

PrimePower Manual

Version Y-2006.06, June 2006

Comments?

Send comments on the documentation by going to <http://solvnnet.synopsys.com>, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2006 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSiM, HSPICE, Hypermodel, iN-Phase, in-Sync, Leda, MAST, Meta, Meta-Software, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, RapidScript, Saber, SiVL, SNUG, SolvNet, Superlog, System Compiler, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAI, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, Direct RTL, Direct Silicon Access, Discovery, DW8051, DWPCI, Dynamic Model Switcher, Dynamic-Macromodeling, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, Galaxy, Gattran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSiM^{plus}, HSPICE-Link, i-Virtual Stepper, iN-Tandem, Integrator, Interactive Waveform Viewer, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Libra-Visa, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Protocol Compiler, PSMGen, Raphael, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

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

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

PrimePower Manual, version Y-2006.06

Contents

About This Manual	xxii
Customer Support	xxv
1. Introduction	
Overview	1-1
PrimePower Simulation Flow	1-3
Switching Activity Generation	1-5
Power Analysis.	1-6
Command Interfaces	1-7
PrimeTime Compatibility	1-7
Power Analysis Overview	1-8
Power Types	1-9
Static Power	1-9
Dynamic Power	1-9
Switching Power	1-10
Internal Power	1-10

Calculating Power	1-12
Calculating Leakage Power	1-12
Calculating Dynamic Power	1-15
Event-based Analysis	1-15
Calculating Internal Power	1-16
Calculating Switching Power	1-19
Statistical Activity Based Power Analysis	1-20
Switching Power Calculation	1-22
Modeling Special Conditions	1-22
Glitch Power	1-23
Z States	1-23
X States	1-24
Operating Condition Effects	1-24
Dual VT Libraries	1-26
 2. Starting and Using PrimePower	
Overview	2-1
Starting PrimePower	2-2
Starting a PrimePower Session	2-2
Ending a PrimePower Session	2-3
Licenses	2-3
Startup Files	2-4
Command Log File	2-5
Using pp_shell	2-6

Entering Commands Interactively	2-6
Using Command-Line Editing	2-7
Choosing an Editor Mode.	2-7
Using Command-Line Editing.	2-8
Command Scripts	2-9
Running Scripts in Batch Mode	2-10
Command Results	2-10
Variables	2-12
Tcl Interface	2-14
Using the PrimePower Command Template	2-14
PrimePower Documentation and Online Help	2-15
Using the help Command	2-15
Using the man Command	2-18
Man Pages for Commands.	2-19
Man Pages for Variables	2-20
Man Pages for Messages.	2-20
PrimePower Documentation	2-21
SolvNet Articles	2-22
Synopsys Online Documentation (SOLD)	2-23
Analysis Flow in PrimePower	2-24
Reading the Design Data.	2-26
Checking the Design Setup	2-27
Providing Circuit Activity	2-28
Checking the Activity Files.	2-30
Specifying the Environment and Analysis Conditions and Calculate Power	2-31

Operating Conditions	2-31
Parasitics	2-32
Analysis Parameters	2-33
Performing a Full Analysis	2-33
Examining the Results	2-34
Saving and Restoring PrimePower Sessions.	2-36
3. PrimePower Tutorial	
Test Case	3-1
Power Analysis With VCD	3-2
Related Files	3-2
Technology Library	3-2
Gate-level Netlist	3-2
VCD File	3-3
Parasitic File	3-3
PrimePower Script	3-3
Steps for Analyzing Power	3-4
Working Directory is pp_vcd.	3-4
Checking the PrimePower installation	3-5
Running PrimePower	3-5
Reviewing the Power Report	3-6
Viewing the Power Waveform.	3-6
Power Analysis With SAIF	3-7
Related Files	3-8
The SAIF File	3-8
PrimePower Script File	3-8
Steps for Analyzing Power	3-9

Running PrimePower	3-10
Reviewing the Power Report	3-10
Viewing the Waveform	3-10
Vector Free Power Analysis	3-11
Related Files	3-12
PrimePower Script	3-12
Steps For Analyzing Power	3-13
Running PrimePower	3-14
Viewing the Power Report	3-14
RTL VCD Power Analysis	3-14
PrimePower Script	3-15
Steps for Analyzing Power	3-16
Running PrimePower	3-16
Viewing the Power Report	3-16
4. Graphical User Interface	
Analysis Flow With the GUI	4-1
Starting and Stopping a GUI Session	4-2
Top-Level GUI Window	4-3
Manipulating Windows	4-4
Window Types	4-8
Toolbars	4-8
Selecting Objects	4-9
Menu Commands	4-9
Console	4-11
Design Status	4-13

Histograms	4-14
Hierarchy Browser	4-15
5. Design Data	
Overview	5-1
Search Path and Link Path	5-2
Reading Design and Library Data in .db Format	5-4
Reading Design Data in .db Format	5-4
Reading Design Data in the .ddc Format	5-5
Reading Verilog and VHDL Design Data	5-6
Using the PrimePower Verilog Reader	5-6
Optional Preprocessor	5-7
Assign Statements and Synonyms	5-8
PrimePower Verilog Reader Limitations	5-9
Using the HDL Compiler Verilog Reader	5-10
Setup Files for the HDL Compiler Verilog Reader	5-10
Reading VHDL Design Files	5-12
Setup Files for the VHDL Compiler	5-13
Setting the Current Design and Current Instance	5-14
Listing Design and Library Information	5-17
Linking the Design	5-19
Per-instance Link Paths	5-20
Link Errors	5-21
Design Objects	5-22

State Objects	5-23
Netlist Objects	5-24
Design Information Reports.	5-25
Displaying the Design Attributes	5-26
Displaying Clock Information	5-28
Displaying Cell Information	5-29
Port/Net Reports	5-31
Displaying Port Information	5-31
Displaying Net Information	5-34
Bus Report	5-37
 6. Design Parasitics	
Overview	6-1
Modeling Wire Loads.	6-2
Calculating the Capacitance of a Net.	6-3
Choosing a Wire Load Model.	6-6
Wire Load Model Modes	6-7
Setting a Wire Load Model	6-10
Setting a Wire Load Mode	6-12
Setting a Wire Load Minimum Block Size	6-12
Setting a Wire Load Selection Group.	6-13
Defining Wire Load Models or Selection Groups for Hierarchical Cells.	6-14
Selecting the Wire Load Model Automatically	6-14
Defining the Smallest Block Size	6-15
Reporting Wire Load Models	6-17

Specifying Port/Net Loads	6-21
Back-annotated Parasitics	6-22
Parasitic Reports	6-22
Wire Load Information for the Current Design	6-23
Wire Load Information on Ports	6-23
Verifying the Wire Capacitance	6-24
Checking for Unannotated Nets	6-25
 7. Setting Design Constraints	
Design Conditions	7-1
Operating Conditions	7-2
Setting Operating Conditions	7-3
Operating Condition Information	7-5
Input Transition Times	7-5
Verifying Input Transition Times	7-5
Specifying Clock Characteristics	7-6
Creating a Clock	7-7
Automatically Identifying Clock Sources	7-8
Creating a Virtual Clock	7-8
Selecting Clock Objects	7-9
Applying Commands to All Clocks	7-10
Removing Clock Objects	7-10
Synopsys Design Constraints Formatted Script Files	7-11
Checking the Syntax of the SDC File	7-11

Reading an SDF File	7-12
Annotating Timing From a Subdesign Timing File	7-12
Annotating Load Delay	7-12
Annotating Timing Checks	7-13
Reading the File	7-13
Removing Annotated Timing Checks and Delays	7-15
Reporting Delay Back-Annotation Status	7-15
Reporting Annotated or Nonannotated Delays	7-16
Reporting Annotated or Nonannotated Timing Checks	7-17
Annotating Conditional Delays From SDF	7-18
Multi-VDD	7-21
Reporting Power Calculation	7-22
Leakage Power Calculation	7-23
Internal Power	7-24
Net Switching Power	7-25
 8. Statistical Activity Based Power Analysis	
Overview	8-1
Statistical Activity Based Analysis	8-2
Switching Activity	8-3
Vector-free/Default Activity	8-5
Toggle Rate	8-5
No Clock Specification	8-5
With Clock Specification	8-6

Static Probability	8-7
Annotating Switching Activity With set_switching_activity.	8-8
Annotating Switching Activity Using SAIF Files.	8-14
Generating SAIF	8-15
Using HDL Simulation	8-15
Using the vcd2saif Utility	8-15
RTL SAIF	8-16
Gate-level SAIF	8-17
SAIF File Commands.	8-17
Reading SAIF Files Using merge_saif	8-20
Removing Switching Activity Annotation	8-23
Writing SAIF File Out of PrimePower	8-25
Estimating Unannotated Switching Activity	8-25
Annotating Design Nets with Default Switching Activity Values	8-26
Propagating Switching Activity.	8-28
Deriving SDPD Switching Activity	8-28
Correlation.	8-29
Complex Cells	8-29
Debugging the Annotation.	8-30
Case Analysis	8-32
Performing Case Analysis	8-33
Constant Propagation Based on Cell Logic.	8-37
Setting Case Analysis Values	8-38
Reporting Case Analysis Values	8-40

Removing Case Analysis Values	8-41
Constant Propagation Log File	8-42
Power Calculation	8-45
Power Calculation Variables	8-45
Calculate Power Options	8-47
Average Cycle Waveform Data	8-48
 9. Event-based Power Analysis	
Overview	9-1
Event-based Power Analysis	9-2
Activity File Formats	9-2
Mapping the Testbench Instance to the Design Module	9-4
Handling Large Activity Files	9-4
Handling Activity Data From Mixed-language Simulations	9-5
Handling VCD Files With Non-module Scopes	9-7
Processing Multiple Activity Files	9-7
Zero-delay VCD Files	9-8
Debugging Problems With the VCD File	9-9
Power Calculation	9-10
Variables for Power Calculation	9-11
Event Based Initial X-State Handling	9-13
Unmatched-States	9-14
Using the calculate_power Command Options	9-15
Default Behavior	9-15
Time/Instance/Mode Options	9-15

Checking for Out-of-Range Loads and Transition Times	9-16
Interrupting and Resuming Power Calculation	9-18
Power Waveforms	9-20
Peak Power	9-20
Options for Power Waveform Calculation Algorithm	9-21
10. Generating Reports	
Overview	10-1
Default Power Report	10-2
New Power Report	10-6
Optional Reports	10-8
Cell-type Power Dissipation Report	10-8
Leaf Power Dissipation Report	10-9
Net Power Report	10-11
Waveform Report	10-12
Viewing Waveform Data	10-14
Scaling Waveform Data	10-14
Clock-related Power	10-15
Clock Tree Estimation	10-15
Clock Tree Power	10-19
Register Power	10-20
Related Clock Power	10-20
Scan Chain Power	10-20
PrimeRail Reports	10-21
Custom Report Generation	10-21

Statistics	10-25
11. Multi-rail Power Analysis	
Multi-voltage Design Power Report	11-3
Multi-voltage Cells	11-6
Reporting Power Rail Mapping	11-7
Power-gating Support	11-7
Monitoring the Power Off Signal	11-8
Sample Script	11-9
Dealing with the Multi-voltage Libraries	11-10
Multiple NLDM Library	11-10
SPDM Library	11-11
12. Command Interface and Tcl	
Overview	12-1
Tcl Syntax and PrimePower Commands	12-2
Abbreviating Commands and Options	12-3
Using Tcl Special Characters	12-5
Basic System Commands	12-6
Using the Result of a Command	12-7
Using Built-In Commands	12-7
PrimePower Extensions and Restrictions	12-8
Redirecting and Appending Output	12-9
Using the redirect Command	12-10

Getting the Result of Redirected Commands	12-11
Using the Redirection Operators	12-11
Using Command Aliases	12-12
Interrupting Commands	12-13
Using Scripts	12-14
Adding Comments	12-14
Controlling Script Processing When Errors Occur	12-15
Finding Scripts Using the search_path Variable	12-16
Logging Source File Contents	12-16
Listing and Running Previous Commands	12-16
Rerunning Previously Entered Commands	12-18
Modifying the Previous Command Before Rerunning	12-19
Suppressing Warning and Informational Messages	12-19
Using Variables	12-20
Listing Variables	12-22
User-Defined Variables	12-22
Using Collections	12-24
Creating Collections	12-25
Primary Commands That Create Collections	12-26
Other Commands That Create Collections	12-28
Saving Collections	12-28
Querying Objects	12-30
Implicit Query	12-30
Using Wildcard Characters	12-31
Filtering Collections	12-33

Using -filter Options	12-33
Using Filter Operators	12-34
Relation Rules	12-36
Using Pattern Match Filter Operators	12-36
Using the filter_collection Command	12-37
Using Implicit Collections as Arguments	12-38
Iterating Over the Elements of a Collection	12-38
Removing From and Adding to a Collection.	12-39
Sorting Collections	12-42
Using Collection Utility Commands	12-43
Determining the Size of a Collection	12-44
Comparing Collections	12-44
Copying Collections	12-45
Indexing Collections	12-46
Using Lists.	12-47
Using Other Utilities	12-49
Quoting Values.	12-50
Nesting Commands	12-51
Evaluating Expressions	12-51
Flow Control	12-52
Using the if Command	12-52
Using while Loops	12-53
Using for Loops	12-54
Iterating Over a List: foreach	12-55
Terminating a Loop	12-56
Using the switch Command	12-56

Using Procedures	12-56
Using the proc Command to Create Procedures	12-58
Programming Default Values for Arguments	12-60
Specifying a Varying Number of Arguments.	12-60
Displaying the Body of a Procedure.	12-61
Displaying the Formal Parameters of a Procedure	12-62
Tcl Advanced Features	12-63
Tcl Packages and Autoload	12-63
Regular Expressions	12-65
Using Regular Expressions With Implicit Collections.	12-66
Using Regular Expressions With Hierarchy	12-66
Anchoring Regular Expressions	12-67
Using Regular Expressions With Buses	12-67
Extending Procedures	12-68
Default Procedure Attributes	12-69
Changing the Aspects of a Procedure	12-70
Format for the -define_args Option	12-71
Parsing Arguments Passed in to a Tcl Procedure	12-74
TclPro Toolkit.	12-77
Installation Requirements	12-77
Checking the Syntax and Semantics of Your Scripts.	12-78
Limitations Using the Synopsys Syntax Checker.	12-78
Running the Synopsys Syntax Checker.	12-79
Bytecode-Compiled Files.	12-80
Debugging Scripts	12-81

Launching Prodebug From PrimePower	12-81
TclPro Limitations	12-82
Object Attributes	12-82
Using Attributes	12-83
Defining User Attributes	12-84
Importing User-Defined Attributes	12-85
Attribute Names and Usage	12-88
Appendix A. FAQ, Troubleshooting, and Known Limitations	
Frequently Asked Questions (FAQ)	A-1
Known Limitations	A-3
Index	

Preface

This preface contains the following sections:

- [About This Manual](#)
- [Conventions](#)
- [Customer Support](#)

About This Manual

This manual describes the PrimePower tool, its methodology, and its use. PrimePower is a dynamic, full-chip power analysis tool for complex multimillion-gate designs. Its high-capacity power analysis includes gate-level average and peak power verification.

PrimePower supports industry-standard synthesis libraries and comprises a powerful and flexible methodology that is fully integrated with existing design flows. It provides a high degree of accuracy, performance, ease of use and comprehensive power diagnostics.

Audience

PrimePower is intended for designers of high-performance ASIC and structured custom IC who need accurate power dissipation data for cell-based designs.

Related Publications

For additional information about PrimePower, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at <https://solvnet.synopsys.com/DocsOnWeb>
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at <http://mediadocs.synopsys.com>

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

- PrimeTime
- Power Compiler

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates command syntax.
<i>Courier italic</i>	Indicates a user-defined value in Synopsys syntax, such as <i>object_name</i> .
<i>Regular italic</i>	A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog statement.
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples.
Regular bold	User input that is not Synopsys syntax, such as a user name or password you enter in a GUI.
[]	Denotes optional parameters, such as <i>pin1 [pin2 ... pinN]</i>
...	Indicates that a parameter can be repeated as many times as necessary
	Indicates a choice among alternatives, such as <i>low medium high</i> (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
—	Connects terms that are read as a single term by the system, such as <i>set_annotated_delay</i>
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit>Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at <http://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to <http://solvnet.synopsys.com> (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr.
- Telephone your local support center.
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

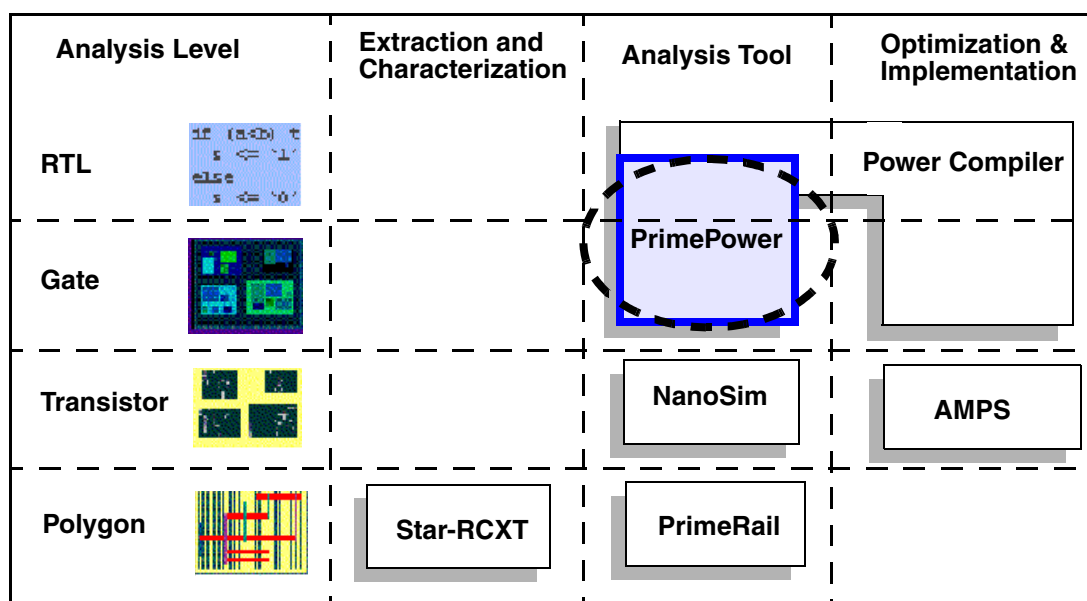
Introduction

PrimePower is a gate-level power analysis tool that accurately analyzes power dissipation of cell-based designs. It is intended as an advanced solution for ASIC and structured custom circuit designers who are developing products for power-critical applications such as portable computing and telecommunications.

