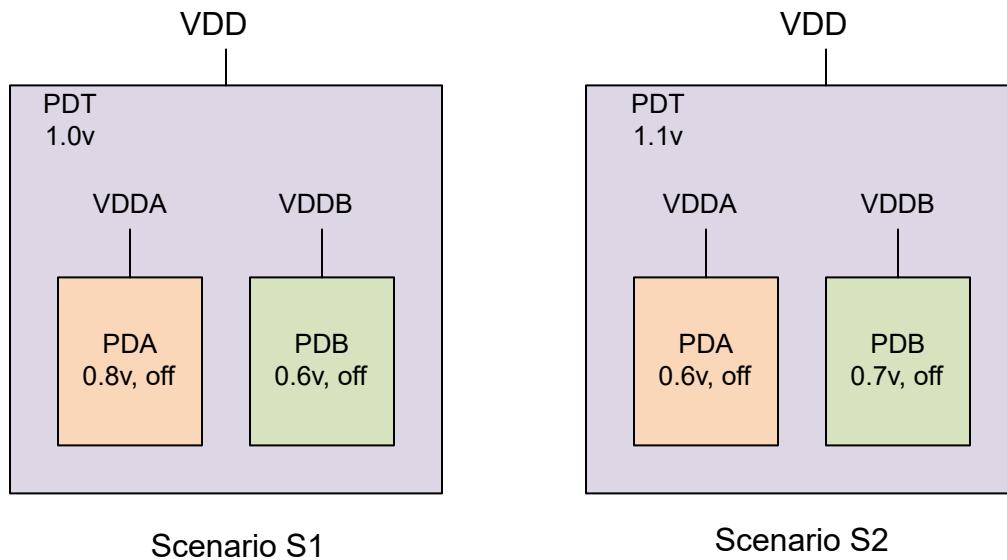
Multicorner-multimode multivoltage designs are useful in applications such as dynamic voltage and frequency scaling (DVFS). In hierarchical designs, the top-level design is generally optimized at a different voltage and in a different corner than the subdesigns of the hierarchy. The power intent specification can be for the entire design in a single UPF (Unified Power Format) file.

Standard cell and special cell libraries should be available to satisfy all voltages defined across multiple corners.

The design in [Figure 127](#) has two scenarios of operation, S1 and S2. In the scenario S1, the power domain PDT operates at 1.0V, while the power domain PDA operates at 0.8V or OFF and power domain PDB operates at 0.6V or OFF. In scenario S2, the power domain PDT operates at 1.1V, while the power domain PDA operates at 0.6V or OFF and power domain PDB operates at 0.7V or OFF.

Although the subdesigns operate at different voltages, you need only a single UPF file to specify your power intent for the entire design and all its subdesigns. The specific voltages set on the supply nets are scenario-specific and are set by using the `set_voltage` command in each scenario.

Figure 127 Multicorner-Multimode Design With Multivoltage



[Example 48](#) and [Example 49](#) show example scripts using the UPF flow for the multivoltage, multicorner-multimode design in [Figure 127](#).

Example 48 UPF File Describing Design Intent

```
Example UPF File
## Create Power Domains
create_power_domain PDT -include_scope
create_power_domain PDA -elements PD_PDA
create_power_domain PDB -elements PD_PDB

## Create Supply Nets
create_supply_net VDD -domain PDT
create_supply_net VDDA -domain PDA
create_supply_net VDDB -domain PDB
create_supply_net VSS -domain PDT
create_supply_net VSS -domain PDA -reuse
create_supply_net VSS -domain PDB -reuse

## Create Supply Ports
create_supply_port VDD
create_supply_port VDDA
create_supply_port VDDB
create_supply_port VSS

## Connect supply nets
connect_supply_net VDD -ports VDD
connect_supply_net VDDA -ports VDDA
connect_supply_net VDDB -ports VDDB
connect_supply_net VSS -ports VSS
```

```
### Adding port states
add_port_state VDD -state {HV1 1} -state {HV2 1.1}
add_port_state VDDA -state {LV1 0.8} -state {LV3 0.6} -state {OFF off}
add_port_state VDDB -state {LV2 0.9} -state {LV4 0.7} -state {OFF off}
create_pst top_pst -supplies "VDD VDDA VDDB"
add_pst_state PM1 -pst top_pst -state { HV1 LV1 LV3 }
add_pst_state PM2 -pst top_pst -state { HV1 LV1 OFF }
add_pst_state PM3 -pst top_pst -state { HV1 OFF LV3 }
add_pst_state PM4 -pst top_pst -state { HV1 OFF OFF }
add_pst_state PM5 -pst top_pst -state { HV2 LV2 LV4 }
add_pst_state PM6 -pst top_pst -state { HV2 LV2 OFF }
add_pst_state PM7 -pst top_pst -state { HV2 OFF LV4 }
add_pst_state PM8 -pst top_pst -state { HV2 OFF OFF }
```

Example 49 Tcl Script Example

```
load_upf example.upf ## UPF file defined above

create_scenario s1
read_sdc s1.sdc
set_operating_conditions WCCOM lib1.0V
set_voltage -object_list VDD 1.0
set_voltage -object_list VDDA 0.8
set_voltage -object_list VDDB 0.9
set_scenario_options -scenarios s1 -setup false -hold false \
-leakage_power true

create_scenario s2
read_sdc s2.sdc
set_operating_conditions BCCOM lib1.1V
set_voltage -object_list VDD 1.1
set_voltage -object_list VDDA 0.6
set_voltage -object_list VDDB 0.7
set_scenario_options -scenarios s2 -setup false -hold false \
-leakage_power true

compile_ultra -scan -gate_clock
```

Note:

The UPF file is not scenario-specific. As a result, the UPF file must contain port state definitions and power state tables for all the scenarios.