Overview

PrimePower supports event-based and statistical activity based power analysis at the gate level. [Figure 1-1](#) shows how PrimePower fits into the Synopsys complete low power solution.

Figure 1-1 PrimePower and Synopsys low-power solution



PrimePower supports multiple modes of operation. Based on the user requirements and available inputs, PrimePower can provide the following types of power analyses:

- Statistical activity based power analysis

For purely average power analysis, PrimePower supports propagation of switching activity based on defaults, user-defined switching or switching derived from an HDL simulation (either RTL or gate-level).

- Event-based power analysis

For extremely accurate analysis of power with respect to time, PrimePower supports analysis based on the gate-level simulation activity over time.

PrimePower builds a detailed power profile of the design based on the circuit connectivity, the switching activity, the net capacitance and the cell-level power behavior data in the Synopsys .db library. It then calculates the power behavior for a circuit at the cell level and reports the power consumption at the chip, block, and cell levels.

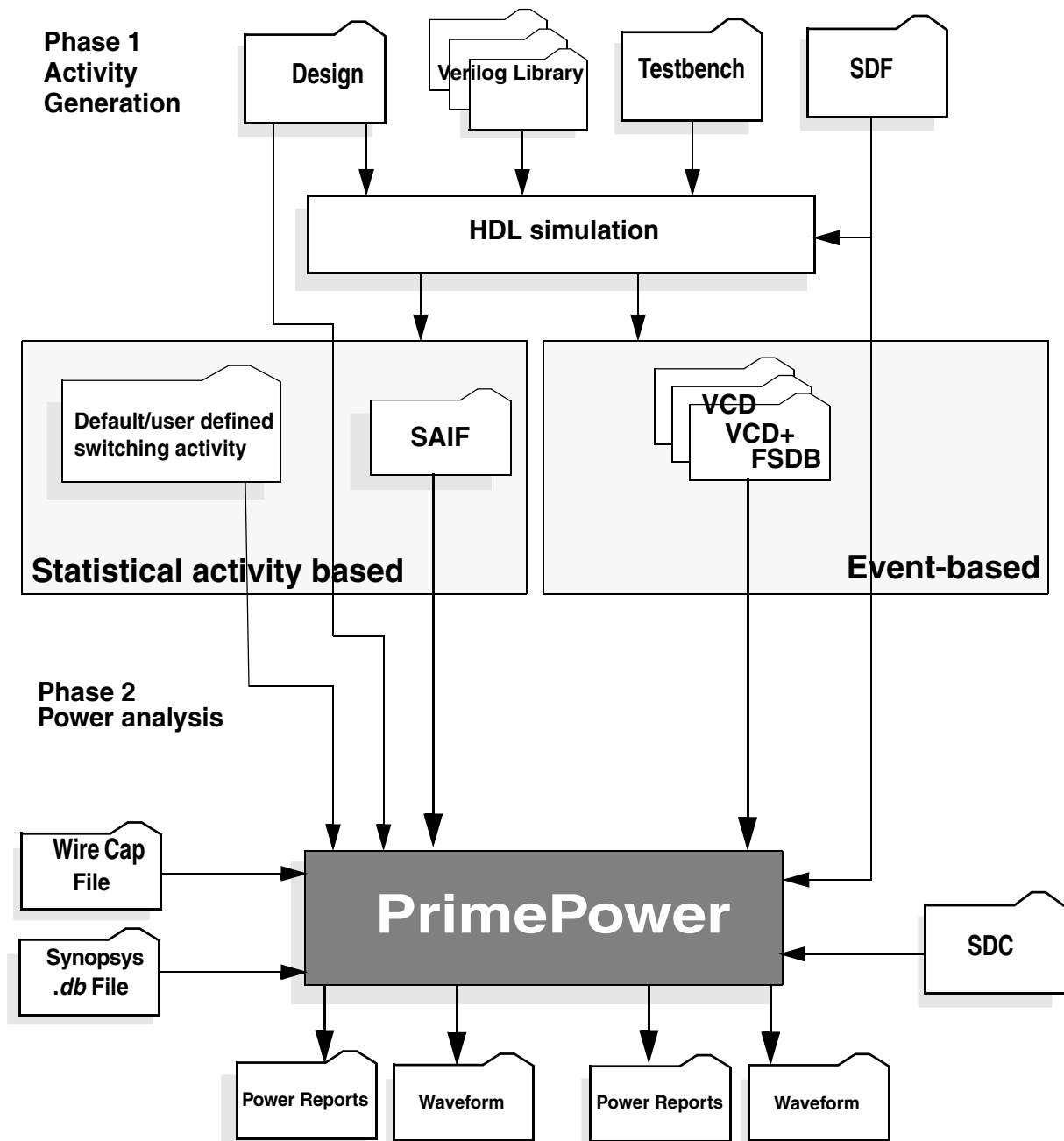
This chapter includes the following sections:

- [PrimePower Simulation Flow](#)
- [PrimeTime Compatibility](#)
- [Power Analysis Overview](#)
- [Power Types](#)
- [Calculating Power](#)
- [Calculating Dynamic Power](#)
- [Statistical Activity Based Power Analysis](#)
- [Modeling Special Conditions](#)
- [Operating Condition Effects](#)
- [Dual VT Libraries](#)

PrimePower Simulation Flow

PrimePower's simulation flow is comprised of two distinct phases. Generation of switching activity and power profile generation. These phases are shown in [Figure 1-2](#) and described in more detail in the sections that follow.

Figure 1-2 PrimePower power simulation flow using VCD files



Switching Activity Generation

For average power analysis, the user has many options when it comes to switching activity generation.

1. No switching activity

If no toggle information is provided, PrimePower assigns default switching activity to the primary input ports and uses the propagation engine to determine the switching of the internal nodes. Users do not have to simulate in order to do a gate-level average power analysis within PrimePower.

You can also use the `set_case_analysis` command to calculate average power for different modes.

2. User-defined switching activity generation

It is recommended that users provide at least the toggle rate of the clock and other critical nets for more accurate propagation and analysis.

3. RTL SAIF

Users can run an RTL level simulation and input the RTL toggle information into PrimePower. The propagation engine will use the provided switching activity, and propagate the required information to unannotated nets.

4. RTL VCD (beta feature in X-2005.06)

Users can run an RTL-level simulation and dump out the time-based switching activity in VCD, VCD+, or FSDB. Then input the switching activity to PrimePower with a gate-level netlist.

PrimePower will propagate the switching activity to unannotated nets and generate time-based power waveform and peak power report.

5. Gate-level SAIF

Even more accurate, is the use of the gate-level SAIF file derived from gate-level simulation.

6. Gate-level VCD

For accurate event-based analysis, users must provide the gate-level time-based switching activity in one of the supported formats: VCD/VCD+/FSDB. These files are generated via gate-level simulation.

Power Analysis

PrimePower performs the following steps to accurately analyze the power of the design:

1. Based on the design connectivity, design constraints, and wire-capacitance, PrimePower determines the transition times for all the pins within the design.
2. For average power analysis, PrimePower determines the state- and path-dependent switching for all the nodes within the design. If nets are not annotated, the propagation engine is used to determine the switching of unannotated nets.
3. PrimePower accesses the Synopsys library power tables and determines the power consumption for leaf-level cells which are summed to obtain the total design power. When performing event-based analysis, PrimePower processes every activity in the activity file to build an accurate power profile over time and to determine the peak power consumption.

Command Interfaces

You can run a power analysis in PrimePower using its graphical user interface (GUI) or `pp_shell`:

- The GUI can be started and stopped at any time from the `pp_shell`. It provides a suite of analysis tools to view the design data and results graphically.
- The `pp_shell` interface is a command-line and script execution environment based on the Tcl programming language.
- Interactive mode produces a set of text-formatted power reports based on data you enter interactively at a command-line prompt.

PrimeTime Compatibility

The PrimePower analysis tool is based on PrimeTime. PrimePower and PrimeTime are compatible in the following ways:

- They use the same Synopsys enhanced Tcl interface (CCI—Command Command Interfaces).
- They use the same technology libraries and read the same design data files in `.db` format.
- They use the same netlist parsers, parasitics, and library readers.
- They share the same delay calculation algorithms.
- They share many of the same commands, such as `create_clock`, `set_input_transition`, and `report_design`. Shared commands are identical or very similar in operation.

Power Analysis Overview

Power analysis is performed to determine the power consumption of the chip based on the activity.

PrimePower has two methods for power analyses:

- Event-based power analysis: PrimePower uses an event driven simulation algorithm to calculate the power consumption for each events. It provides an accurate and detailed time-based power information. Event-based analysis is vector dependent and can produce an average and peak power report.
- Statistical activity based power analysis: PrimePower uses a statistic and probability estimation algorithm to calculate the average power and construct the average cycle power waveform. It is vector free (no vector input) or vector weak dependent (SAIF input or user-defined switching activity input).

During power analysis, Synopsys power tools report the following types of power:

- Switching power
- Internal power
- Leakage power

These types of power belong to the broader categories of static and dynamic analyses.

Comprehending these power terms and understanding tool methodology and data flow are important for understanding power analysis reports.

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.

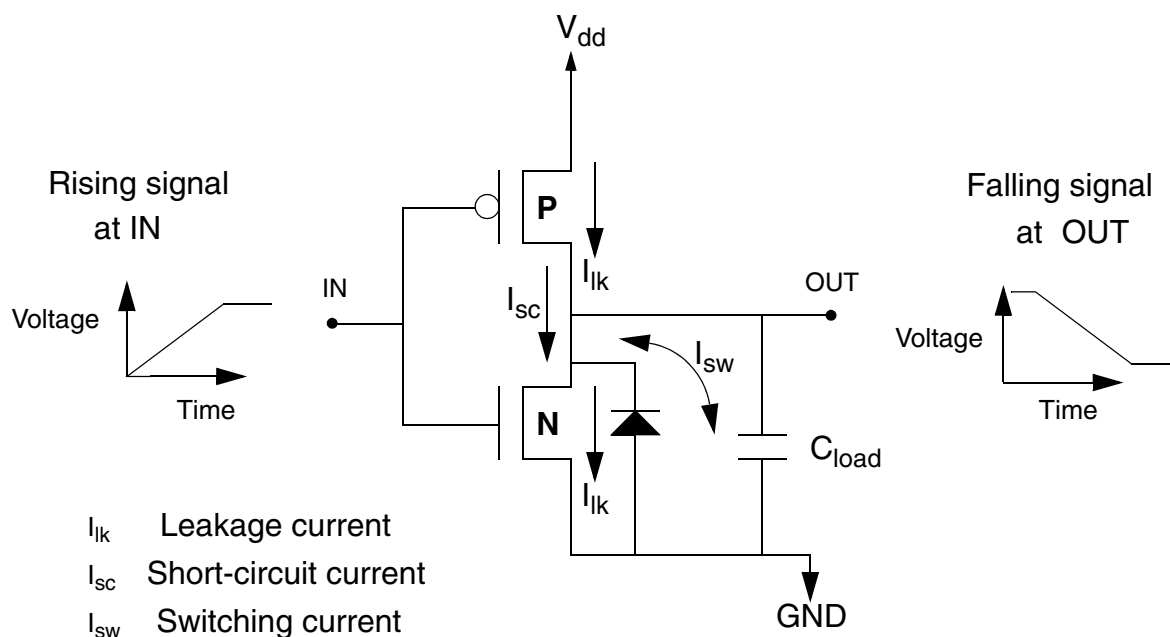
Because such charging and discharging are the result of the logic transitions at the output of the cell, switching power increases as logic transitions increase. Therefore, the switching power of a cell is a function of both the total load capacitance at the cell output and the rate of logic transitions.

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 1-3](#). 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}).

Figure 1-3 Components of Power Dissipation



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 > 50% 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 may be the dominant source of internal power.

Library developers can model internal power by using the internal power library group. For more information on modeling internal power, see the *Library Compiler Reference Manuals, Volumes 1, 2, and 3*.

Calculating Power

PrimePower calculates static power and dynamic power based on the library information, netlist, design constraints, and activity. The following sections describe how leakage, internal and switching power is calculated for static and event-based power analysis modes.

Calculating Leakage Power

Leakage powers refers to the power consumption when the gates are inactive.

Whether the user has selected statistical activity based power analysis or event-based power analysis, calculation of leakage power is the same because it is dependent on the state of the design, not switching.

PrimePower analysis 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:

$$P_{\text{LeakageTotal}} = \sum_{\forall \text{cells } (i)} P_{\text{CellLeakage}_i}$$

$P_{\text{LeakageTotal}}$ = 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 the approximate total leakage power dissipated by each library cell.

There are number of options for modeling the leakage power, based on the technology and the application. It can be defined at library level or at cell level. It can be state dependent or non-state dependent. It can be a single value, or a polynomial equation which modeled in SPPM (Scalable Polynomial Power Model) format. The SPPM leakage power table can take consideration of the effects of supply voltage and temperature.

When leakage power was insignificant compared with event-based power, this was not critical. Today however, as transistors shrink in size, the level of doping is increased, causing leakage currents to become relatively larger. The end result, is that even if a large portion of the device is totally inactive, it may still be consuming a significant amount of power in the form of leakage.

Library developers can provide a single leakage power for all cells in the library with the `default_cell_leakage_power` : `<value>; attribute`, or per cell with the `cell_leakage_power` : `<value>; attribute`. To model state-dependent leakage, library developers can additionally use Boolean expressions to define the conditions for different cell leakage power values. The `leakage_power attribute() {when: ...value:..}` enables state-dependent leakage modeling.

To calculate the cell-leakage, PrimePower determines the units based on the library attribute: `leakage_power_unit: <value>;`.

PrimePower uses the cell leakage power table in the following priority order: state-dependent leakage power table, cell-level default leakage power table and library-level default leakage power table. The state-dependent leakage power table is represented by `leakage_power` group with `when` statement. The cell-level default leakage power is represented either by `leakage_power` group without `when` statement, or by `cell_leakage_power` attribute.

The library-level default leakage power is represented by `default_cell_leakage_power` attribute. If none of the leakage power table is defined in the library, PrimePower assumes the leakage power to be 0.

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.

Shown below is an example of how PrimePower calculates leakage power for a non-SPPM model with state dependencies for a simple NAND gate.

Figure 1-4 Leakage power calculation for an NAND Gate with state-dependent values



```
library ....
leakage_power_unit : 1nW ;
cell (NAND) ...
cell_leakage_power : .5 ;
leakage_power ( ) {
    when : "A&B" ;
    value : .2
```

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

Hence, the total cell leakage value is:

$$(.33 * .2nW) + (.67*.5nW)=.4nW$$

In the above example, PrimePower examines the VCD (for time based) or SAIF (for statistical activity based) file to determine the percentage of time the gate was at each library-defined state.

Calculating Dynamic Power

Switching power and internal power provide the event-based power of the design:

Dynamic power = Switching power + Internal power

Based on the analysis mode selected, PrimePower calculates the dynamic power differently.

For event-based analysis, PrimePower calculates the switching energy for every event.

For statistical activity based analysis, PrimePower calculates the dynamic power based on the net toggles.

The mechanisms for calculating dynamic power based on the analysis type are described in the sections below.

Event-based Analysis

PrimePower performs the following steps during event-based analysis:

1. Captures events from the simulation as VCD data.
2. Sets up the ramp times of the input ports.
3. Annotates the wire capacitance.
4. Propagates the ramp times.
5. Matches the internal power table with the simulation events.
6. Calculates the switching power.

7. Models the X-state, Z-state, and glitch power.

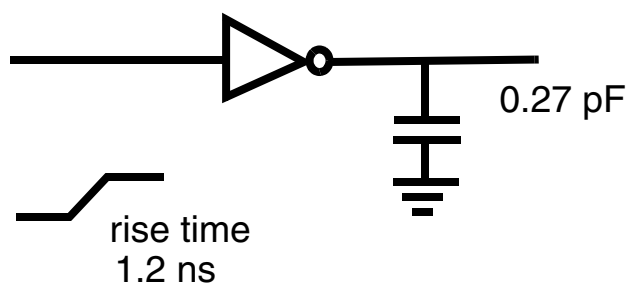
Calculating Internal Power

When computing internal power, PrimePower uses information characterized in the technology library. The `internal_power` library group and its associated attributes and groups define scaling factors and a default value for internal power.

Library developers can model internal power using tables (NLPM format) or scalable polynomials (SPPM format). The NLPM format allows library developers to store the energy consumption based on input transition time, output load, state dependencies and path dependencies. The SPPM format additionally, takes the operating conditions into account. In both cases, PrimePower sums the energy for each transition on every cell to determine the total power consumption.

Simple cells such as inverters define the tables/equations based on the rising or falling transition of the output pins. Shown below is an example of how PrimePower calculates power for a simple inverter modeled with a power table. See [Figure 1-5](#) below.