You use the `load_upf` command to read the UPF script shown in [Example 48](#).

Appendices

The following topics provide more information and examples about specific features:

- [Lower-Domain Boundary Support](#)
- [Integrated Clock-Gating Cell Example](#)
- [Attributes for Querying and Filtering](#)

A

Lower-Domain Boundary Support

By default, the Power Compiler tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, the tool can consider a power domain boundary to include the boundary of another domain contained in it.

For more information, see the following topics:

- [Overview of Power Domain Boundaries](#)
 - [Applying Isolation and Level-Shifter Strategies](#)
-

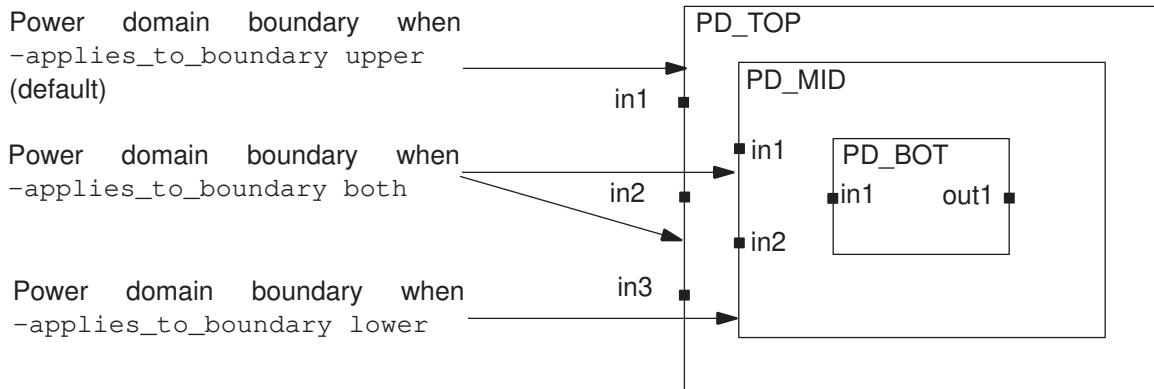
Overview of Power Domain Boundaries

By default, the Power Compiler tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801 (UPF) standard, the tool can consider a power domain boundary to include the boundary of another domain contained in it. You can specify the elements on the lower-domain boundary for level-shifter and isolation strategy definition, which gives you additional flexibility in selecting the location of the power management cells.

To extend the definition of the power domain boundary to the boundary of another power domain contained in it, use the `-applies_to_boundary` option in the `set_isolation` and `set_level_shifter` commands as shown in the following example:

```
set_isolation ISO1 -domain PD_TOP -applies_to_boundary lower
```

Figure 128 Definition of Power Domain Boundaries for PD_TOP



In [Figure 128](#), by default, the tool considers only in1, in2, and in3 ports of the PD_TOP domain to be at the domain boundary. This is also the case if the `-applies_to_boundary` option is set to `upper`.

When the `-applies_to_boundary` option is set to `both`, the tool considers the in1, in2, in3, MID/in1, and MID/in2 ports to be at the power domain boundary. However, the boundary does not extend to the interface of the BOT design or the PD_BOT power domain.

When the `-applies_to_boundary` option is set to `lower`, the tool considers the MID/in1 and MID/in2 ports to be at the power domain boundary.

Note that the boundary does not extend to the interface of the BOT design or the PD_BOT domain.

You can set the lower boundary of a power domain at the HighConn side of all hard macros included within the power domain, as follows:

- Set the `macro_as_domain_boundary` design attribute to `true`. This is a scope-level attribute that indicates whether macros at or below that scope need to be considered as design boundaries. Use the `-elements` option to apply the attribute to specific elements.
- Set the design attribute `lower_domain_boundary` to `true` for the top-level scope or a block-level scope.
- Specify the `-applies_to_boundary lower` or `-applies_to_boundary both` option of the `set_isolation`, `set_level_shifter`, or `set_repeater` commands.

In the following example, the lower domain boundary of power domain PD-TOP includes the ports of all macros, with the exception of macros contained in cell U2. As a result, the

tool inserts isolation cells at the input ports of all macros except macros contained in cell U2.

```
set_design_attributes -elements {.} \
    -attribute macro_as_domain_boundary true
set_design_attributes -elements {U2} \
    -attribute macro_as_domain_boundary false
create_power_domain PD_TOP -elements {.}
set_isolation ISO -domain PD_TOP -applies_to outputs \
    -applies_to_boundary both
```

When the `macro_as_domain_boundary` attribute is set to `true` for specific hard macros, the `terminal_boundary` attribute is allowed on the pins of the specified macros. In addition, you can specify pins or instances of the macros with the `-clamp_value` and `-repeater_supply` options of the `set_port_attributes` command.