Figure 1-5 Calculating power



calculating power = rise energy + fall energy

```
power_lut_template(pwarc_rl_1) {
    variable_1 : input_transition_time ;
    variable_2 : total_output_net_capacitance ;
    index_1 ("0.400000, 1.200000, 2.800000, 4.200000, 6.000000") ;
    index_2 ("0.040000, 0.130000, 0.270000, 0.410000, 0.740000") ;
}

internal_power() {
    rise_power(pwarc_rl_1) {
        values("0.153765, 0.134493, 0.125847, 0.122712, 0.118753", \
              "0.330695, 0.260982, 0.214947, 0.192012, 0.165283", \
              "0.748326, 0.608307, 0.498417, 0.431097, 0.342493", \
              "1.128931, 0.953322, 0.791952, 0.696087, 0.552538", \
              "1.624707, 1.419942, 1.210227, 1.070307, 0.869998");
    }

    fall_power(pwarc_rl_1) {
        values("0.040883, 0.022974, 0.014491, 0.012210, 0.008594", \
              "0.208048, 0.137357, 0.094129, 0.072895, 0.048874", \
              "0.622676, 0.469622, 0.355640, 0.290894, 0.207652", \
              "1.000246, 0.815231, 0.638252, 0.537338, 0.399563", \
              "1.492820, 1.275449, 1.055851, 0.898160, 0.692917");
    }

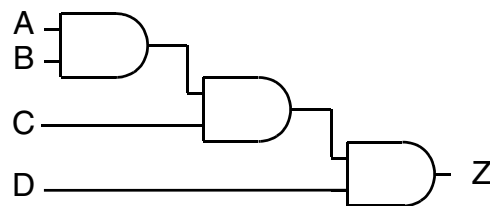
    Internal energy = rise_energy + fall_energy
    related = 0.21497 + 0.094129 = 0.309076pJ
}

for a 100ns period pulse,
```

Based on the functionality of the cell, the power tables can define state and path dependencies to accurately model the transistor behavior.

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 lookup tables/polynomials for internal power, depending on the state, or mode, of the cell.

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/polynomials for outputs, depending on the input transitions.

PrimePower chooses the appropriate path-dependent internal power table/polynomials for an output by checking the `related_pin` attribute in the library.

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

For cells in which output pins have equal or opposite logic values, PrimePower can use a three-dimensional lookup table in the NLPM format, or the condition can be included in the SPPM equation. Flip-flops are an example of cells with equal or opposite logic values.

See the *Library Compiler Reference Manual: Technology and Symbol Libraries*, *Library Compiler Reference Manual: VHDL Libraries*, and *Library Compiler Reference Manual: Physical Libraries* for more information on modeling internal power and for details on library modeling syntax and methodology.

Calculating Switching Power

PrimePower calculates the switching power (P_{sw}) by calculating the switching energy for every rising transition and dividing it by total simulation time. PrimePower assumes that switching power is generated only when the capacitance is being charged.

$$P_{sw} = \sum E_{sw_i} / T$$

$$E_{sw_i} = C_{load_i} \times V_{dd}^2 = \text{switching energy}$$

$$C_{load_i} = \text{capacitance of net}_i$$

$$V_{dd} = \text{supply voltage}$$

$$T = \text{Total simulation time}$$

Switching power is calculated on a per instance basis and the power consumption due to the switching of a net and is associated with the driver. The capacitive load of the net includes the parasitic capacitance and the gate and drain capacitance of all the pins connected to that net.

The parasitic capacitance can be obtained from the specified wire-load model or from back-annotation files. The pin capacitance values are obtained from the Synopsys library.

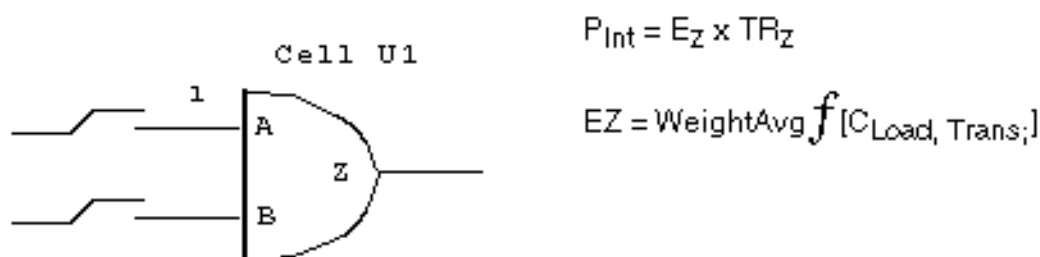
For the inverter example, [Figure 1-5](#), the switching power on the output net; assuming a supply voltage of 3.3V is $(0.27\text{pF} \times 3.3\text{V}/100\text{ns})$.

Statistical Activity Based Power Analysis

When computing internal power for statistical activity based analysis, PrimePower also uses information from the `internal_power` library group and its associated attributes.

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 technology library. [Figure 1-6](#) shows how Synopsys power tools calculate the internal power for a simple combinational cell, U1.

Figure 1-6 Internal Power Model (Combinational)



P_{Int}	Total internal power of the cell
E_Z	Internal energy for output Z as a function of input transitions, output load, and voltage if SPPM libraries are used
TR_Z	Toggle rate of output pin Z
TR_i	Toggle rate of input pin i, transitions per second
$Trans_i$	Transition time of input i

Based on input toggle rates and input transition times, PrimePower derives a weighted average of the power table values based on the load and transition time indexes. PrimePower supports both NLDM (table based) models and SPPM (scalable polynomial based) models for statistical activity based calculation.

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

Switching Power Calculation

PrimePower analysis calculates switching power (P_c) in the following way:

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

P_c	Switching power of the design
C_{Load_i}	Capacitive load of net i
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 net i.

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

Modeling Special Conditions

Power consumption for the following conditions

- Glitches
- Z states
- X states

is considered as part of the dynamic switching power. It affects both the internal and switching power.

Glitch Power

In order to calculate glitch power, PrimePower determines whether the pulse is small enough to be considered a glitch and then calculates the power based on the following formula:

Glitch power = Ratio * (Dynamic_power when there is no glitch)

where

Ratio = $[\text{pulse\_width} * 2 / (\text{rise\_ramp} + \text{fall\_ramp})]^2$

Note:

The above formula is only applied when the glitch width is less than the rise and fall ramp. If it is greater than that, PrimePower assumes that a full transition has occurred. Both internal and switching power are considered in the glitch power calculation.

The pulse width can be obtained from the simulation, and the rise/fall transition times can be obtained by executing the `report_power_calculation` command on the desired instance.

Z States

Transitions through the Z state are assumed to consume no power. PrimePower tracks it and if the transition after the Z state differs from the original transition, it is considered to be one full transition.

For example: 0 -> Z -> 1 or 1 -> Z -> 0 are considered as 1 transition.

However; 1 -> Z -> 1 or 0 -> 0 is not a transition and consumes no power.

X States

X Power is consumed during transitions to and from the X state. By default, every time a net transitions to or from the X state; it is considered as 1/2 of a power transition.

For example: 0 -> X = 1/2 transition

 0 -> X -> 0 or 0 -> X -> 1 = 1 transition

To change the scale factor of 1/2; users can modify the value of the controlling variable:

```
set pwr_xpower_scale_factor 0.2
```

Operating Condition Effects

Operating conditions are treated differently when calculating internal power with NLPM and SPPM models.

- *NLPM*

For various operating conditions, the table model supports scaling factors for the internal power calculation. These are listed below:

- k_process_internal_power
- k_temp_internal_power
- k_volt_internal_power

These factors however do not accurately model the non-linear effects of the operating conditions, so most vendors generate separate table-based libraries for different operating conditions.

- *SPPM*

To compute the internal power consumption of SPPM models, the input transition time is one of the variables in the polynomial equations.

The model may include up to seven variables:

- temperature
- voltage
- voltage1
- input_net_transition
- ouptut_net_transition
- total_net_capacitance
- equal_or_opposite_output_net_transition
- voltage2

Additionally for large data sets with abrupt changes, the models may include domains which allow for the definition of different polynomials for different ranges of the input variables.

Since the voltage and the temperature are variables in the equation, the same model can be used to calculate the power for different operating conditions.

You can apply the `create_operating_conditions` command to define the conditions under investigation and then apply them directly to the SPPM analysis.

Based on the equation and variable values, PrimePower solves the equation to determine the internal power of the cell. Refer to the *Library Compiler Reference Manuals* to obtain more information on the SPPM model syntax.

Dual VT Libraries

Static power dissipation has an exponential dependence on the switching threshold of the transistors V_{th} . In order to address low-power designs IC foundries offer technologies that enable multiple V_{th} libraries.

Each type of logic gate is available in two or more forms:

- with low-threshold transistors that switch quickly but have higher leakage and consume more power
- with high-threshold transistors that have lower leakage and consume less power but switch more slowly.

Since HighVT and Low VT cells are named differently, PrimePower locates the appropriate model based on the instantiation in the netlist and calculate the appropriate leakage power based. No special handling is required, as the leakage values and equations differ based on whether the cell is HighVT or Low VT.

2

Starting and Using PrimePower

PrimePower is a command-line-driven tool that runs under the UNIX or Linux operating system. You enter commands at the pp_shell prompt and PrimePower responds with text messages. PrimePower also offers a graphical user interface (GUI) for displaying design data and analysis results in graphical form. This chapter concisely explains how to start and use PrimePower for power analysis.

Overview

This chapter contains the following sections:

- [Starting PrimePower](#)
- [Using pp_shell](#)
- [PrimePower Documentation and Online Help](#)

- [Analysis Flow in PrimePower](#)

Starting PrimePower

Before you can use PrimePower, the software must be installed and licensed at your site. For information on installation and licensing, see the documentation that comes with the software release.

Starting a PrimePower Session

To start `pp_shell` (the PrimePower command-line interface), enter the following command at the operating system prompt:

```
% pp_shell
```

PrimePower automatically checks out a PrimePower license and displays the initial message and the PrimePower prompt.

```
PrimePower (R)
Version 2005.06 -- June 1, 2005
Copyright (c) 1988-2005 by Synopsys, Inc.
ALL RIGHTS RESERVED
```

```
This program is proprietary and confidential information of
Synopsys, Inc. and may be used and disclosed only as
authorized
in a license agreement controlling such use and disclosure.
```

```
pp_shell>
```

You enter commands at the `pp_shell` prompt (`pp_shell>`). PrimePower responds with text messages in the terminal window below your command entry line.

To start the PrimePower graphical user interface (GUI) rather than `pp_shell`, enter:

```
% primepower
```

This opens the PrimePower GUI window. You can then enter `pp_shell` commands in the command entry line at the bottom of the console. PrimePower displays system messages in the rectangular area above the command entry line.

Ending a PrimePower Session

Before ending a PrimePower session, the user may want to save the information set for the current design. To do so, the user can use the `save_session` command. For details, refer to the man page.

To end a PrimePower session, enter the command `quit` or `exit` at the `pp_shell` prompt:

```
pp_shell> exit
Maximum memory usage for this session: 0.72 MB

CPU usage for this session: 0 seconds

Thank you for using pp_shell!

%
```

Licenses

You need a PrimePower license in order to start `pp_shell` or the PrimePower GUI console window. PrimePower automatically checks out a license when you start PrimePower. When you exit from PrimePower, the license is automatically checked in, allowing others at your site to use the license.

The following additional licenses are optional:

- HDL-Compiler — To read Verilog with the `read_verilog` command and the `-hdl_compiler` option
- VHDL-Compiler — To read VHDL with the `read_vhdl` command and the `-vhdl_compiler` option
- `primepower_beta` — To invoke the `-rtl_direct` option to the `read_vcd` command that enables event-based analysis using an RTL VCD file.

PrimePower automatically checks out licenses as needed and checks in licenses when you exit from PrimePower. To explicitly check out or check in a license, use the `get_license` or `remove_license` command.

Startup Files

Each time you start a PrimePower session, PrimePower executes the commands contained in a set of PrimePower startup files. You can put commands into a startup file to set variables, to specify the design environment, and to select your preferred working options.

The name of each startup file is `.synopsys_pp.setup`. PrimePower checks for the presence of the file in the following directories:

1. The Synopsys installation setup directory at `admin/setup`. For example, if the installation directory is `/usr/synopsys`, the startup file name is `/usr/synopsys/admin/setup/.synopsys_pp.setup`.
2. Your home directory.
3. The current working directory from which you started PrimePower.

If more than one of these directories contains a `.synopsys_pp.setup` file, the files are executed in the order shown above: first in the Synopsys setup directory, then in your home directory, then in the current working directory. Typically, the file in the Synopsys setup directory contains setup information for all users at your site; the one in your home directory sets your personal preferred working configuration; and the one in your current working directory sets the environment for the current project.

To suppress execution of any `.synopsys_pp.setup` files, use the `-no_init` option when you start PrimePower.

Command Log File

When you end a PrimePower session, it saves the session history into a file called the command log file. This file contains all of the commands executed during the PrimePower session and serves as a record of your work. You can later repeat the whole session by running the file as a script (using the `source` command).

PrimePower creates the log file in the current working directory and names it `pp_shell_command.log`. Any existing log file with the same name is overwritten. Before you start a new PrimePower session, be sure to rename any log file that you want to keep.

You can specify a different name to use for the command log file by setting the `sh_command_log_file` variable in your startup file. (You cannot change this variable during a working session.)

Using pp_shell

The PrimePower shell, `pp_shell`, is a command-driven, text-only user interface based on the Tcl scripting language. You enter commands at the `pp_shell` prompt. PrimePower carries out the action for each command and reports the result in text format.

The PrimePower command syntax is case-sensitive. Commands, command options, arguments, and variables generally consist of lowercase characters and must be entered as such at the `pp_shell` prompt.

Object names in the design are also case-sensitive. For example, the following two commands are not the same because they refer to two different clocks, named `clk` and `CLK`:

```
pp_shell> report_wire clk
```

```
pp_shell> report_wire CLK
```

Entering Commands Interactively

When you are working interactively at the `pp_shell` prompt, you can abbreviate PrimePower command names and options to the shortest unambiguous string. For example, you can abbreviate the `get_attribute` command to `get_attr`.

Using command abbreviations is convenient for interactive `pp_shell` sessions. However, you should avoid using abbreviations in script files because they are susceptible to command changes in later versions of PrimePower. Such changes could make the abbreviations ambiguous.

When you enter a long command, you can split it across multiple lines using the back slash (\) continuation character. During entry of a long command using back slashes (or in other incomplete input situations), PrimePower displays a secondary prompt, a question mark, for each additional line of the command. For example:

```
pp_shell> alias my_rep_pwr { report_power \
? -sortby name}
```

In this manual, a command that cannot fit on one line is shown on multiple lines using the continuation character. However, for easier reading, the secondary prompt is usually omitted from the examples shown in the manual.

Using Command-Line Editing

To enable command-line editing, add the following line to your `.synopsys_pp.setup` file (the default value of the `sh_enable_line_editing` variable is false):

```
set sh_enable_line_editing true
```

Note:

This variable must be set in your `.synopsys_pp.setup` file. It has no effect from the command line.

Choosing an Editor Mode

To set the command line editor mode to either `vi` or `emacs` you can add the following variable to your setup file or enter it directly in your shell (valid values are `emacs` or `vi`; the default is `emacs`):

```
set sh_line_editing_mode editor
```

Using Command-Line Editing

Once you have enabled command-line editing, you can use the following types of keystrokes:

- Up and down arrows for history recall
- Left and right arrows to move forward or backward one character
- Home and End keys to move to the beginning or end of the current line
- Emacs line editing keystrokes
- Vi editing keystrokes
- Tab for auto-completion of command, variable, and file names

Note:

The auto-completion feature does not work with aliased commands. For instance, if you use `pr` for the `printvar` command, using the Tab key will not complete any variable names.

To list the editing keys that are active in either emacs or vi mode, use the following Tcl command:

```
list_key_bindings
```

Note:

PrimePower does not save and restore the settings of the `sh_enable_line_editing` and `sh_line_editing_mode` variables. If you restore a previously saved session, these variables will retain the settings they had before the restore (for instance, if you had not enabled command-line editing in the

saved session, but did have it enabled at the time you restored that session, command-line editing would still be enabled after the restore).

Command Scripts

A command script is a text file containing a sequence of `pp_shell` commands. The setup file `.synopsys_pp.setup` is an example of a script. Also, the log file generated at the end of a PrimePower session can be used as a script. You can create your own scripts, or use scripts created by others, to carry out complex or repetitive tasks.

PrimePower can recognize script files in plain ASCII format, ASCII compressed in gzip format, and ASCII encoded into bytecode format by the TclProCompiler (3rd party tool). To execute a script in any of these forms, use the `source` command:

```
pp_shell> source file_name
```

To execute a specified command script upon startup, use the `-f` option when you start PrimePower:

```
% pp_shell -f file_name
```

By using the features of the Tcl command language, you can create scripts that use variables, loops, and conditional execution. The flow control commands `if`, `while`, `for`, `foreach`, `break`, `continue`, and `switch` determine the execution order of other commands.

Any line of text in a script file that begins with the pound sign (`#`) is a comment, which PrimePower ignores. Any text from a semicolon and pound sign (`; #`) to the end of a line is also considered comment text.

Running Scripts in Batch Mode

To run PrimePower in batch mode, use the `-f` option when you invoke PrimePower to source the script, and redirect the output to a file. For example:

```
% pp_shell -f big_analysis.tcl > big_analysis.out &
```

PrimePower redirects output and error messages to the specified file (in this example, `big_analysis.out`), which you can read at any time.

If your script contains a syntax error, PrimePower stops and waits for input unless the variable `sh_continue_on_error` is set to true.

Make sure you end the script with the `quit` or `exit` command. Otherwise, the `pp_shell` prompt does not appear and you will not know when the script has finished executing. If your script does not end with the `quit` command, PrimePower waits for input. In that case, bring the process to the foreground, then type `quit` or `exit` to end the session.

Command Results

Every PrimePower command has a result. Many commands result in “1” to indicate success or “0” to indicate failure. For example:

```
pp_shell> link  
1
```

For many other commands, the result is a collection. Collections are covered in more detail in the "[Command Interface and Tcl](#)" chapter. For example, the result of the `get_ports` command is a collection of ports:

```
pp_shell> get_ports IN*  
{"IN1", "IN2", "IN3", "IN4"}
```

The command `get_ports IN*` creates a collection of all ports in the design beginning with the letters `IN`.

In PrimePower, commands are often nested, one command within another, so that the result of one command is used as an argument for another. Each nested command is enclosed in square brackets (`[]`). This command structure is common for performing operations on design elements.

For example, the `set_input_transition` command sets a transition time on one or more specified input ports. You can gather a collection of input ports with the `get_ports` command and then pass the result to the `set_input_transition` command:

```
pp_shell> set_input_transition 2.3 [get_ports IN*]
```

The command `get_ports IN*` creates a collection, which is then used as the second argument of the `set_input_transition` command. The effect of the entire command is to set the input transition time to a specific value for all ports beginning with the letters `IN`.