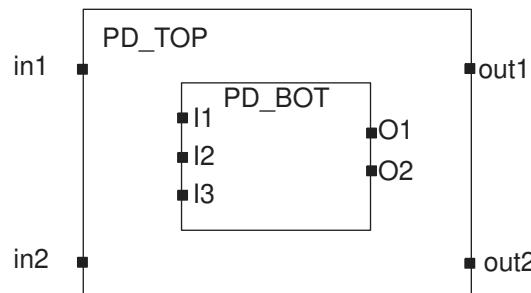
Applying Isolation and Level-Shifter Strategies

When you define the level-shifter and isolation strategies with the `-applies_to_boundary` option, you specify the domain and domain boundary to which the strategy applies.

When you specify the domain boundary in a strategy definition,

- The isolation and level-shifter strategy you specify applies to the pins of the domain boundary specified by the `-applies_to_boundary` option.
- The tool does not support the `-location fanout` option in the `set_isolation` and `set_isolation_control` commands.

Figure 129 Example of Nested Power Domains



For example, in the nested power domains shown in [Figure 129](#), the O1 and O2 output pins of the PD_BOT power domain are input pins for the strategies defined in the PD_TOP power domain. Similarly, I1, I2, and I3 input pins of the PD_BOT power domain are the output pins for the strategies defined in the PD_TOP power domain.

So, the isolation and level-shifter strategies that apply to the input pins of the top power domain also apply to the output pins of the root cell of the lower, nested power domain.

Similarly, the isolation and level-shifter strategies that apply to the output pins of the top power domain also apply to the input pins of the root cells of the lower, nested power domains.

For [Figure 129](#), the following table shows which pins apply to the specified boundary crossing for a strategy defined at PD_TOP.

Table 46 Pins Considered for Boundary Crossings

-applies_to_boundary	Pins
upper	in1, in2, out1, out2
lower	I1, I2, I3, O1, O2
both	in1, in2, out1, out2, I1, I2, I3, O1, O2

Specifying Domain Boundaries With the **-applies_to** Option

You can use the **-applies_to** option of the `set_isolation` and `set_level_shifter` commands to filter strategies to specific pins. For the nested domains in [Figure 129](#), the example in [Example 50](#) illustrates the lower-domain boundary with the **-applies_to** option.

Example 50 Lower-Domain Boundary Specification

```
create_power_domain PD_TOP -include_scope
set_isolation out_iso -domain PD_TOP -applies_to output \
    -applies_to_boundary lower
```

In [Example 50](#), the `out_iso` strategy defined for the PD_TOP power domain applies to the BOT/I1, BOT/I2, and BOT/I3 pins, which are the lower-domain boundary output pins of the PD_TOP power domain.

Example 51 Upper-Domain Boundary Specification

```
create_power_domain PD_TOP -include_scope
set_isolation both_iso -domain PD_TOP -applies_to output \
    -applies_to_boundary upper
```

In [Example 51](#), the boundary specification is `both` and `both_iso` strategy for PD_TOP applies to `out1` and `out2`, which are the upper-domain boundary output pins.

Defining Cell Placement With the **-location** Option

When you define a strategy using the `set_isolation` or `set_level_shifter` commands, the tool supports both the `-location parent` or `-location self` options. You can use

the location placement along with the `-applies_to` option for added flexibility in placing your isolation or level-shifter cells.

Figure 130 Isolation Cell Insertion With Domain Boundary Specified

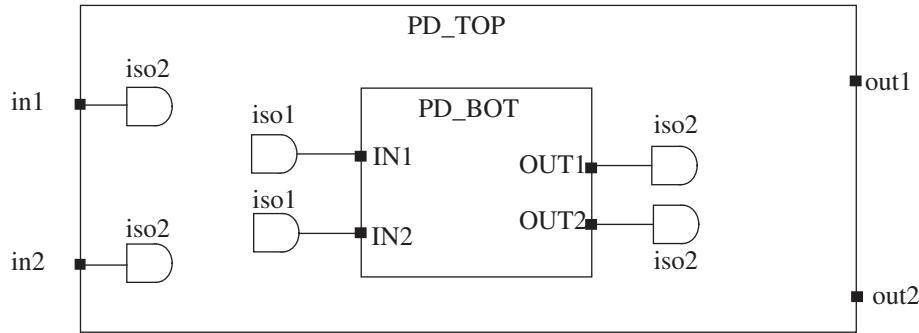


Figure 130 illustrates how the tool applies the isolation strategies in [Example 52](#).

Example 52 Example of Strategies With Different Port Directions

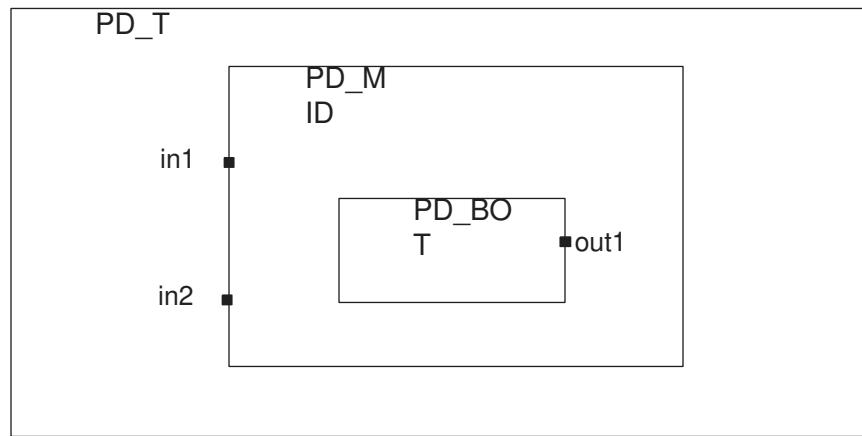
```

create_power_domain PD_TOP
create_power_domain PD_BOT
set_isolation ISO1 -domain PD_TOP -applies_to output \
                    -applies_to_boundary lower -location self
set_isolation ISO2 -domain PD_TOP -applies_to input \
                    -applies_to_boundary both -location self

```

Since the `-location` option in [Example 52](#) is set to `self`, the isolation cells are placed in domain PD_TOP.

Figure 131 Isolation Cell Insertion Example With the Parent Location



For the example in [Figure 131](#), if you have the following

```

create_power_domain PD_TOP
create_power_domain PD_MID -elements {A}
create_power_domain PD_BOT -elements {A/B}
set_isolation ISO1 -domain PD_MID -applies_to inputs \
                    -applies_to_boundary both -location self
  
```

The tool applies ISO1 strategy to ports A/in1, A/in2, and A/B/out1.

Suppose for the example in [Figure 131](#), you have the following

```

create_power_domain PD_TOP
create_power_domain PD_MID -elements {A}
create_power_domain PD_BOT -elements {A/B}
set_isolation ISO1 -domain PD_MID -applies_to inputs \
                    -applies_to_boundary both -location parent
  
```

The tool implements ISO1 only on A/in1 and A/in2 at the parent location PD_TOP. However, the tool does not apply the strategy to port B/out1 because the parent domain for PD_MID is PD_TOP and the tool cannot place the isolation cell there. In this case, the tool issues a warning message.

B

Integrated Clock-Gating Cell Example

This appendix contains an example .lib description of an integrated clock-gating cell and some schematic examples of rising (positive) and falling (negative) edge integrated clock-gating cells.

- [Library Description](#)
 - [Example Schematics](#)
-

Library Description

[Example 53](#) is a description of an integrated clock-gating cell that demonstrates the following features:

- The `clock_gating_integrated_cell` attribute
- Appropriate clock-gating attributes on three pins
- Setup and hold arc on enable pin (EN) with respect to the clock pin (CP)
- Combinational arcs from enable pin (EN) and clock pin (CP) to the output pin (Z)
- State table and state function on the output pin (Z)
- Internal power table

Example 53 HDL Description, Integrated Clock-Gating Cell

```
cell(CGLP) {
    area : 1;
    clock_gating_integrated_cell : "latch_posedge";
    dont_use : true;
    statetable(" CP EN ", "IQ ") {
        table : " L  L  : - : L , \
                  L  H  : - : H , \
                  H  -  : - : N ";
    }
    pin(IQ) {
        direction : internal;
        internal_node : "IQ";
    }
    pin(EN) {
```

Appendix B: Integrated Clock-Gating Cell Example

Library Description

```

direction : input;
capacitance : 0.017997;
clock_gate_enable_pin : true;
timing() {
    timing_type : setup_rising;
    intrinsic_rise : 0.4;
    intrinsic_fall : 0.4;
    related_pin : "CP";
}
timing() {
    timing_type : hold_rising;
    intrinsic_rise : 0.4;
    intrinsic_fall : 0.4;
    related_pin : "CP";
}
pin(CP) {
    direction : input;
    capacitance : 0.031419;
    clock_gate_clock_pin : true;
    min_pulse_width_low : 0.319;
}
pin(Z) {
    direction : output;
    state_function : "CP * IQ";
    max_capacitance : 0.500;
    max_fanout : 8;
    clock_gate_out_pin : true;
    timing() {
        timing_sense : positive_unate;
        intrinsic_rise : 0.48;
        intrinsic_fall : 0.77;
        rise_resistance : 0.1443;
        fall_resistance : 0.0523;
        rise_resistance : 0.1443;
        fall_resistance : 0.0523;
        slope_rise : 0.0;
        slope_fall : 0.0;
        related_pin : "CP";
    }
    timing() {
        timing_sense : positive_unate;
        intrinsic_rise : 0.22;
        intrinsic_fall : 0.42;
        rise_resistance : 0.1443;
        fall_resistance : 0.0523;
        slope_rise : 0.0;
        slope_fall : 0.0;
        related_pin : "EN";
    }
    internal_power (){
        rise_power(li4X3){
            index_1("0.0150, 0.0400, 0.1050, 0.3550");
    }
}

```

Appendix B: Integrated Clock-Gating Cell Example

Library Description

```

        index_2("0.050, 0.451, 1.501");
        values("0.141, 0.148, 0.256", \
        "0.162, 0.145, 0.234", \
        "0.192, 0.200, 0.284", \
        "0.199, 0.219, 0.297");
    }
    fall_power(li4X3){
        index_1("0.0150, 0.0400, 0.1050, 0.3550");
        index_2("0.050, 0.451, 1.500");
        values("0.117, 0.144, 0.246", \
        "0.133, 0.151, 0.238", \
        "0.151, 0.186, 0.279", \
        "0.160, 0.190, 0.217");
    }
    related_pin : "CP EN" ;
}
}
}