The output of some commands is a report, such as the `man` or `report_power` command. By default, PrimePower pauses between each screenful of text (like using the `more` command under UNIX). To cause it to issue a long report all at once, set the page mode variable to false:

```
pp_shell> set sh_enable_page_mode false
```

Variables

PrimePower offers many options that you control by setting variables. To set a variable, you use the `set` command:

```
pp_shell> set variable_name variable_setting
```

When you set a variable, the displayed result is simply the new setting for the variable. For example:

```
pp_shell> set sh_enable_page_mode false
false
```

If you attempt to set a variable to an invalid value for that variable, PrimePower responds with an error message. For example:

```
pp_shell> set sh_enable_page_mode maybe
Error: can't set "sh_enable_page_mode": invalid value:
      use true or false
Use error_info for more info. (CMD-013)
```

If you mistype a system variable name, PrimePower interprets it as a request to define a new, user-defined variable.

By default, PrimePower does not issue any other message. You can set the variable: `set sh_new_variable_message true` to issue a message when variables are created.


```
pp_shell> set sh_enable_pag_mode true  
Information: Defining new variable 'sh_enable_pag_mode'.  
(CMD-  
041)  
true
```

To find out the current setting for a variable, use the `printvar` command. For example:

```
pp_shell> printvar sh_enable_page_mode  
sh_enable_page_mode = "true"
```

You can use the wildcard character (*) to view multiple related settings because related commands often begin with the same prefix. For example, to see a list of shell-related variables:

```
pp_shell> printvar sh_*  
sh_arch = "sparcOS5"  
sh_command_abbrev_mode = "Anywhere"  
sh_command_log_file = ""  
sh_continue_on_error = "false"  
sh_dev_null = "/dev/null"  
sh_enable_page_mode = "true"  
sh_new_variable_message = "true"  
...
```

You can use multiple wildcard characters. For example, to see a list of all power-related variables:

```
pp_shell> printvar *pwr*  
pwr_calc_use_slew="max"  
pwr_default_voltage="3.3"  
pwr_match_xstate="0"  
pwr_xpower_scale_factor="0.5"  
pwr_no_sppm_extrapolation="false"  
pwr_table_internal_energy_only="true"
```

The command `printvar` by itself, with no arguments, lists all the variables and their settings:

```
pp_shell> printvar
auto_link_disable      = "false"
auto_wire_load_selection = "true"
bus_extraction_style    = "%s[%d:%d]"
bus_inference_descending_sort = "true"
bus_inference_style     = ""
bus_naming_style        = "%s[%d]"
...
```

Tcl Interface

The `pp_shell` interface is based on the Tcl scripting language, like the Tcl shell of Design Compiler (`dc_shell-t`). This means that you can use features of the Tcl language such as user-defined variables, procedures, conditional execution, lists, and expressions.

The general features of the Tcl language are beyond the scope of the PrimePower documentation. For this type of information, see any reference book on Tcl. For information on how the features of Tcl are used in `pp_shell`, see the "[Command Interface and Tcl](#)" chapter.

Using the PrimePower Command Template

To assist you in generating the PrimePower command files, templates are provided in the following directory:

```
<path to PrimePower installation>/doc/pp/templates
```

The `vcd.tcl` file contains the basic command set for the VCD flow.

PrimePower Documentation and Online Help

If you need help while using PrimePower, information is readily available from the following resources:

- The PrimePower `help` command
- The PrimePower `man` command
- PrimePower manuals
- SolvNet articles
- Synopsys Online Documentation (SOLD)

Using the help Command

The `help` command provides concise information on the PrimePower commands. You can get a partial or complete list of commands or view the syntax of a particular command.

The `help` command by itself shows a list of all PrimePower commands, organized by command group:

```
pp_shell> help
DC Emulation:
  cell_of, drive_of, filter, find, link, load_of, read_file,
  remove_wire_load_min_block_size

PT Backward Compatibility:
  add_to_selection, filter_selection, foreach_selection,
  remove_from_selection, select_cell, select_clock,
  ...
```

You can use wildcard characters to restrict the scope of the command list or to find the name of a command that you cannot remember exactly. For example, to find all commands containing the string “report”:

```
pp_shell> help *report*
report_annotated_parasitics # Report on annotated RC
parasitics
  report_bus # Report the bussed ports/net in the
current_instance/design
  report_cell # Report cell info
  report_clock # Report clock info
  report_design # Report design information
  report_hierarchy # Report the reference hierarchy of
current_instance/design
  report_lib # Generate a library report
  report_net # Report net info
  report_port # Report port info
  report_power # Report the result of power analysis
for
the design
  report_reference # Report the references in
current_instance/design
  report_vcd_hier # Report design hierarchy in the VCD
file
  report_wire # Display net information
  report_wire_load # Report wire load info
```

For a concise description of a particular command, use `help` with just the command name:

```
pp_shell> help set_input_transition
set_input_transition # Set input transition on ports
or pins
```

To see the full command syntax, including options and arguments, use the `-verbose` option:

```
pp_shell> help -verbose set_input_transition
set_input_transition # Set port transition time
    [-rise]                (Set rise transition only)
    [-fall]                (Set fall transition only)
    [-min]                 (Set minimum transition only)
    [-max]                 (Set maximum transition only)
    [-clock clock_name]    (Transition is relative to the clock)
    [-clock_fall]          (Transition is relative to falling edge
of the clock)
    transition              (Port transition:
                             Value >= 0)
    port_list              (List of input ports)
```

An alternative method to get the same help is to enter the command name directly and use the `-help` option:

```
pp_shell> set_input_transition -help
Usage:
    set_input_transition # Set port transition time
        [-rise]                (Set rise transition only)
        [-fall]                (Set fall transition only)
        [-min]                 (Set minimum transition only)
        [-max]                 (Set maximum transition only)
        [-clock clock_name]    (Transition is relative to the clock)
        [-clock_fall]          (Transition is relative to
falling edge
of the clock)
        transition              (Port transition:
                             Value >= 0)
        port_list              (List of input ports)
```

If need more detailed information on a command, use the `man` command.

Using the man Command

To get detailed information on any command, variable, or system message in PrimePower, use the `man` command. It works very much like the UNIX `man` command. At the `pp_shell` prompt, type the word `man` followed by the command, variable, or message code. PrimePower responds with a detailed description.

Man Pages for Commands

The man page for a command shows the command syntax, describes each option and argument, describes the how the command operates, provides some examples, and lists some related commands. For example:

```
pp_shell> man read_db
```

```
read_db          2.  Synopsys Commands  Command Reference
```

NAME

```
read_db          Reads in one or more design or library
                  files in Synopsys database (db) format.
```

SYNTAX

```
string read_db [-netlist_only] [-library] file_names
list file_names
```

ARGUMENTS

```
-netlist_only  Fordesignsonly; ignoredforlibraries.
```

```
Indicates that only the netlist is to be
read; attributes are not to be read.
This can greatly increase performance
while reading designs
```

```
...
```

DESCRIPTION

```
...
```

EXAMPLES

```
...
```

SEE ALSO

```
...
```

Man Pages for Variables

The man page for a variable shows the variable name, variable type (string, list, Boolean, integer, or floating-point number), the default setting, and a description of what the variable controls. For example:

```
pp_shell> man search_path

search_path 3.  Attributes and Variables  Command Reference

NAME
    search_path

TYPE
    list

DEFAULT
    " " (empty)

DESCRIPTION
    A list of directory names that specifies directories
    to be searched for design and library files that are
    specified without directory names ...

    ...

SEE ALSO
    link_design(2), printvar(2), read_db(2);
    sh_source_uses_search_path(3), source(2).
```

Man Pages for Messages

When PrimePower issues a warning or error message, it shows a letter-number code at the end of the message. For example:

```
pp_shell> calculate_power
Error: Current design is not defined. (DES-001)
```


The message code is DES-001. To get more information about the warning or error that occurred, use the `man` command with the message code. Be sure to type uppercase letters for the error code. For example:

```
pp_shell> man DES-001
```

messages	N. Messages	Command Reference
NAME	DES-001 (error) Current design is not defined.	
DESCRIPTION	The current design is not defined. Many commands require that the current design is set.	
WHAT NEXT	You must read a design database file and link a design.	

PrimePower Documentation

Information on PrimePower is available from the following manuals:

- *PrimePower Manual*
- *PrimePower Quick Reference Guide*

These manuals are available in electronic form on the Synopsys Online Documentation (SOLD) CD provided with the PrimePower software release. They are also available from the Synopsys SolvNet Web page on the Internet at <http://solvnet.synopsys.com>. In order to read these manuals in electronic form, you need the Adobe Acrobat Reader (available for free from www.adobe.com). Printed manuals are available for purchase from the Synopsys MediaDocs Shop; for ordering information, go to <http://mediadocs.synopsys.com>.

The *PrimePower Manual* (the manual you are now reading) introduces basic concepts of power analysis and using PrimePower.

The *PrimePower Quick Reference Guide* is a pocket-sized reference book that lists and briefly describes all the PrimePower commands and variables. It is a concise summary of the man page text.

SolvNet Articles

Users often contact Synopsys customer support with technical questions about Synopsys products, including PrimePower. The answers to the most frequently asked questions are put into a database that you can access through the Internet. This question-and-answer article database is part of a larger network of Synopsys online services called SolvNet.

To access the SolvNet database,

1. Go to the SolvNet Web page at <http://solvnet.synopsys.com>.
2. When prompted, enter your user name and password. If you do not already have a Synopsys user name and password, click on the link to the registration page.
3. In the SolvNet page, click Advanced Search. Enter your search criteria and click Search. SolvNet displays a “hit list” showing the titles of articles that match your search criteria.
4. Click the title of interest and read the article.

Synopsys Online Documentation (SOLD)

The PrimePower manuals are available on the Synopsys Online Documentation (SOLD) CD-ROM provided with the PrimePower software release.

In order to read the documentation, you need to have an Adobe product that includes PDF reader capabilities. If you do not already have this capability, you can use Adobe Acrobat Reader, which is available as a free download from Adobe. To allow full-text searches through the documentation database, be sure to install the optional search feature of Acrobat Reader.

You can view the SOLD documentation on any platform, not just the one on which you are using PrimePower. For example, you can use your Windows desktop computer to view the documentation while you work on your UNIX workstation, or you can read the documentation on your laptop computer while traveling. For even greater convenience, you can copy the SOLD documentation files to your hard disk and leave the SOLD CD on your office shelf.

For fast and easy access to the SOLD documentation,

1. If you do not already have the ability to view and search PDF files on your computer, download Adobe Acrobat Reader with Search from www.adobe.com and install the software.
2. Copy the SOLD CD files to a directory on your computer. To save disk space, copy only the directories containing information you need.
3. Start Adobe Acrobat Reader and open the file called TOP.pdf. From there, click the document title that you want to view, or click Using the Online Documentation for further instructions.

4. Each time you open a document for viewing, you automatically add the full-search index for that document's group to the database of searchable indexes. View some pages from each group for products that you use to make those databases searchable; or add the indexes (.pdx files) manually using *Tools > Search > Indexes*.
5. To search for occurrences of a text string throughout the documentation set, use *Tools > Search > Query*. Enter the string and click Search. Acrobat Reader responds with a "hit list" showing the titles of the documents containing the string, with documents having the most occurrences listed first. Click the desired title to view the document.

To restrict the scope of the search, in the Search dialog box, click Indexes. In the Index Selection dialog box, toggle on the databases you want to search and toggle off the others. Then click OK.

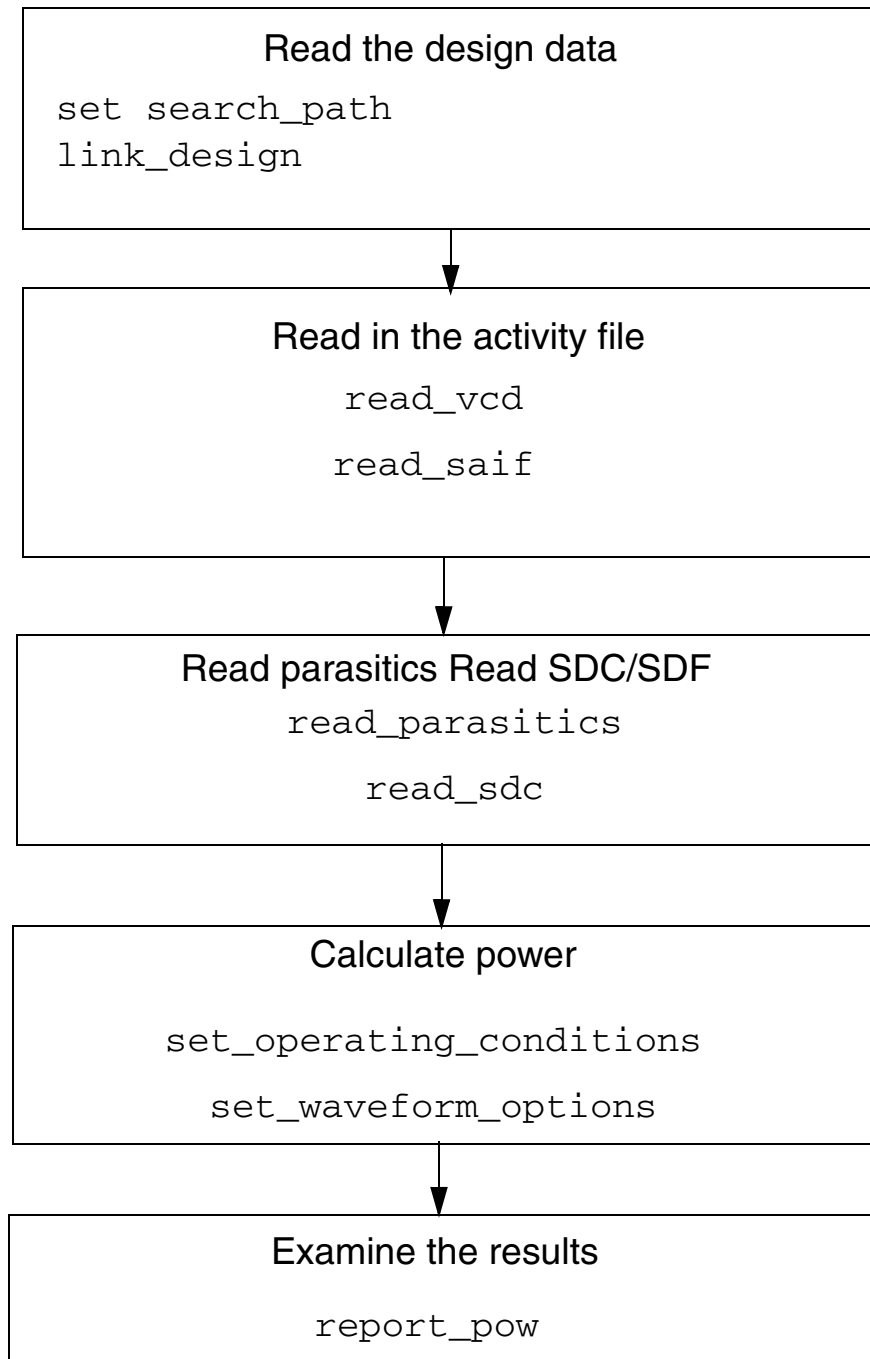
Analysis Flow in PrimePower

To successfully perform power analysis for a design, you need to do certain steps in the proper order. A typical analysis session consists of the following steps:

1. Read in the design data (a gate-level netlist and associated technology libraries).
2. Read in the activity file.
3. Specify the environment and analysis conditions such as operating conditions, `wire_load` models and calculate the power consumption.
4. Examine the power reports

Figure 2-1 shows the basic steps in the PrimePower analysis flow and some commonly used commands for each step.

Figure 2-1 PrimePower Analysis Flow



The following sections provide an overview of the basic analysis steps and commands used in PrimePower. Note that this overview only describes some of the more common work flows and commands. For more complete information, see the later chapters in this manual, the other manuals in the PrimePower documentation set, and the man pages for the PrimePower commands.

Reading the Design Data

The first step is to read in the gate-level design description and associated technology library information. PrimePower accepts design descriptions and library information in .db (Synopsys database) format and gate-level netlists in Verilog, and VHDL formats. The corresponding commands for reading design files are `read_db`, `read_ddc`, `read_verilog`, and `read_vhdl`.

After you read in a set of hierarchical design files, the `link_design` command resolves all references between different modules in the hierarchy and builds an internal representation of the design for power analysis.

You can set variables with the `set` command to control the behavior of PrimePower. The `search_path` variable specifies a list of directories from which to read the design and library files, so that you do not need to specify a full path each time you read in a file. The `link_path` variable specifies where and in what order PrimePower looks for design files and library files for linking the design. For example:

```
pp_shell> set search_path ". /u/proj/design /u/proj/lib"
pp_shell> set link_path "* STDLIB.db"
```

Checking the Design Setup

Before you begin a full analysis, it is a good idea to check the characteristics of the design such as the hierarchy, library elements, ports, nets, cells and the analysis setup parameters such as clocks and wire load models.

You can use a variety of “report” commands to get information on the characteristics of the design and the timing constraints that have been placed on the design.

These are some of the more common “report” commands:

- The `report_design` command lists the attributes of the design, including the chosen operating conditions, wire load information, and design rules.
- The `report_port` command lists the ports and shows port information such as the direction, pin capacitance, wire capacitance, input delay, output delay, related clock, design rules, and wire load information.
- The `report_net` command lists the nets and shows net information such as the fanin, fanout, capacitance, wire resistance, number of pins, net attributes, and connections.
- The `report_cell` command lists the cells used in the design and cell information such as the library name, input names, output names, net connections, cell area, and cell attributes.
- The `report_hierarchy` command generates a hierarchical list of submodules and leaf-level cells in the current design.

- The `report_reference` command generates a list of hierarchical references, showing for each submodule the reference name, unit area, number of occurrences, total area, and cell attributes.
- The `report_lib` command generates a report on a specified library showing the library's time units, capacitance units, wire load information, defined operating conditions, logic trip-point thresholds, and names of library cells.
- The `report_wire_load` command shows the wire load models set on the current design or on specified cells.