```

When creating your model, examine whether it includes all the `clock_gate` attributes on both the cell and on the pins. Some of the Power Compiler commands require these attributes to recognize the functionality of the cell. The TestMAX DFT tool does not recognize this cell. If these attributes are not included, an error message displays. Include the following attributes in your model:

- `clock_gating_integrated_cell`
- `clock_gate_test_pin`
- `clock_gate_enable_pin`
- `clock_gate_out_pin`
- `clock_gate_clock_pin`

Library Compiler can interpret the functionality of the integrated clock-gating cell directly from the state table and state function. The following example shows the `clock_gating_integrated_cell` attribute with a `generic` value:

```

cell(CGLP) {
area : 1;
clock_gating_integrated_cell : "generic";
dont_use : true;
statetable(" CP EN ", "IQ ") {
table : " L L : - : L , \
L H : - : H , \
H - : - : N ";
}
pin(IQ) {
direction : internal;
internal_node : "IQ";
}

```

```
... ...
pin(Z) {
    direction : output;
    state_function : "CP * IQ";
    max_capacitance : 0.500;
    max_fanout : 8;
    clock_gate_out_pin : true;
    timing() {
        ...
    }
}
```

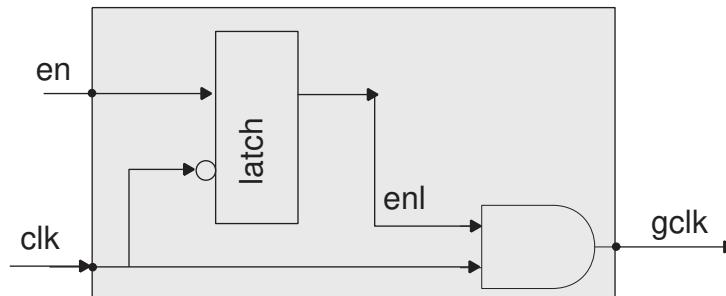
Example Schematics

This section contains example schematics of latch-based and latch-free clock-gating styles for rising- and falling-edge-triggered logic. These are a subset of integrated clock-gating cells supported by the Power Compiler tool.

Rising-Edge Latch-Based Integrated Cells

[Figure 132](#) displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs.

Figure 132 Rising-Edge Latch-Based Integrated Cell (latch_posedge)



[Figure 133](#) displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 133 Rising-Edge Latch-Based Integrated Cell With Pre-Control (latch_posedge_precontrol)

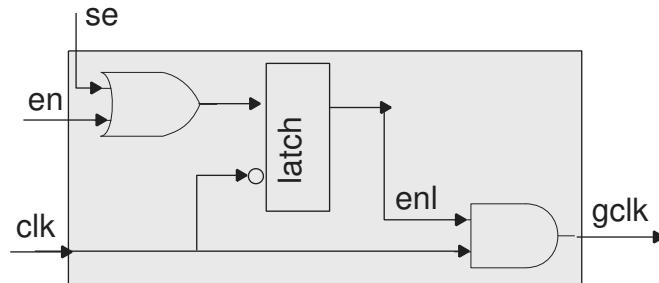


Figure 134 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 134 Rising-Edge Latch-Based Integrated Cell With Post-Control (latch_posedge_postcontrol)

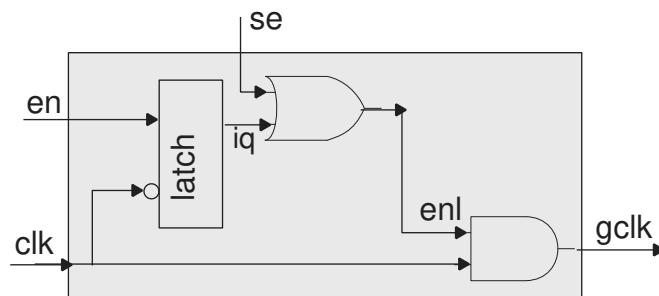


Figure 135 Rising Edge Latch Based Integrated Cell With Post-Control Observable Point (latch_posedge_postcontrol)

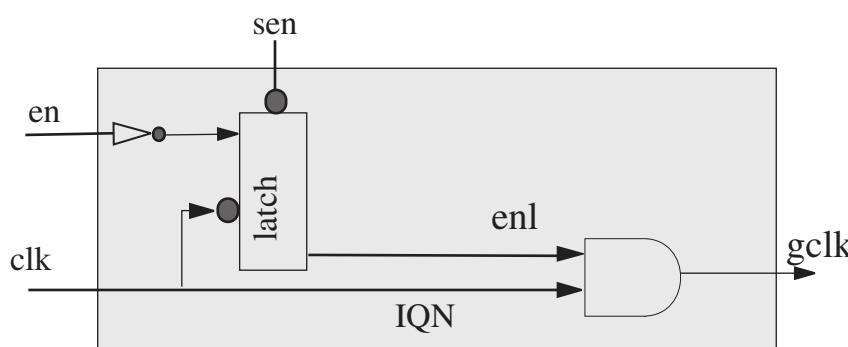


Figure 136 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 136 Rising-Edge Latch-Based Integrated Cell With Pre-Control Observable Point (latch_posedge_precontrol_obs)