Providing Circuit Activity

The analysis options are based on the type of circuit activity which is provided. If no activity is provided, or if users provide SAIF files (either RTL or gate-level). PrimePower can calculate the average power based on toggle rates and optionally generate average cycle waveforms. If event-based activity data is provided, such as a VCD, VPD, or FSDB file, PrimePower can perform accurate analysis based on the power consumption for each event.

- Statistical activity power analysis
 - Vector-free

Users have the option to provide no activity in which case default toggle rates are applied and zero-delay simulation is used to propagate the activity throughout the circuit.

The following commands affect the default toggle rates applied:

```
create_clock  
set pwr_default_toggle_rate  
set pwr_default_probability  
set_switching_activity  
reset_switching_activity  
set_case_analysis
```

By default, the probability and toggle rates are set to 0.5. The period is set to the highest frequency clock defined.

- RTL-level SAIF

The following commands are used when providing an RTL-level SAIF file:

```
read_saif  
set_rtl_to_gate_name
```

The SAIF file should contain hierarchical ports and synthesis invariant nodes. If names have been altered during synthesis, the user may require the `set_rtl_to_gate_name` to map the RTL names to the gate-level names.

PrimePower performs zero-delay simulation and propagation to annotate the nodes for which no activity is provided.

- Gate-level SAIF

More accurate analysis is obtained with gate-level SAIF files. Specification of the gate-level SAIF is done also with the command:

```
read_saif
```

- Event-based power analysis

Event-based power analysis is performed when the user provides activity data for every node based on HDL simulation. PrimePower supports VCD and VCP files directly.

The following command is used to specify activity files which contain the activity over time:

```
read_vcd
```

Checking the Activity Files

Power analysis problems due to the annotation can be investigated with the following commands.

To verify the correct mapping of the nets in the activity file with those in the design and percentage annotation, use the following commands based on the activity file format.

- Toggle-rate Activity

To check the SAIF files, users can apply the following commands to view the annotation.

To view the annotation contained in the SAIF file, use the

command:

```
report_saif
```

- Event-based Activity

If mapping problems exist, users can apply the following command to compare the VCD and design hierarchies:

```
report_vcd_hier
```

This command can be used to display the VCD hierarchy prior to `calculate_power`.

Specifying the Environment and Analysis Conditions and Calculate Power

PrimePower allows you to specify the operating environment and the conditions for power analysis. For example, you can do the following:

- Specify the process, temperature, and voltage operating conditions, as characterized in the technology library
- Specify parasitics
- Specify analysis parameters

Operating Conditions

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. The ASIC vendor typically characterizes the device parameters in the laboratory under varying conditions, and then specifies the parameter values under different sets of conditions in the technology library.

In PrimePower, the `set_operating_conditions` command specifies the operating conditions for analysis, so that PrimePower can use the appropriate set of parameter values in the technology library.

Parasitics

- *Port/Net Loads*

You can use the `set_load` command to specify the amount of capacitance on a port or net, allowing PrimePower to more accurately calculate the effects of the load on power.

- *Wire Load Models*

In order to accurately calculate power, PrimePower needs information on the parasitic loads of the wire interconnections. Before placement and routing have been completed, PrimePower estimates these loads by using wire load models provided in the technology library. The `set_wire_load_model` command specifies which wire load model to use for the current analysis.

- *Back-Annotated Parasitics*

After placement and routing, you should back-annotate the design with detailed parasitic resistance and capacitance information using `read_parasitics`. This information overrides the wire load models, allowing PrimePower to perform a layout-accurate power analysis.

- *Synopsys Design Constraints (SDC)*

PrimePower can read the SDC information written out by Design Compiler or PrimeTime.

Analysis Parameters

Prior to issuing the command to calculate the power consumption; the following variables and commands can be applied which affect how the power is calculated and what data is stored.

- *Waveforms*

The user can choose to generate waveform data in order to view the power waveform and to report the peak power consumption. The `set_waveform_options` command allows users to apply parameters for waveform generation.

- *Power Calculation Variables*

The following PrimePower variables affect the power how power is calculated. These are listed below along with their default values:

```
pwr_default_voltage = "3.3"  
pwr_match_xstate    = "0"  
pwr_table_internal_energy_only = "true"  
pwr_xpower_scale_factor = "0.5"
```

Performing a Full Analysis

After you load and constrain the design and specify the conditions for analysis, you can perform a full analysis. The following command is often used to analyze the power:

```
calculate_power
```

Examining the Results

By default, the `report_power` command is used to generate a report which contains the power consumption for all instances within the design except for the leaf level cells.

As of the X-2005.06 release, PrimePower can also generate Power Compiler-like reports. Please refer to the section, [“New Power Report” in Chapter 10](#).

Shown below is the sample output of the `report_power` command:

```
PrimePower (TM)

Version 2004.12 for sparcOS5 -- Nov 1, 2004
Copyright (c) 2000-2004 by Synopsys, Inc.
ALL RIGHTS RESERVED

*****
Report:                average power
Design:                inverter
Library File(s):       /usr/lab1/src/libt/tech_pwarc.db
Simulation period[1]:  0 ns - 1050.11 ns
Sampling interval:     1 ns
Reporting threshold:   0 Watt
Date:                  Mon Nov  1 10:11:09 2004
*****
Definitions:
Total Power            = Dynamic + Leakage
Dynamic Power          = Switching + Internal
Switching Power        = load capacitance charge or discharge power
Internal Power          = power dissipated within a cell
X-tran Power           = component of dynamic power-dissipated into x-
transitions
Glitch Power           = component of dynamic power-dissipated into
detectable              glitches at the nets
Leakage Power          = reverse-biased junction leakage + subthreshold
leakage
```

```

-----
Level          Instance_Name (Cell_Name)    #_of_States
-----
Total          Dynamic    Leakage    Switching    Internal    X-tran    Glitch
Power          Power      Power      Power      Power      Power      Power
in Watt      in Watt    in Watt    in Watt    in Watt    in Watt    in Watt
              (% of Tot) (% of Tot) (% of Dyn) (% of Dyn) (% of
Dyn) (% of Dyn)
-----
              Peak          Peak
              Power      Time Interval
              in Watt    in ns
-----

----- Instances -----
*----- 0  pp_root (.) 43
3.673e-05 3.673e-05 4.853e-10 6.466e-06 3.026e-05 1.504e-07
0.000e+00
              (100.00%) ( 0.00%) ( 17.60%) ( 82.40%) ( 0.41%) ( 0.00%)
              1.900e-03    100-101

-*----- 1  inverter (inverter) 43
3.673e-05 3.673e-05 4.853e-10 6.466e-06 3.026e-05 1.504e-07
0.000e+00
              (100.00%) ( 0.00%) ( 17.60%) ( 82.40%) ( 0.41%) ( 0.00%)
              1.900e-03    100-101
----- End of Instances -----

----- End of report -----

```

Additional power reporting commands include:

- `report_clock_tree_power` which generates a list of the clock-tree elements and their power consumption
- `report_register_power` which lists all the registers in the design and associated power
- `report_scan_register_power` which lists all the scan registers and associated power

Users have the option to generate their own reports using TCL constructs and accessing the power attributes.

Saving and Restoring PrimePower Sessions

The commands `save_session` and `restore_session` allow you to save the current state of a PrimePower session and restore the same session later. When you need to examine the results of an earlier power analysis, using the save/restore feature avoids repeating the time-consuming `calculate_power` portion of the analysis. Using `restore_session` takes you to the same point in the analysis using only a small fraction of the original runtime.

A saved and restored PrimePower session retains the following types of information:

- linked design and loaded libraries
- clocks, switching activity, and other constraints
- operating conditions
- back-annotated parasitics
- variable settings
- analysis data

Here are some situations where the save/restore feature can be useful:

- You use a script to run a PrimePower session overnight. The script uses `save_session` command after the final `calculate_power` . At any time later on, you and other users can restore the session and examine the results using `report_power`, `gui_start`, and so on.

- You save the current state of the analysis as a checkpoint, which can be restored later in case of an error or unexpected change in the analysis environment. Users can interrupt, save the session, and then resume power analysis.
- You save the current session and then restore it multiple times to apply different chip operating modes or different vectors.

The platforms used for saving and restoring the PrimePower session can be different (Solaris, HP, Linux, 32-bit, 64-bit). However, the version of PrimePower must be exactly the same, including any service packs.

A restored session gives results that are functionally equivalent to the saved session. However, the ordering of items in reports can be different for commands such as `list_libraries`, `get_clocks`, and `get_pins`.

Note that the following types of information are not saved or restored:

- Collections of derived data types
- Tcl procedures and command history
- GUI state (timing path tables, schematics, histograms, and so on)

The save/restore feature is intended only as a PrimeTime session tool, not a database archiving tool or a method for storing test cases.

You must specify a session name when you save or restore a session. For example:

```
pp_shell> save_session session1A
...
pp_shell> restore_session session1A
```

The session is saved in a new subdirectory created in the current directory. If you want to overwrite an existing saved session, you must use the `-replace` option of the `save_session` command. For more information, see the man pages for the `save_session` and `restore_session` commands.

3

PrimePower Tutorial

The following tutorials will lead you through the four basic usage flows.

Test Case

The tutorial file is located at `<installation directory>/doc/pp/tutorial.tgz`. The design consists of a multiplier, an adder and some glue logic.

To use the tutorial, copy the tutorial file to your own directory and type the following command: `gunzip tutorial.tgz` and `tar xvf-`. Then follow the instructions below for the selected exercise.

Power Analysis With VCD

The VCD file is from gate-level simulation. It contains most details of the switching activity. With VCD, the power analysis is the most detailed and accurate. Power waveform over the whole simulation period can be generated from this analysis.

Related Files

In the tutorial directory, the files listed in Table 1 are related to the first tutorial.

Table 3-1 Files for the VCD tutorial

File Name	Description
<code>./pp_vcd/pp_vcd.tcl</code>	PrimePower script to run the analysis with VCD file
<code>./sim/vcd.dump</code>	VCD file
<code>./src/hdl/gate/mac.vg</code>	Gate- level netlist of the design
<code>./src/lib/snps/core_typ.db</code>	Technology library file
<code>./src/annotate/mac.spef.gz</code>	Parasitic file

Technology Library

This file contains library cells. Each cell has timing, power and. characterization information. Internal and leakage power are in the library.

Gate-level Netlist

PrimePower only supports gate-level netlist. It contains leaf-level cells that are the instantiation of the library cells. The netlist can be in formats of Verilog, VHDL, and .db. Verilog is used for this tutorial. It can be either flatten or hierarchical.

VCD File

The VCD file contains switching activity of the design. It has headers, nodes, and value change sessions.

Parasitic File

The parasitic file contains capacitance of the nets. Capacitance is one of the factors determine the dynamic power. You can unzip and view the file.

PrimePower Script

In this file you will find the basic commands used for power analysis.

The commands are grouped into a few sessions, which represent the basic steps of power analysis. They are common for all kinds of power analysis.

- Read and link the design
- Set input transition and annotate parasitics
- Read switching activity file
- Power analysis

For more information of each command, review the remaining sections of this chapter or check the man page.

```
#####
#      link design
#####
set search_path  "../src/hdl/gate ../src/lib/snps ."
set link_library " * core_typ.db"
read_verilog    mac.vg
current_design  mac
link

#####
#      set transition time / annotate parasitics
#####
set_input_transition 0.1 [all_inputs]
read_parasitics      ../src/annotate/mac.spef.gz

#####
#      read switching activity file
#####
read_vcd          -strip_path tb/macinst ../sim/vcd.dump

#####
#      power analysis
#####
set_waveform_options -file pwr_vcd -effort high
calculate_power       -waveform -statistics
report_power          -file pwr_vcd

#quit
```

Steps for Analyzing Power

For this tutorial, the working directory is `./tutorial/pp_vcd`. The search path is set based on that directory. It should be your current directory. Before running the tutorial, the installation of PrimePower needs to be verified.

Working Directory is pp_vcd

Change your current directory to `./tutorial/pp_vcd`. The search path is set based on this directory.

Checking the PrimePower installation

You can check the installation in following ways.

1. `which pp_shell`

You should get result similar to the following:

- `unix > which pp_shell`
- `/root/synopsys/sparcOS5/syn/bin/pp_shell`

If the file `pp_shell` cannot be found, the installation has a problem. Refer to installation guide for more information.

2. `pp_shell`

The `pp_shell` command will start the PrimePower shell:

```
PrimePower (TM)
  Version V-2004.06 for sparcOS5 -- May 17, 2004
  Copyright (c) 1988-2004 by Synopsys, Inc.
  ALL RIGHTS RESERVED
```

This program is proprietary and confidential information of Synopsys, Inc. and may be used and disclosed only as authorized in a license agreement controlling such use and disclosure.

Welcome to PrimePower!

```
pp_shell>
```

Running PrimePower

To start running PrimePower, type this command:

```
pp_shell -f pp_vcd.tcl
```

PrimePower will run in the batch mode. It stops when all the commands in the script are executed.

Reviewing the Power Report

At the end of the run, PrimePower prints out the power report. It itemizes the power consumption on each instance.

```
-----
Level          Instance_Name (Cell_Name)    #_of_States
-----
Total          Dynamic      Leakage      Switching   Internal    X-tran      Glitch
Power          Power          Power          Power          Power          Power          Power
in Watt        in Watt        in Watt        in Watt        in Watt        in Watt        in Watt
                (% of Tot)    (% of Tot)    (% of Dyn)    (% of Dyn)    (% of Dyn)    (% of Dyn)
-----
                Peak          Peak
                Power          Time Interval
                in Watt        in ns
-----
----- Instances -----
*----- 0  pp_root (.) 1860815
3.792e-03  3.792e-03  2.594e-07  1.529e-03  2.263e-03  4.818e-07  5.400e-05
( 99.99%) ( 0.01%) ( 40.32%) ( 59.68%) ( 0.01%) ( 1.42%)
7.612e-02 1776.01-1776.02
-*----- 1  mac (mac) 1860815
3.792e-03  3.792e-03  2.594e-07  1.529e-03  2.263e-03  4.818e-07  5.400e-05
( 99.99%) ( 0.01%) ( 40.32%) ( 59.68%) ( 0.01%) ( 1.42%)
7.612e-02 1776.01-1776.02
```

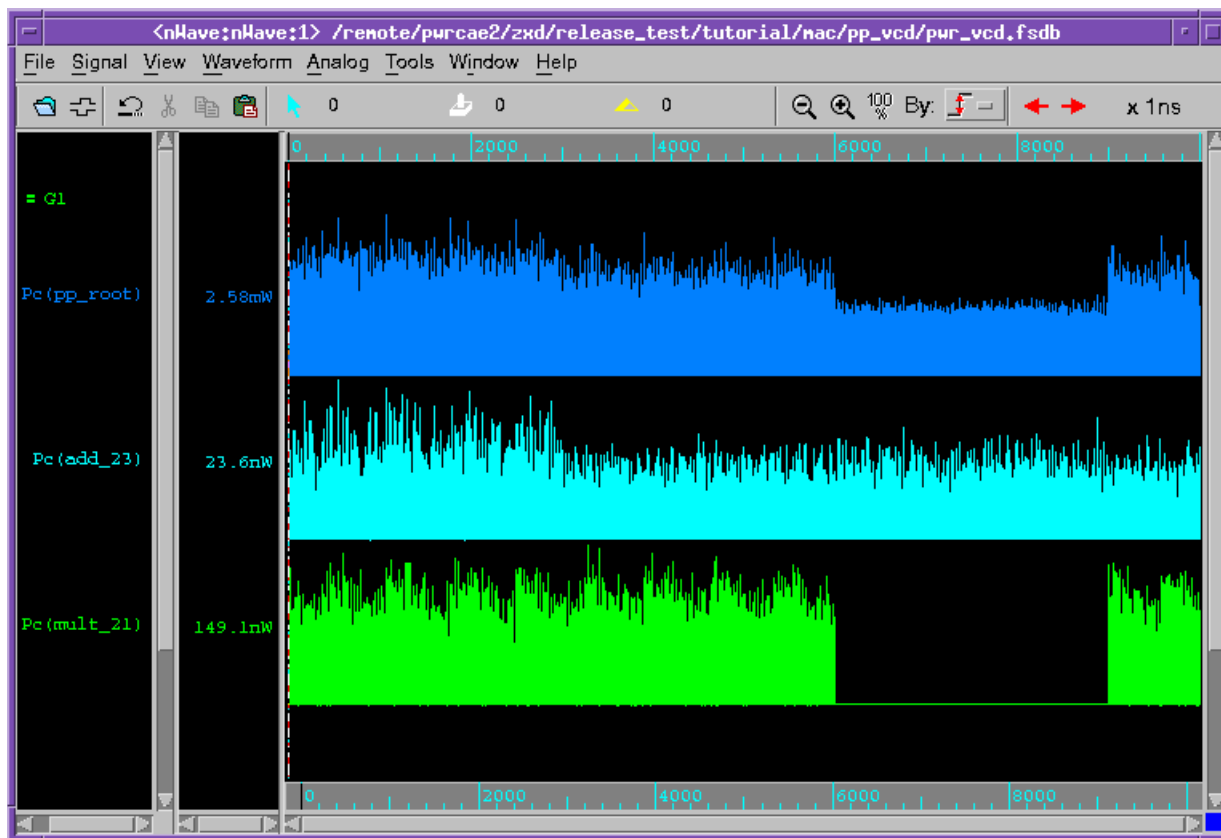
Viewing the Power Waveform

After the power analysis, a power waveform file named `pp_vcd.fsdb` is saved in the current directory. The waveform viewer `turboWave` can be used to view it.

In the Unix shell, type `turboWave &` to start the viewer.

In the tool menu, select *File -> open -> pp_vcd.fsdb* In tool menu, select *Signal -> Get Signals -> Pc(pp_root) -> OK*

Figure 3-1 turboWave waveform viewer



Power Analysis With SAIF

The SAIF file is another form of switching activity. With SAIF, PrimePower can perform average power analysis. This analysis takes less runtime.

SAIF file is generated either from gate level or RTL simulation. RTL SAIF only captures switching activity for part of the design. PrimePower propagates the partial switching activity through out the whole design.

Related Files

In the tutorial directory, the files listed in Table 2 are related to the SAIF based power analysis tutorial.