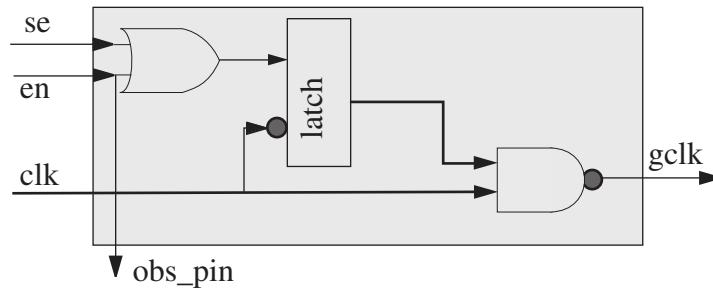
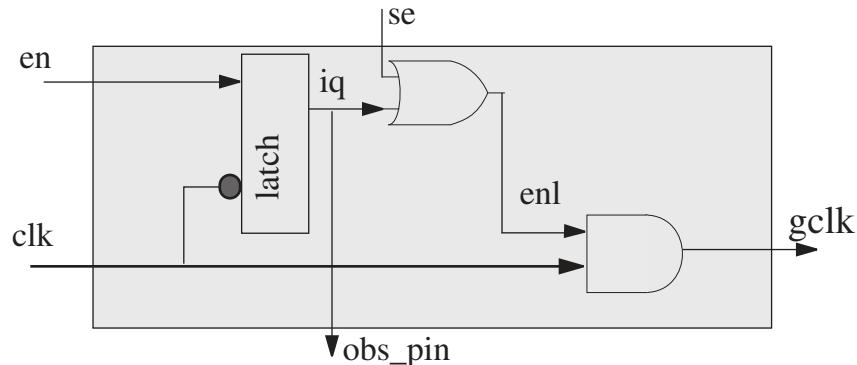


Figure 137 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 137 Rising-Edge Latch-Based Integrated Cell With Post-Control Observable Point (latch_posedge_postcontrol_obs)



Rising-Edge Latch-Free Integrated Cells

Figure 138 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs.

Figure 138 Rising-Edge Latch-Free Integrated Cell (none_posedge)

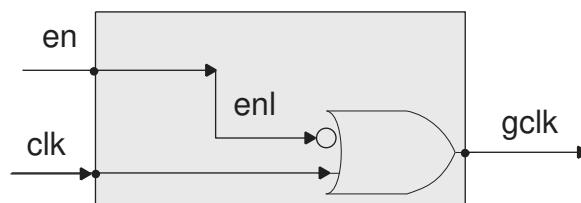


Figure 139 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 139 Rising-Edge Latch-Free Integrated Cell With Control (none_posedge_control)

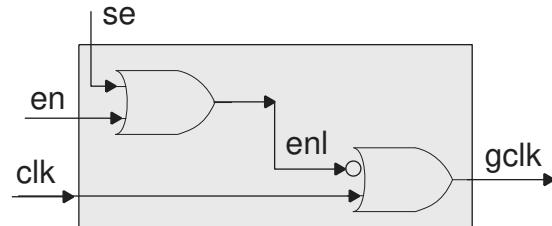
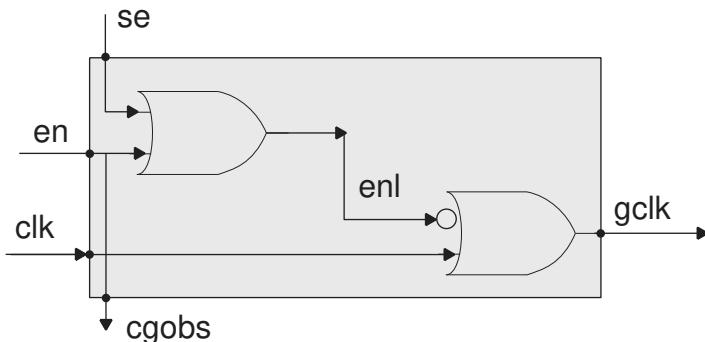


Figure 140 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 140 Rising-Edge Latch-Free Integrated Cell With Control Observable Point (none_posedge_control_obs)



Falling Edge Latch-Based Integrated Cells

Figure 141 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs.

Figure 141 Falling-Edge Latch-Based Integrated Cell (latch_negedge)

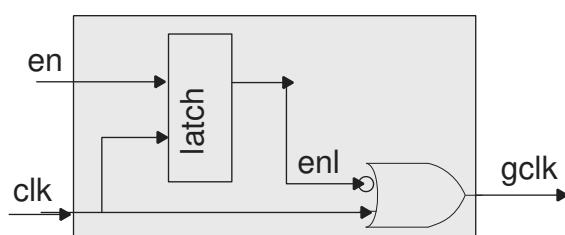


Figure 142 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 142 Falling-Edge Latch-Based Integrated Cell With Pre-Control Observable Point (latch_negedge_precontrol)