Table 3-2 Files for the SAIF based power analysis tutorial

File Name	Description
<code>./pp_saif/pp_saif.tcl</code>	PrimePower script to run the analysis with SAIF file
<code>./sim/saif.dump.gz</code>	SAIF file with switching activity
<code>./src/hdl/gate/mac.vg</code>	Gate-level netlist of the design
<code>./src/lib/snps/core_typ.db</code>	Technology library file
<code>./src/annotate/mac.spef.gz</code>	Parasitic file

The SAIF File

This is an ASCII file. The switching activity is represented in the form of toggle rate. The SAIF file is compact compared to the VCD file. It can only be used for average power analysis

PrimePower Script File

The PrimePower script for this tutorial is similar to the one with VCD. The lines with bold font show the differences.

For details of each command, see the man page.

```
#####
#       link design
#####
set search_path    "../src/hdl/gate ../src/lib/snps ."
set link_library    " * core_typ.db"
read_verilog    mac.vg
current_design mac
link

#####
#       clock required only for average waveform
#####
create_clock          -p 12 [get_ports clk]

#####
#       set transition time / annotate parasitics
#####
set_input_transition    0.1 [all_inputs]
read_parasitics          ../src/annotate/mac.spef.gz

#####
#       read switching activity file
#####
read_saif -input      ../sim/mac.saif -instance tb/macinst
report_saif          -missing

#####
#       analyze power
#####
set_waveform_options    -file pwr_saif
calculate_power          -waveform -statistics
report_power            -file pwr_saif

#quit
```

Steps for Analyzing Power

The current working directory should be `./tutorial/pp_saif`.

Running PrimePower

To start running PrimePower, type this command:

```
pp_shell -f pp_saif.tcl
```

PrimePower runs in batch mode. It stops when all the commands in the script are executed.

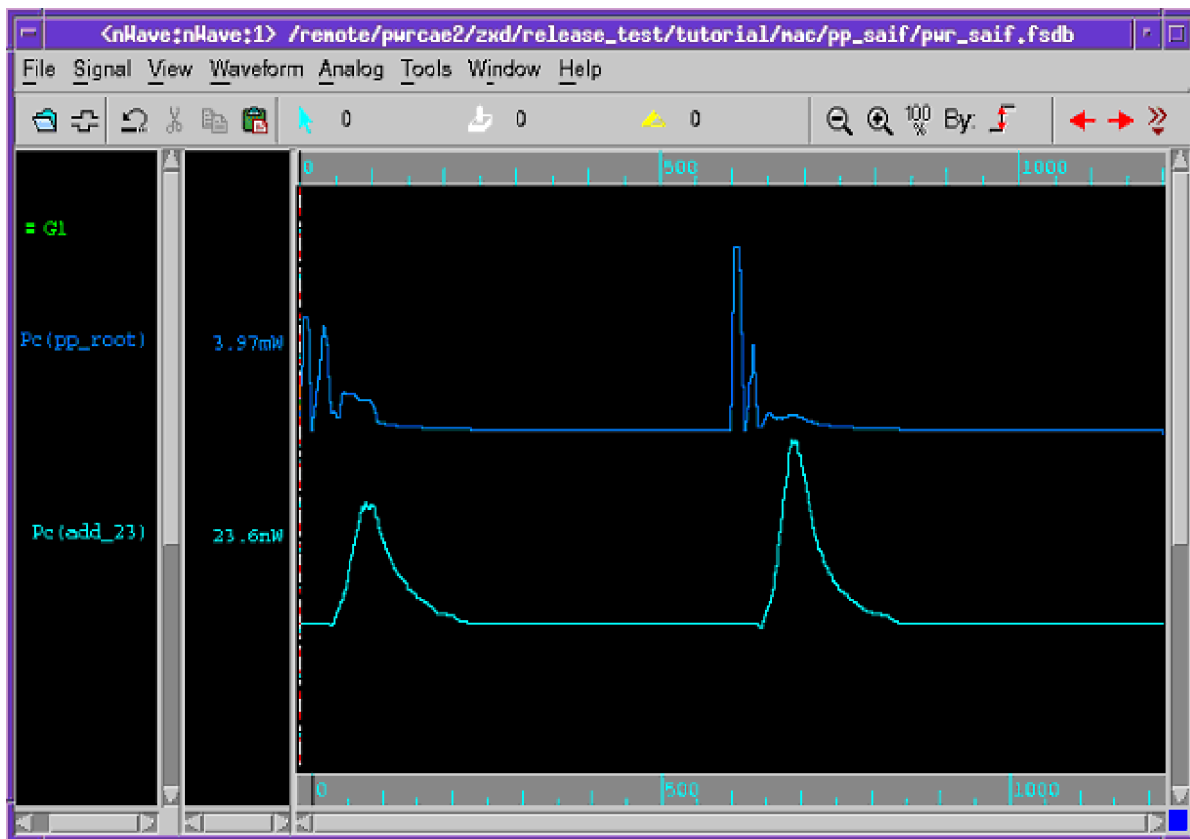
Reviewing the Power Report

This waveform is similar to the one generated with VCD. However, there is no peak power in the report.

Viewing the Waveform

This is an optional step. The waveform is for a single clock cycle. It is based on average power and the time window in each clock cycle when the powers are consumed.

Figure 3-2 Viewing the waveform



Vector Free Power Analysis

With vector free power analysis, users do not need to run simulation. PrimePower will apply default toggle rate on the primary inputs, black box outputs and then propagate them. The default toggle rate can be adjusted. This usage model can be used for quick power estimation.

Related Files

In the tutorial directory, the files listed in Table 3 are related to the vector free analysis tutorial.

Table 3-3 Files for the SAIF based power analysis tutorial

File Name	Description
<code>./pp_vf/pp_vf.tcl</code>	PrimePower script for vector free analysis
<code>./src/hdl/gate/mac.vg</code>	Gate-level netlist of the design
<code>./src/lib/snps/core_typ.db</code>	Technology library file
<code>./src/annotate/mac.spef.gz</code>	Parasitic file

Most of the files are identical to the previous tutorials.

PrimePower Script

This script is slightly different than the previous ones. For switching activity, you can set the toggle rate directly. You can also set the default toggle rate as well.

```
#####
#       link design
#####
set search_path      "../src/hdl/gate ../src/lib/snps . "
set link_library     " * core_typ.db"
read_verilog         mac.vg
current_design       mac
link

#####
#       set transition time / annotate parasitics
#####
set_input_transition  0.1 [all_inputs]
read_parasitics      ../src/annotate/mac.spef.gz
create_clock          -p 12 [get_ports clk]

#####
#       set switching activity
#####
set pwr_default_toggle_rate 0.8
set_switching_activity -toggle_rate 2 -period 10000
[get_ports ct1]
set_switching_activity -toggle_rate 1 -period 10000
[get_ports ct2]

#####
#       power analysis
#####
set_waveform_options  -file pwr_vf
calculate_power        -waveform -statistics
report_power          -file pwr_vf

#quit
```

Steps For Analyzing Power

The current working directory should be ./tutorial/pp_vf.

Running PrimePower

To start running PrimePower, type this command:

```
pp_shell -f pp_vf.tcl
```

PrimePower runs in batch mode. It stops when all the commands in the script are executed.

Viewing the Power Report

This report has the same format as the SAIF-based report. There are average powers, but no peak power.

RTL VCD Power Analysis

Power analysis with RTL VCD files to view the average and peak power is a beta feature in PrimePower 2005.06. A `primepower_beta` license feature is required to perform this analysis.

To view the peak power, users do not need to generate a gate-level VCD file. PrimePower uses the RTL VCD file. The events for the ports and registers are annotated; zero-delay simulation is used to propagate the activity from these events and the timing windows are used to skew them so that they occur at the appropriate times.

In the tutorial directory the files listed in Table 4 are used for the RTL VCD analysis.

Table 3-4 Files for the RTL VCD based power analysis tutorial

File Name	Description
../pp_rtlvcd/pp_rtlvcd.tcl	PrimePower script for rtlvcd analysis
./src/hdl/gate/mac.vg	Gate level netlist for PrimePower
./src/lib/snps/core_typ.db	Synopsys technology library file
./src.annotate/mac/spef.gz	Parasitic file
./sim/runsim_rtl	Runscript to generate RTL VCD file
./sim/rtlvcddump	RTL simulation VCD file
./src/hdl/tb/tb.v	Verilog testbench
./src/hdl/rtl/mac.v	RTL Verilog description for simulation

PrimePower Script

This script varies from the earlier scripts in that the `-rtl_direct` option is used when reading the VCD file, informing PrimePower that it is RTL level.

```
#####
#          link design
#####
set search_path      "../src/hdl/gate ../src/lib/snps ."
set link_library     " * core_typ.db"

read_verilog         mac.vg
current_design        mac
link

#####
#          clock required for average waveform
#####
create_clock          -p 12 -waveform {6 12} [get_ports clk]
```

```
#####
#       set transition time / annotate parasitics
#####
set_input_transition 0.1 [all_inputs]
read_parasitics ../src/annotate/mac.spef.gz

#####
#       read switching activity file
#####
read_vcd -strip_path tb/macinst../sim/rtlvcd.dump -
rtl_direct

#####
#       report
#####

set_waveform_options -file pwr_rtlvcd
calculate_power -waveform -statistics
report_power -file pwr_rtlvcd
#quit
```

Steps for Analyzing Power

The current working directory should be `./tutorial/pp_rtlvcd`

Running PrimePower

To start running PrimePower, type this command:

```
pp_shell -f pp_rtlvcd.tcl
```

PrimePower runs in batch mode and stops when all the commands in the script are executed.

Viewing the Power Report

This report has the same format as the VCD based report. It includes both average and peak power values.

4

Graphical User Interface

The PrimePower graphical user interface (GUI) provides a way to view design data and analysis results in graphical form, including histograms and waveforms.

Analysis Flow With the GUI

The GUI can help you visualize and understand the nature of power problems in the design, including the type, number, magnitude, and locations of the problems. You can use the visual analysis tools after you read, link, and calculate the power of the design.

Typically, an analysis session consists of the following tasks:

1. Read and calculate the power for the design.
2. Start the GUI.

3. Generate a power histogram to get a high-level overview of the power consumption of the design and to identify specific items that contribute to the highest power (such as high activity and capacitance).
4. Generate a toggle histogram to find the instances with the most activity.
5. View the instance and net capacitance profiles to determine the source of high power.
6. Examine the logic vectors and power waveforms to determine the cause of peak power.

Starting and Stopping a GUI Session

Before you start the GUI, make sure that the DISPLAY environment variable is set to your UNIX display name.

To start a PrimePower session that includes the GUI, at the system prompt, enter the following:

```
% pp_shell -gui
```

You can optionally set the DISPLAY environment variable at the same time that you invoke PrimePower GUI. For example:

```
% pp_shell -gui -display 192.180.50.155:0.0
```

You can also start the GUI from a pp_shell session. To start the GUI, use the following command at the pp_shell prompt:

```
pp_shell> gui_start
```

The `gui_start` command does not work when used in a script.

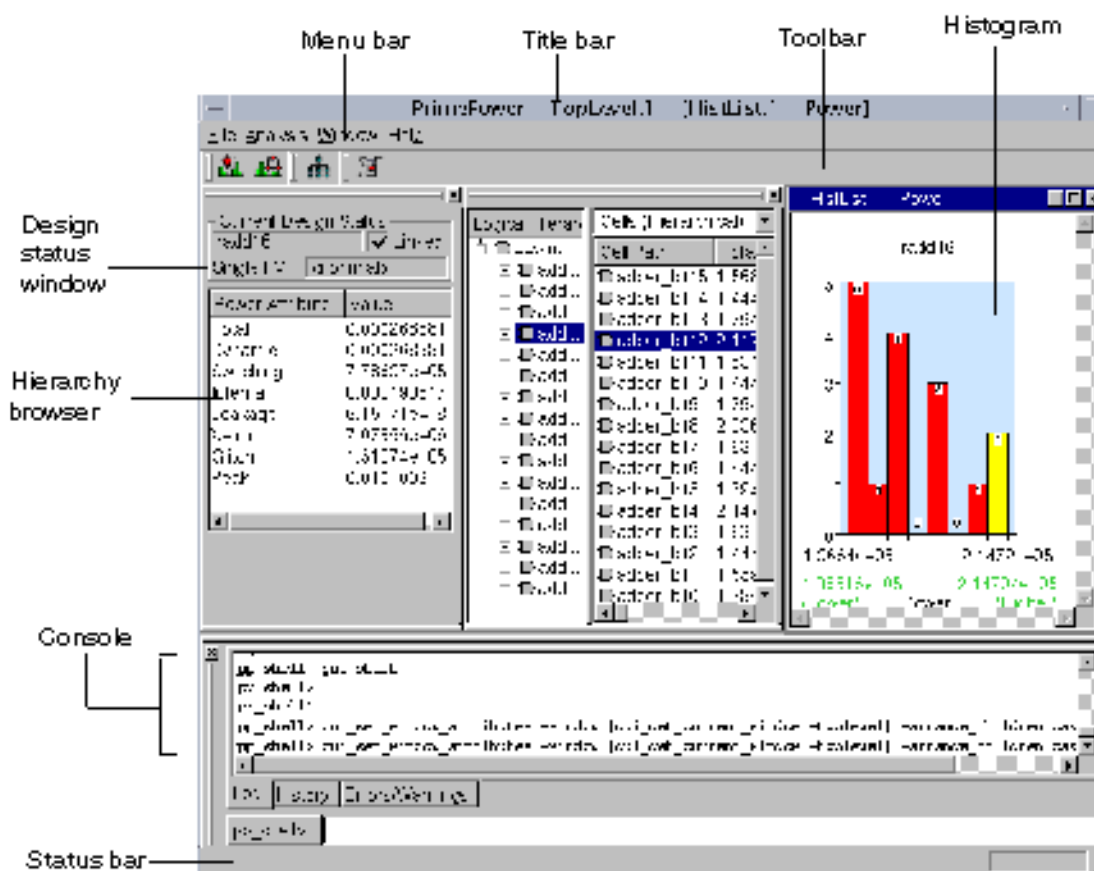
To stop the GUI while still keeping the original `pp_shell` session going in the terminal window, use the `gui_stop` command or the *File > Close GUI* menu command. To exit from PrimePower entirely, use the *File > Exit* command.

Top-Level GUI Window

Starting the GUI opens the top-level GUI window. This is the main PrimePower window in which you can display all types of PrimePower windows. You can open, close, move, resize, dock, undock, maximize, and minimize various types of windows within the top-level window.

[Figure 4-1](#) shows a top-level window containing a menu bar, a toolbar, the console, a hierarchy browser, and a histogram. You can change the appearance by adding, resizing, and removing windows within the top-level window. You can also open multiple top-level windows to create different analysis views.

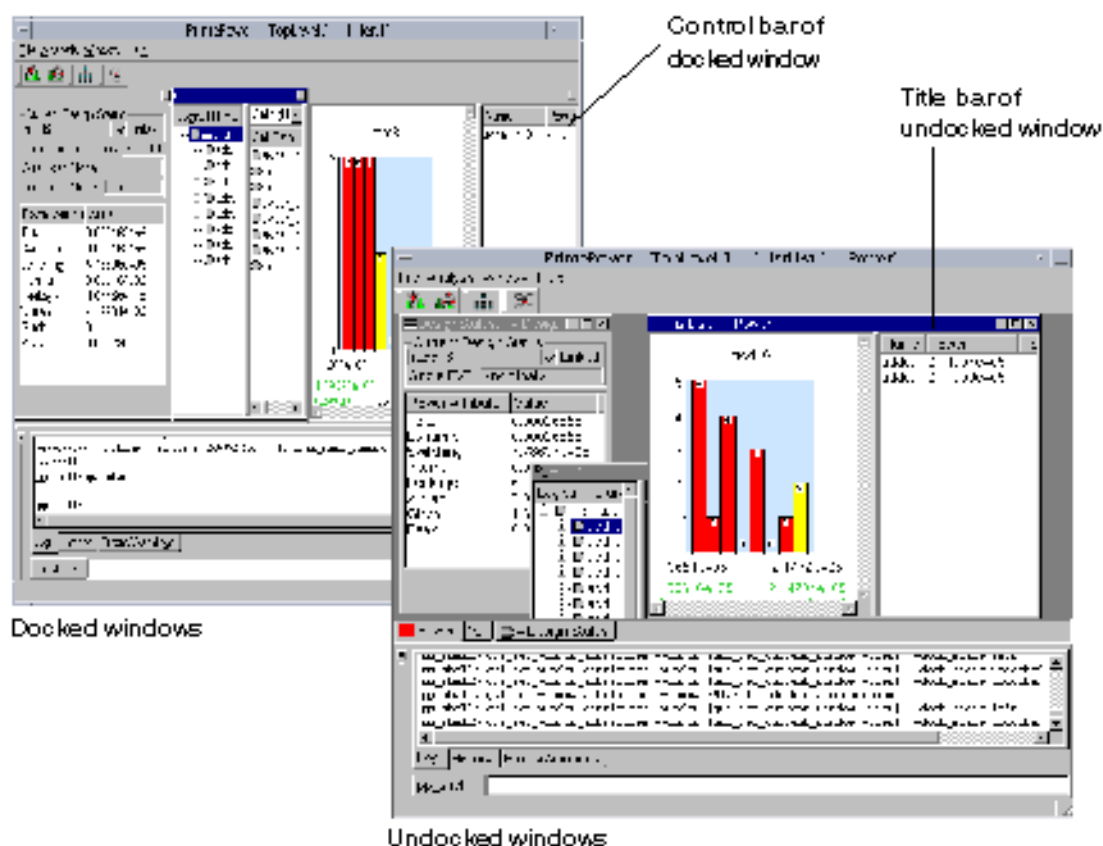
Figure 4-1 Top-Level GUI Window



Manipulating Windows

Each window can be docked, undocked, or maximized. [Figure 4-2](#) shows all windows docked and all windows undocked. A docked window is attached to a wall of the top-level window and cannot be moved freely within the top-level window, whereas an undocked window is not attached and can be moved anywhere within the top-level window. Undocked windows can overlap each other, but docked windows cannot.