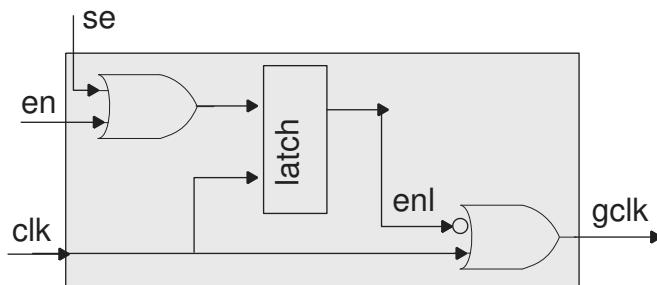


Figure 143 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 143 Falling-Edge Latch-Based Integrated Cell With Post-Control Observable Point (latch_negedge_postcontrol)

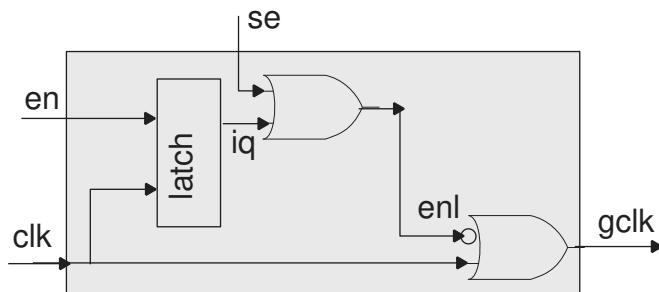


Figure 144 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 144 Falling-Edge Latch-Based Integrated Cell With Pre-Control Observable Point (*latch_negedge_precontrol_obs*)

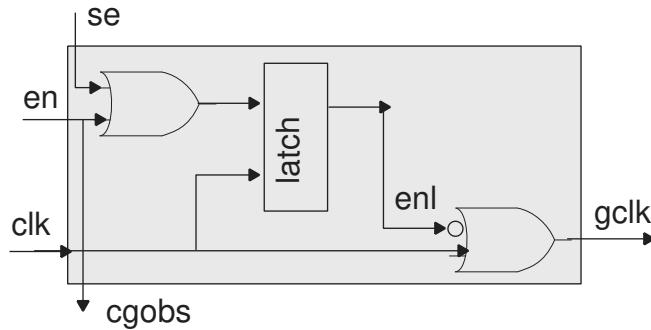
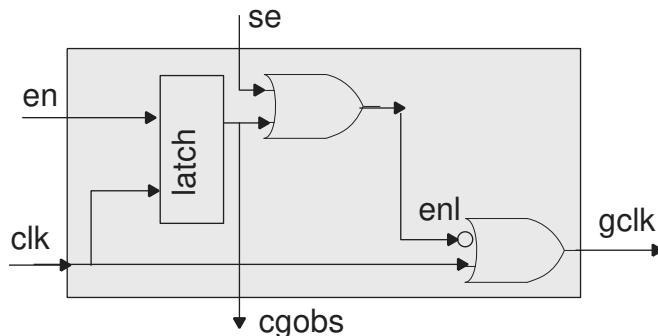


Figure 145 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 145 Falling-Edge Latch-Based Integrated Cell With Post-Control Observable Point (*latch_negedge_postcontrol_obs*)



Falling-Edge Latch-Free Integrated Cells

Figure 146 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs.

Figure 146 Falling-Edge Latch-Free Integrated Cell (*none_negedge*)

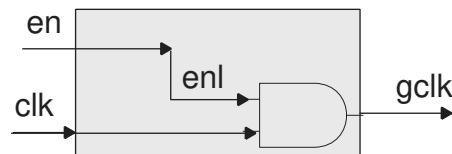


Figure 147 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure 147 Falling-Edge Latch-Free Integrated Cell With Control (none_negedge_control)

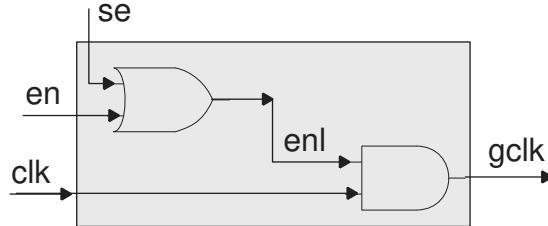
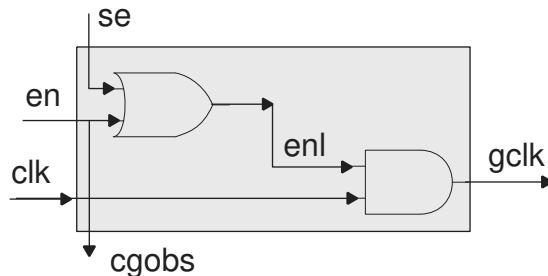


Figure 148 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure 148 Falling-Edge Latch-Free Integrated Cell With Control Observable Point (none_negedge_control_obs)



C

Attributes for Querying and Filtering

This appendix describes derived attributes that you can use in scripts to view and filter design objects related to clock gating for power optimization.

The derived attributes described in this appendix are read-only properties that the Power Compiler tool automatically assigns to designs, cell, and pins based on other attributes or the netlist configuration.

At times, you might want to view and use design objects according to their attributes. For example, you might want to filter for cells that are integrated clock gates (the `is_icg` attribute). Alternatively, your queries might be required for back end processes such as clock tree synthesis in which fanout considerations have priority.

- [Derived Attribute Lists](#)
 - [Usage Examples](#)
-

Derived Attribute Lists

You can query for the following derived attributes assigned by the tool. Specify `man power_attributes` in `dc_shell` to view a list of these attributes. [Table 47](#) and [Table 48](#) show the derived attributes for designs and cells, respectively.

Table 47 Derived Attributes for Designs

Name	Type	Description
<code>is_clock_gating_design</code>	Boolean	<code>true</code> if the design is a clock-gating design
<code>is_clock_gating_observability_de</code> <code>sign</code>	Boolean	<code>true</code> if the design is a clock-gating observable design

Table 48 Derived Attributes for Cells

Name	Type	Description
<code>is_clock_gate</code>	Boolean	<code>true</code> if the cell is a clock gate
<code>is_icg</code>	Boolean	<code>true</code> if the cell is an integrated clock gate

Table 48 Derived Attributes for Cells (Continued)

Name	Type	Description
is_gicg	Boolean	true if the cell is a generic integrated clock gate
is_latch_based_clock_gate	Boolean	true if the cell is a latch-based clock-gating cell
is_latch_free_clock_gate	Boolean	true if the cell is a latch-free clock-gating cell
is_positive_edge_clock_gate	Boolean	true if the cell is a positive edge clock gate
is_negative_edge_clock_gate	Boolean	true if the cell is a negative edge clock gate
clock_gate_has_precontrol	Boolean	true if the cell is a clock gate with a pre-latch control point
clock_gate_has_postcontrol	Boolean	true if the cell is a clock gate with a post-latch control point
clock_gate_has_observation	Boolean	true if the cell is a clock gate with observation point
is_clock_gated	Boolean	true if the cell is a clock-gated register or clock gate
clock_gating_depth	integer	number of clock gates on the clock path to this cell; -1 if not a clock gate or register
clock_gate_level	integer	position in a multistage clock tree: number of clock gates on the longest branch in the fanout of this cell; -1 if not a clock gate
clock_gate_fanout	integer	number of registers and clock gates in the direct fanout of the clock gate; -1 if not a clock gate
clock_gate_register_fanout	integer	number of registers in the direct fanout of the clock gate; -1 if not a clock gate
clock_gate_multi_stage_fanout	integer	number of clock gates in the direct fanout of the clock gate; -1 if not a clock gate
clock_gate_transitive_register_fanout	integer	number of registers in the transitive fanout of the clock gate; -1 if not a clock gate
clock_gate_module_fanout	integer	number of modules in the local fanout of the clock gate; -1 if not a clock gate

For hierarchical clock-gating cells, the derived clock-gating attributes only work when applied to the hierarchical clock-gate wrapper. If you apply an attribute to the leaf cell

of a discrete clock gate or a leaf integrated clock gate, the attribute returns false for Boolean attributes, -1 for integer attributes, or an empty string for string attributes. The only exception to this rule is the `is_icg` attribute; this attribute is true when applied to a leaf integrated clock gate contained within a hierarchical clock gate wrapper but false when applied to that wrapper. This behavior allows you to recognize the actual integrated clock-gating cell, not the hierarchical wrapper.

Table 49 Derived Attributes for Pins

Name	Type	Description
<code>is_clock_gate_enable_pin</code>	Boolean	<code>true</code> if the pin is a clock-gate enable input
<code>is_clock_gate_clock_pin</code>	Boolean	<code>true</code> if the pin is a clock-gate clock input
<code>is_clock_gate_output_pin</code>	Boolean	<code>true</code> if the pin is a clock-gate gated-clock output
<code>is_clock_gate_test_pin</code>	Boolean	<code>true</code> if the pin is a clock-gate scan-enable or test-mode input
<code>is_clock_gate_observation_pin</code>	Boolean	<code>true</code> if the pin is a clock-gate observation point

Usage Examples

You can query the attributes described in the previous section using the `get_attribute`, `get_designs`, `get_cells`, `get_pins`, and `all_clock_gates` commands. You can also use these commands with the `-filter` option.

The following examples show how the attributes might appear in scripts.

To gather all the clock gates specific to a clock “clk”:

```
all_clock_gates -clock [ get_clocks clk]
```

The `all_clock_gates` command creates a collection of clock-gating cells or pins that satisfy the parameters you set. Additional options allow you to filter for enable, clock, and gated-clock pins; `scan_enable` or `test_mode` pins; and observation pins. For more information, see the man page.

To filter out the multistage clock-gating cell associated with the clock “clk”:

```
set multi_stage_cg [filter [all_clock_gates -clock [get_clocks clk]]] \
    "@clock_gate_level >0" ]
```

To retrieve the number of fan outs of a clock-gating cell:

```
get_attribute [ get_cells top/clk_gate_1 ] \
    clock_gate_fanout
```

To gather a collection of clock-gating cells with a precontrol point and a fanout greater than four:

```
set CG_collection [filter [all_clock_gates] \
    "@clock_gate_has_precontrol== \
    true && @clock_gate_fanout > 4"]
```

To gather a collection of clock-gating designs (the wrapper design where the clock-gating cells reside):

```
set CG_designs [get_designs -filter \
    "@is_clock_gating_design==true"]
```