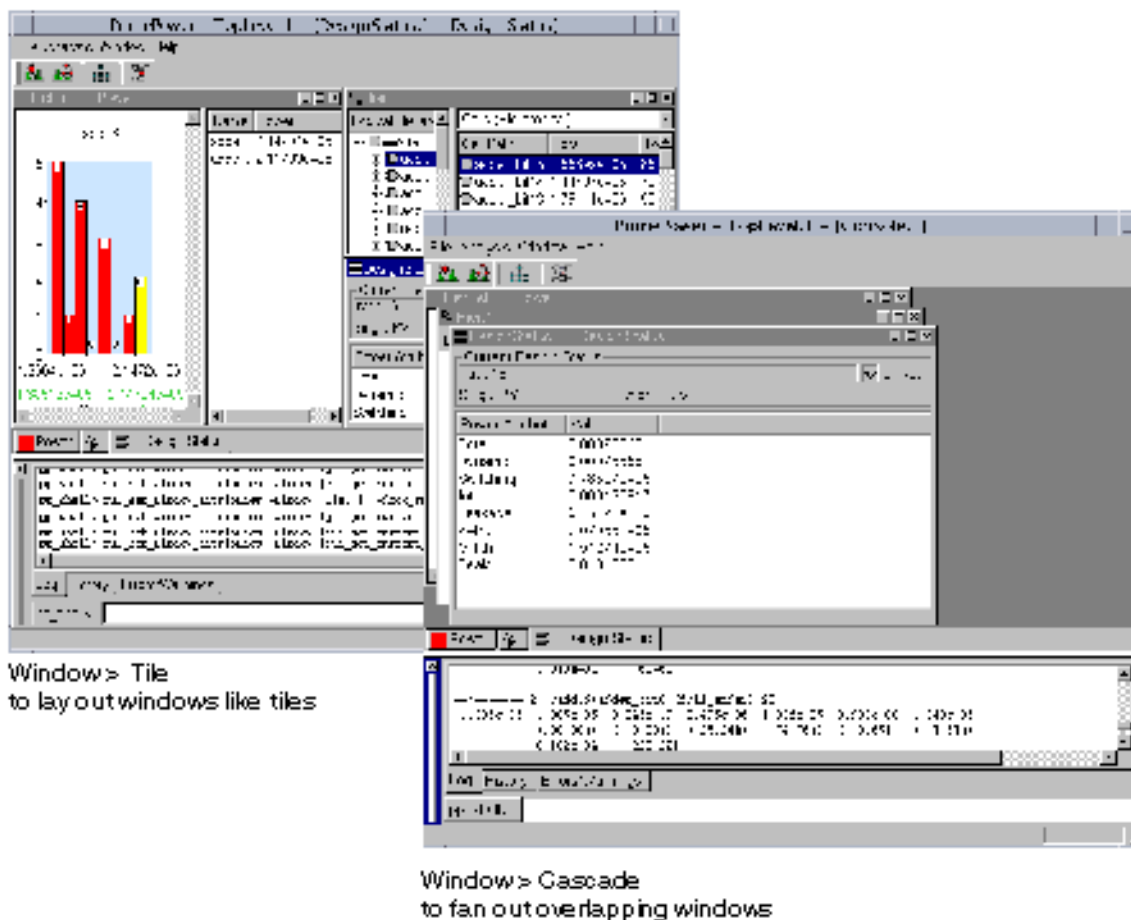
Figure 4-2 Docked and Undocked Console Windows



A docked window has a control bar perpendicular to the side attached to the top-level window. To undock a docked window, double-click the control bar. An undocked window has a title bar with buttons to minimize, maximize, or close the window, and you can freely move and resize the window.

To undock all windows and lay them out evenly like tiles, choose the menu command *Window > Tile*; or to fan them out like a deck of cards, choose *Window > Cascade*. See [Figure 4-3](#).

Figure 4-3 Tiled and Cascaded Windows



You can move, resize, and minimize undocked windows at any time. To move an undocked window, point to the control bar or title bar and drag it to the desired position.

To perform an operation on a window, you need to select it and make it the active window. The control bar or title bar of the currently selected window is highlighted in a distinctive color. If multiple windows are overlapping, the active window appears in front of all others.

To select a window and make it current, point to the window and click. You can also choose the menu commands *Window > Next Window*, *Window > Previous Window*, and *Window > window name*. You can dock and undock windows by choosing *Windows > Dock > ...* and *Windows > Undock*.

You can minimize any window, which means to temporarily turn it into a small icon and otherwise hide it from view. To do so, click the minimize [–] button in the corner of the window (if it is an undocked window), or click the tab at the bottom corresponding to that window. To restore a minimized window, click the corresponding tab at the bottom.

To maximize a window (make it fill the whole top-level window), first undock it if necessary, and then click the maximize button in the corner of the window.

To close a window, click the close [X] button in the corner of the window.

Window Types

[Table 4-1](#) lists and briefly describes the types of PrimePower windows that can be displayed in the top-level window. You can find additional information on each type of window in the remaining sections of this chapter.

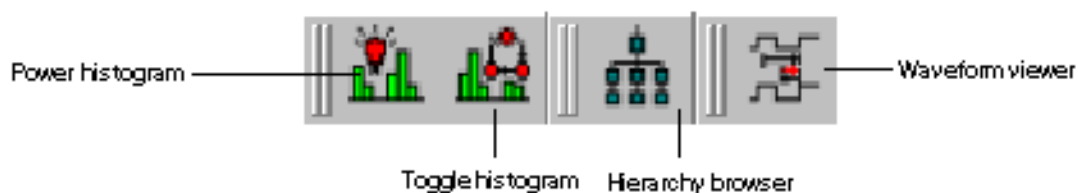
Table 4-1 Types of Windows Within the Top-Level Window

Window type	Description
Console	Lets you run pp_shell commands and view the pp_shell log, command history, or error/warning messages.
Design status	Displays the status of the current design including design name, link status, operating conditions, and wire_load information.
Hierarchy browser	Lets you browse through the design hierarchy and select a cell, net, port, or pin in the design.
Histogram	Displays the distribution of the designing bar graph form for power and toggle data.
Waveform	Invokes turboWave to display the power waveforms.
Operating conditions	Displays operating conditions defined in the libraries.

Toolbars

The toolbar buttons provide quick access to often-used menu selections. [Figure 4-4](#) shows an example of a toolbar.

Figure 4-4 Toolbar Buttons



Different buttons are associated with different windows: one for waveforms, one for histograms, and so on.

Buttons that cannot be used in the current context are disabled and displayed in light gray.

Selecting Objects

You can select objects for subsequent actions by clicking on them.

Menu Commands

[Table 4-2](#) lists and briefly describes the commands available in the menu bar of the top-level window.

Table 4-2 Menu Commands

Menu command	Action taken
File > Execute Script	Executes a script, like the source command.
File > Close GUI	Closes all PrimePower GUI windows, leaving pp_shell active.
File > Exit	Exits completely from PrimePower.

Table 4-2 Menu Commands (Continued)

Menu command	Action taken
Analysis > Browse Hierarchy	Displays a hierarchy browser window, from which you can select objects by traversing the hierarchy.
Analysis > Power Histogram >	Displays a histogram of power data.
Analysis >Toggles Histogram >	Displays a histogram of toggle data.
Analysis > Operating Conditions >	Displays operating conditions defined in the library
Analysis > View Waveforms >	Invokes turboWave to view power waveforms.
Window > New Console	Opens a new console window for running pp_shell commands, viewing the command history, and viewing errors and warnings.
Window > New Design Status View	Opens a new design status window, which shows the design name, the operating conditions, and whether the design has been linked.
Window > Close	Closes the currently active window within the top-level window
Window > Close All	Closes all windows within the top-level window.
Window > Tile	Arranges all windows side-by-side within the top-level window.
Window > Cascade	Cascades all windows into a stack of undocked windows within the top-level window.
Window > Dock >	Docks the currently active window to a specified side of the top-level window.
Window > Undock	Undocks the currently active docked window.
Window > Next Window	Makes the next window active and brings it to the front.
Window > Previous Window	Makes the previous window active and brings it to the front.
Window > <i>window name</i>	Makes the named window active and brings it to the front.

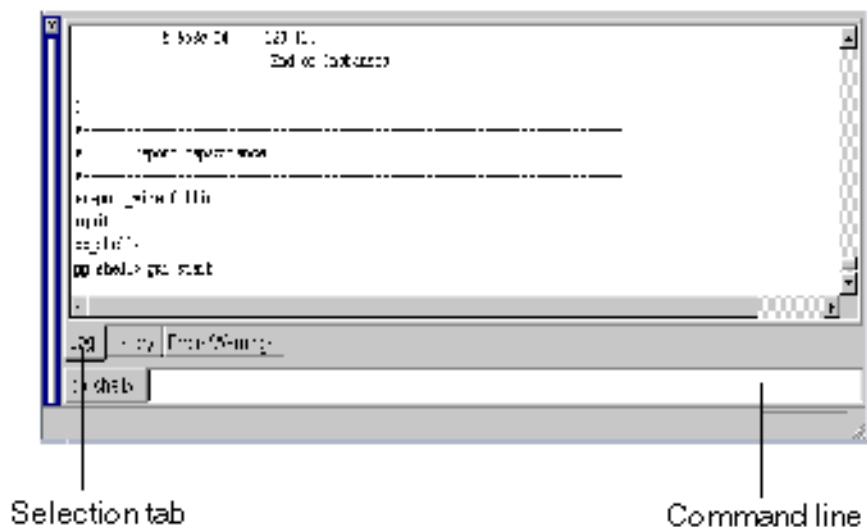
Table 4-2 Menu Commands (Continued)

Menu command	Action taken
Help > Man Pages	Opens a window for displaying man pages.
Help > About	Shows the PrimePower version number and copyright notice.

Console

You can use the console window to execute `pp_shell` commands and view the text-format response from PrimePower, the command history, or error/warning messages. A typical console window is shown in [Figure 4-5](#).

Figure 4-5 Console Window



The command line is the long box at the bottom, to the right of the `pp_shell` prompt. You can enter commands there, just like at the `pp_shell` prompt in a terminal window.

Operation of the console mirrors what is entered and displayed in the `pp_shell` terminal window from which you invoked the GUI. You can enter commands and view the response in either the `pp_shell` terminal window or in the console. The entered commands and the PrimePower response are displayed in both places. If you do not need the console, you can close it and just use the terminal window.

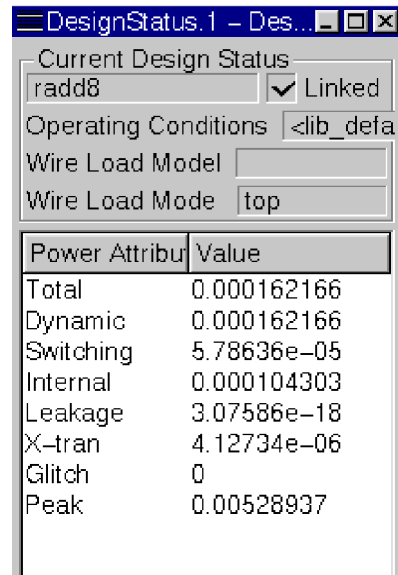
The console offers some views that are not available in the terminal window. The three tabs at the bottom let you select the type of text to display:

- Select the Log tab to display the `pp_shell` text-format response from PrimePower. This is the same as what you see in a `pp_shell` terminal window.
- Select the History tab to view a history of recently executed commands. You can select commands to execute again or copy and paste the commands to create a script.
- Select the Errors/Warnings tab to view the error and warning messages issued in the current session (a subset of the text in the Log view).

Design Status

The current status box shows the status of the current analysis. See [Figure 4-6](#) for an example.

Figure 4-6 PrimePower Design Status Box



Power Attribute	Value
Total	0.000162166
Dynamic	0.000162166
Switching	5.78636e-05
Internal	0.000104303
Leakage	3.07586e-18
X-tran	4.12734e-06
Glitch	0
Peak	0.00528937

The Current Design Status section shows the name of the current design, the Operating Conditions applied, as well as the Wire Load Model and the Wire Load Mode. The Checkbox shows whether the design has been linked.

If the `calculate_power` command has been executed, values are provided for the power attributes of the `current_design`. Otherwise, the Value field is empty.

Histograms

A histogram shows the distribution of design or analysis values in bar-graph form.

Within PrimePower, the following types of information can be viewed as histograms:

- Total Power
- Toggles

The histogram data is available for hierarchical instances, after power calculation is completed. To view Power or Toggle histograms, first select the desired instance in the hierarchy browser; then select *Analysis => Power or Toggle Histogram*. A histogram window appears with the distribution for the selected instance.

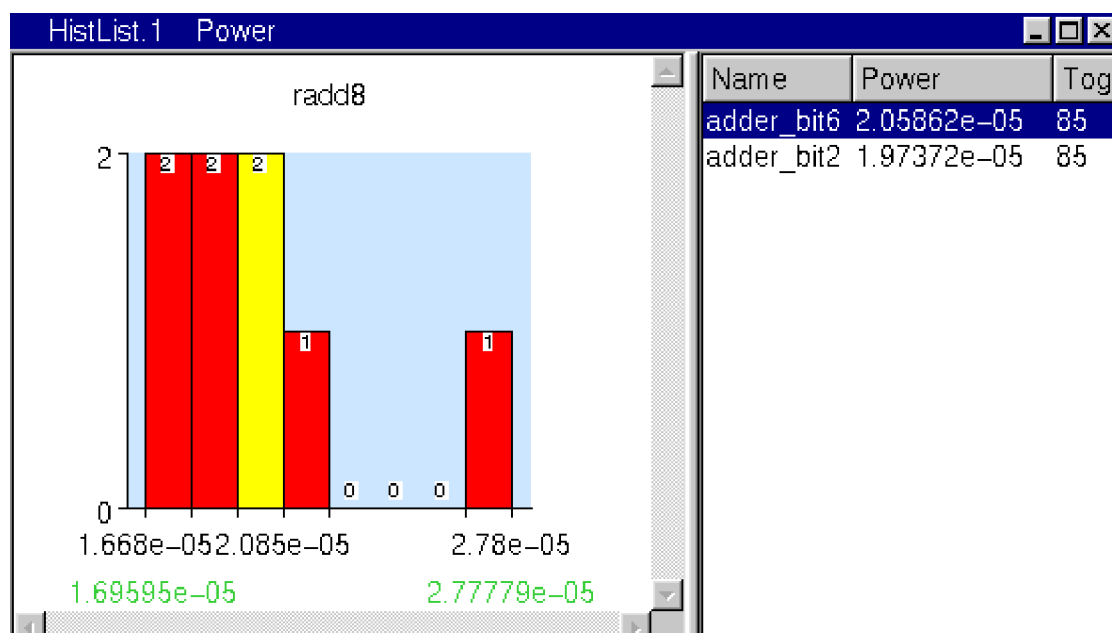
Note:

If no instance is selected, then the histogram for the current design will be displayed.

The power or toggle values are divided uniformly into bins, and the number of instances which fall into each bin is plotted. You can select a bin by clicking it. The selected bin is highlighted in yellow. The pane on the right shows the instances within the bin and their associated values.

Shown below in [Figure 4-7](#) is a sample Power Histogram. Multiple histogram windows can be opened simultaneously.

Figure 4-7 Sample Power Histogram

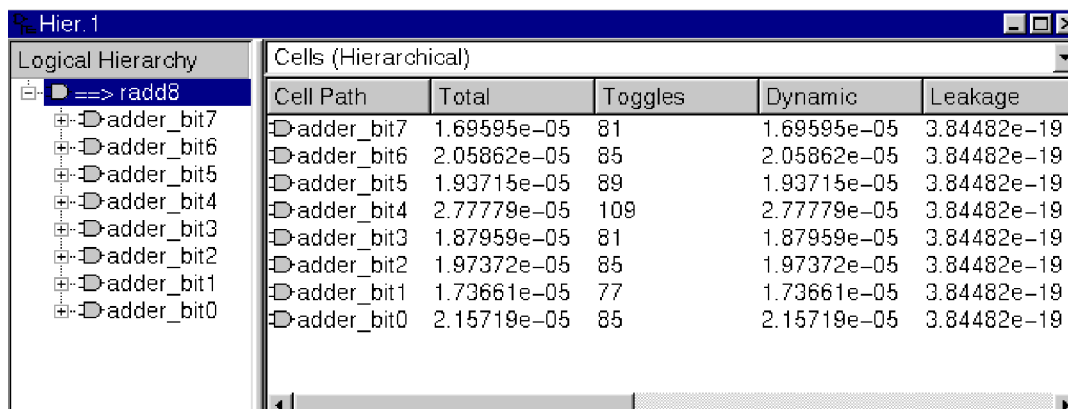


Hierarchy Browser

The hierarchy browser window lets you traverse the hierarchy of the design and select parts of the design for subsequent analysis. If a hierarchy browser window is not already present, you can open one by choosing *Analysis => Browse Hierarchy*.

See [Figure 4-8](#) below, which shows a typical hierarchy browser window.

Figure 4-8 Hierarchy Browser Window



The screenshot shows a window titled 'Hier.1' with two panes. The left pane, 'Logical Hierarchy', shows a tree structure with 'radd8' selected and expanded, revealing sub-items 'adder_bit7' through 'adder_bit0'. The right pane, 'Cells (Hierarchical)', displays a table with columns for 'Cell Path', 'Total', 'Toggles', 'Dynamic', and 'Leakage'. The table lists data for each 'adder_bit' cell.

Cell Path	Total	Toggles	Dynamic	Leakage
adder_bit7	1.69595e-05	81	1.69595e-05	3.84482e-19
adder_bit6	2.05862e-05	85	2.05862e-05	3.84482e-19
adder_bit5	1.93715e-05	89	1.93715e-05	3.84482e-19
adder_bit4	2.77779e-05	109	2.77779e-05	3.84482e-19
adder_bit3	1.87959e-05	81	1.87959e-05	3.84482e-19
adder_bit2	1.97372e-05	85	1.97372e-05	3.84482e-19
adder_bit1	1.73661e-05	77	1.73661e-05	3.84482e-19
adder_bit0	2.15719e-05	85	2.15719e-05	3.84482e-19

The pane on the left, lets you browse the cell hierarchy. Click the [+] button to view the lower-level cells beneath a cell, or [-] to collapse the view of lower-level cells.

When you select a cell in the left pane, the right pane shows the contents of the cell. Click the down arrow to show the types of objects to be listed: hierarchical cells, all cells, pins/ports, pins of child cells, or nets.

Based on the type of object selected, different attributes are displayed in the left pane for the listed objects.

Clicking on the header (e.g. Total) sorts the display based on the values in that column.

5

Design Data

The first step in any power analysis is to load the design description and associated technology libraries. PrimePower accepts design descriptions in .db, .ddc, Verilog, and VHDL formats. It accepts technology libraries in .db format, which are typically provided by the ASIC vendor.

Overview

The reading, reporting, and modification of design data are described in the following sections:

- [Search Path and Link Path](#)
- [Reading Design and Library Data in .db Format](#)
- [Reading Design Data in the .ddc Format](#)

- [Reading Verilog and VHDL Design Data](#)
- [Setting the Current Design and Current Instance](#)
- [Listing Design and Library Information](#)
- [Linking the Design](#)
- [Design Objects](#)
- [Design Information Reports](#)
- [Bus Report](#)

Search Path and Link Path

The `search_path` and `link_path` variables control the directory paths in which PrimePower searches for design data.

The `search_path` variable specifies a list of directory paths that PrimePower uses to find the designs, libraries, and other files. If you include the path to a file in the `search_path` variable, PrimePower can find the file even if you do not specify an absolute path when you read the file. The `search_path` variable typically includes paths to design database files and technology libraries.

The `link_path` variable specifies a list of libraries that PrimePower uses to link designs (in other words, to find and resolve the elements in a design hierarchy). PrimePower searches for design elements in the listed libraries, in the same order that they are listed in the variable. The variable can contain three different types of elements: an asterisk (*), library names, and file names. An asterisk causes PrimePower to search all the designs that have been loaded. PrimePower searches through the loaded designs in the order in which they were read in.

For elements in the `link_path` list other than an asterisk, PrimePower searches for a library that has already been loaded. If that search fails, PrimePower searches for a file name using the `search_path` variable.

In a link operation (invoked by the `link_design` command), the first technology library found in the link path is called the main library because it provides the default values and settings used in the absence of explicit specifications for operating conditions, wire load selection group, wire load mode, and net delay calculation. Relinking a design discards these default settings and reselects the main library, possibly establishing new default values if the `link_path` variable has changed.

To set the `search_path` and `link_path` variables, follow these steps:

1. Set the `search_path` variable with the `set` command. The value should be a list of paths to the directories containing the design files, timing models files, and library files needed for the analysis. Delimit each path with a space and enclose the list in double quotation marks (" "). For example,

```
pp_shell> set search_path ". /u/project/design \  
                        /u/project/library"  
. /u/project/design /u/project/library
```

2. Set the `link_path` variable with the `set` command. The value should be a list of the library names. Use an asterisk (*) to make PrimePower search for designs in memory. Delimit each element with a space and enclose the list in double quotation marks (" ").

```
pp_shell> set link_path "* STDLIB.db"  
* STDLIB.db
```

Reading Design and Library Data in .db Format

To analyze a design, you need to read the design data and associated technology libraries into PrimePower. You can use the `read_db` command to read the technology libraries and design data in .db format. For information on reading design data in other formats, see [“Reading Verilog and VHDL Design Data” on page 5-6](#).

Reading Design Data in .db Format

Follow these steps to load the design information in .db format.

1. Set the search path. For example, enter

```
pp_shell> set search_path ". /u/proj/des1 /u/proj/lib1"
```

PrimePower searches first in the current working directory, then in /u/proj/des1, then in /u/proj/lib1.

2. Set the link path. For example, enter

```
pp_shell> set link_path "* STDLIB.db"
```

The asterisk (*) instructs PrimePower to first search for a design in memory. Then it searches for library cells in the library associated with the STDLIB.db file.

3. Read the technology library into memory. For example, enter

```
pp_shell> read_db STDLIB.db
```

Note that if the STDLIB.db file is in the link path and can be found in the search path, the `link_design` command automatically loads the STDLIB.db file. In that case, it is not necessary to explicitly read the library.

4. Read the design files into memory. For example, enter

```
pp_shell> read_db TOP.db
pp_shell> read_db module1.db
pp_shell> read_db module2.db
pp_shell> read_db module3.db
pp_shell> read_db module4.db
```

Note that if the .db format files are in your search path, you only need to read the `TOP` design. The link step automatically loads subdesigns if the block names are the same as the file names.

5. Link the design to resolve all references in the design. For example,

```
pp_shell> link_design TOP
```

After the files are loaded, you can view a list of the loaded designs or libraries by using the `list_designs` or `list_libraries` command.

Reading Design Data in the .ddc Format

The user can apply the `read_ddc` command to read design files in the Synopsys DDC format. DDC files are generated by Design Compiler or Physical Compiler in XG mode.

By default, attribute information in the DDC files are loaded. If this information is applied through external scripts or other file formats (for example: SDC), the `-netlist_only` option should be used to read the .ddc file.

Reading Verilog and VHDL Design Data

You can use the `read_verilog` or `read_vhdl` commands to read a structural, gate-level netlist into PrimePower. The Verilog or VHDL netlist must contain fully mapped, structural designs. PrimePower cannot link or perform power analysis with netlists that are not fully mapped at the gate level. The netlist cannot contain high-level constructs.

The `read_verilog` or `read_vhdl` command uses the `search_path` variable to find each specified file name. To determine the file that the command will load for given a simple file name, use the `which` command. If you specify an absolute path to a file name, PrimePower loads that file directly and ignores the search path.

After the design files are loaded, you can view a list of the loaded design objects by using the `list_designs` command.

Using the PrimePower Verilog Reader

By default, `read_verilog` invokes the native (built-in) PrimePower Verilog reader. This reader is strictly limited to structural Verilog. Any other constructs are considered syntax errors. The allowed structural constructs include:

- `module`, `endmodule`
- `input`, `output`, `inout`
- `wire`
- `assign`, `tran`

- `supply0`, `supply1`
- `wand`, `wor`, `tri`

The PrimePower Verilog reader allows (but ignores) the following constructs:

- All simulation directives: ``timescale`, ``expand_vectornets`, and so on.
- The `parameter` statement
- The `defparam` statement
- The `specify/endspecify` construct

The Verilog reader supports the `translate_off` and `translate_on` directives. You can turn translation off using one of the following directives:

```
// synopsys translate_off
/* synopsys translate_off */
```

You turn translation back on using one of these directives:

```
// synopsys translate_on
/* synopsys translate_on */
```

Optional Preprocessor

To support certain Verilog directives, you can optionally enable a Verilog preprocessor by setting the `svr_enable_vpp` variable to true. In that case, PrimePower first runs the file through the preprocessor, which scans the file for the following directives:

- ``define`
- ``undef`

- ``ifdef`
- ``include`
- ``else`
- ``endif`

Without the preprocessor, the Verilog reader cannot recognize these directives.

The preprocessor writes a set of intermediate files into the directory specified by the `pt_tmp_dir` variable. When the preprocessor is done, the Verilog reader reads the resulting output. Note that the Verilog reader does not operate as fast with the preprocessor enabled.

When you use the ``include` directive in the Verilog file, PrimePower looks for the referenced files in the directories defined by the `search_path` variable.

Assign Statements and Synonyms

The Verilog reader creates synonyms for discarded names in an assign statement. One net name is chosen, and the others assigned to it are synonyms. During back-annotation, an explicitly named net can be found via one of its synonyms. Although the `get_nets` command finds nets using their synonyms, it cannot find them when mixed with any wildcards. For example, given this assign statement where `n1` wins,

```
assign n1 = n2;
```

using the `get_nets` command finds the real net. For example,

```
pp_shell> get_nets n1
{"n1"}
```

You can find the net using the synonym. Notice that the result is the real net, `n1`, and not the synonym, `n2` (which is not a real net):

```
pp_shell> get_nets n2
{"n1"}
```

You cannot use wildcards with synonyms:

```
pp_shell> get_nets n2*
Warning: No nets matched 'n2*' (SEL-004)
Error: Nothing matched for nets (SEL-005)
```

PrimePower Verilog Reader Limitations

The following limitations apply to the native Verilog reader:

- The `wand`, `wor`, and `tri` statements are essentially synonyms for `wire`. Each creates a wire with no special attributes.
- There is very limited support for global naming, that is, referencing a wire from a different module. For example, `global.gnd` means wire `gnd` in module `global`. Global references are allowed only in instance connections. They cannot be in any other context.

In addition:

- The reference must be to a logic constant.
- Global references cannot be bused.
- The referenced module must be defined in the same file as the module that is referencing it.

- The referenced module must be defined first.
- The global name is used over the default name, but a local name is used over the global name. For example, if `global.gnd` and `1'b0` are used, `gnd` will be the wire name. But if `ZERO` is assigned to `1'b0`, and `global.gnd` is also used, then `ZERO` is used.

Using the HDL Compiler Verilog Reader

Instead of using the native (self-contained) Verilog reader of PrimePower, you can optionally use the HDL Compiler reader, which supports the complete Verilog language. However, note that the native Verilog reader is faster and uses less memory than the HDL Compiler reader, especially for large netlists.

If you need to use the HDL Compiler Verilog reader, use the `-hdl_compiler` option of the `read_verilog` command. In that case, PrimePower starts an external reader program (`ptxr`), which requires an HDL-Compiler license while reading is in progress. After the files are read, the license is released.

Note:

A command failure with the DBR-001 message indicates a possible installation problem. See your system administrator.

Setup Files for the HDL Compiler Verilog Reader

When you use the HDL Compiler Verilog reader, the `ptxr` program reads your `.synopsys_dc.setup` files (not `.synopsys_pp.setup`) to access the HDL Compiler variables. These include the system, home, and local setup files. The system setup file must always be read. However, it is possible (an often preferable) for `read_verilog` to skip the home and local setup files.

You can define the `ptxr_setup_file` variable in PrimePower to reference a ptxr-specific setup file. By default, this variable does not exist. The ptxr setup file is written in Tcl. This file can contain a very limited set of commands:

- Comments
- Blank lines
- Variable assignments

For example, you could create `my_ptxr.setup`:

```
# My ptxr_setup_file
set bus_naming_style "%s(%d)"
set bus_extraction_style "%s[%d:%d]"
```

To use this file:

```
pp_shell> set ptxr_setup_file my_ptxr.setup
my_ptxr.setup
pp_shell> read_verilog -hdl_compiler module1.v
```

To discontinue using the ptxr setup file:

```
pp_shell> unset ptxr_setup_file
```

Note:

Usually, one Control-c entry terminates a command in PrimePower. However, to terminate `read_verilog` when you use the `-hdl_compiler` option, you need to type Control-c three times. This stops the read process without stopping PrimePower.

Example

This example shows the differences in system response using the two Verilog readers.

```
pp_shell> read_verilog newcpu.v
Loading verilog file '/designs/newcpu/v1.6/newcpu.v'
1
pp_shell> remove_design -all
Removing design newcpu...
1

pp_shell> read_verilog newcpu.v -hdl_compiler
Beginning read_verilog...
Loading db file '/release/libraries/syn/standard.sldb'
Loading db file '/release/libraries/syn/gtech.db'
Loading verilog file '/designs/newcpu/v1.6/newcpu.v'
Reading in the Synopsys verilog primitives.
/designs/newcpu/v1.6/newcpu.v:
1
```

Reading VHDL Design Files

The `read_vhdl` command reads structural, gate-level VHDL netlists into PrimePower. Each VHDL netlist must contain structural designs that are fully mapped at the gate level, without any high-level VHDL constructs.

You can use either of two VHDL readers: the VHDL netlist reader or VHDL Compiler. Both readers use the PrimePower external reader, `ptxr`. By default, PrimePower starts the PrimePower VHDL netlist reader, which is the same VHDL netlist reader used by Design Compiler. This reader does not require any other licenses.

If you have a VHDL Compiler license, you can invoke VHDL Compiler from PrimePower using `read_vhdl -vhdl_compiler`. The license is released after the VHDL files are read. This reader can handle files up to about one million lines.

Setup Files for the VHDL Compiler

When you use the VHDL Compiler, the ptxr program reads your .synopsys_dc.setup files (not .synopsys_pp.setup) to access the VHDL variables. These include the system, home, and local setup files. The system setup file must always be read. However, it is possible for read_vhdl to skip the home and local setup files. You can define the ptxr_setup_file variable in PrimePower to reference a ptxr-specific setup file. By default, this variable does not exist. The ptxr setup file is written in Tcl. This file can contain a very limited set of commands: comments, blank lines, and variable assignments.

For example, you could create my_ptxr.setup:

```
# My ptxr_setup_file
set bus_naming_style "%s(%d) "
set bus_extraction_style "%s[%d:%d] "
```

To use this file:

```
pp_shell> set ptxr_setup_file my_ptxr.setup
my_ptxr.setup
pp_shell> read_vhdl module1.vhdl
```

To discontinue using the ptxr setup file:

```
pp_shell> unset ptxr_setup_file
```

Setting the Current Design and Current Instance

The `current_design` command sets or gets the current design. The current design is the working (or focal) design for many PrimePower commands. The combination of the current design and the current instance defines the context for many PrimePower commands.

The syntax is

```
current_design [design_name]
```

If you specify a design, PrimePower sets the current design to that design. This argument can be the name of a design, or a collection containing one design.

If you do not specify a design, PrimePower returns a collection containing the current design.

You can use the `current_design` command as a parameter to other PrimePower commands.

To display designs in PrimePower memory, use the `list_designs` command.

You can only set the `current_design` command before the link command is issued.

Within PrimePower, the `current_design` can be set only once prior to linking the design. Users cannot change it during a PrimePower session.

The `current_instance` command sets the working instance object (cell in the design hierarchy) and enables other commands to be used relative to the instance.

The `current_instance` command differs from the `current_design` command, which changes the working design, then sets the current instance to the top level of the new current design. The combination of the current design and current instance defines the context for many `pp_shell` commands.

The `current_instance` command traverses the design hierarchy similarly to the way the UNIX `cd` command traverses the file hierarchy.

The syntax is

```
current_instance [instance_name]
```

The `current_instance` command operates with a variety of *instance_name* arguments:

- If you do not specify an *instance_name* argument, the focus of PrimePower returns to the top level of the hierarchy.
- If you specify dot (*.instance_name*) as the argument, PrimePower returns the current instance and makes no change.
- If you specify dot dot (*..*) as the *instance_name* argument, PrimePower moves the current instance up one level in the design hierarchy.
- If you specify an *instance_name* argument that begins with slash (/), PrimePower sets both the current design and the current instance.

- If you specify a valid cell at the current level of hierarchy as the *instance_name* argument, PrimePower moves the current instance down to that level of the design hierarchy.
- If you specify multiple cell names separated by slashes, PrimePower traverses multiple levels of hierarchy in a single call to the current instance. For example, `current_instance U1/U2` sets the current instance down two levels of hierarchy.
- You can nest the dot dot (..) directive in complex *instance_name* arguments. For example, the command
`current_instance ../../MY_INST`
attempts to move the context up two levels of hierarchy, then down one level to the `MY_INST` cell.

Examples

To use the `current_instance` command to move up and down the design hierarchy, use a command sequence like the following. The `all_instances` command lists instances of a specified design relative to the current instance.

```
pp_shell> current_design TOP
{"TOP"}

pp_shell> current_instance U1
U1

pp_shell> current_instance "."
U1

pp_shell> query_objects [all_instances ADDER]
{"U1", "U2", "U3", "U4"}

pp_shell> current_instance U3
U1/U3
```

```
pp_shell> current_instance "../U4"  
U1/U4  
  
pp_shell> current_instance  
Current instance is the top-level of design 'TOP'.
```

Listing Design and Library Information

PrimePower provides commands for listing designs and libraries that are loaded.

The `list_designs` command lists the designs that are in PrimePower memory. By default, all designs currently in memory are listed.

- Used with its `-all` option, `list_designs` command lists the designs instantiated in the current design but removed from memory.
- Used with its `-only_used` option, `list_designs` filters out the display designs that are not in use. Designs in use include the current design, linked or partially linked designs, and designs instantiated in a linked design.

The notation for the status of the design is as follows:

- * – The design is the current design.
- L – The design is linked.
- N – The design is not in memory.
- l – The design is partially linked.

Examples

To list the designs in memory, enter

```
pp_shell> list_designs
Design Registry:
    AD4FULA    /u/designs/top.db:AD4FULA
    AD4PG      /u/designs/top.db:AD4PG
    ADD5A      /u/designs/add2.db:ADD5A
    MULTI      /u/designs/top.db:MULTI
```

To list the designs in use and in memory, enter

```
pp_shell> list_designs -only_used
Design Registry:
  *L AD4FULA      /u/designs/top.db:AD4FULA
    ADD5A        /u/designs/add2.db:ADD5A
    MULT1        /u/designs/top.db:MULT1
```

The `list_libraries` command lists the libraries in memory.

You can use the `-only_used` option to filter unused libraries out of the display. A library is in use if a linked design links to library cells from the library.

The status of the library is reported as a single character:

“*” (asterisk) indicates the main library of the current design; the first library in the link path that has a time unit.

Examples

To list all the libraries in memory, enter

```
pp_shell> list_libraries
Library Registry:
    my_lib      /u/lib/my_lib.db:my_lib
    tech1       /u/lib/tech1.db:tech1
```

After you link a design, you can identify the main library using the `-only_used` option. This option limits the display to the libraries used to link the current design. For example, enter

```
pp_shell> link_design top_flat
Linking design top_flat...
Designs used to link top_flat:
    <None>
Libraries used to link top_flat:
    tech1
Design 'top_flat' was successfully linked.
1
pp_shell> list_libraries -only_used

Library Registry:
*   tech1           /u/lib/tech1.db:tech1
```

Linking the Design

To produce a design that is ready for power analysis, link your design using the `link_design` command. Linking the design resolves references to library cells and subdesigns.

Example

To link a design called `example`, enter

```
pp_shell> link_design example
Loading db file '/unit/libraries/library.db'
Loading db file '/unit/designs/A.db'
Loading db file '/unit/designs/B.db'

Designs used to link example:
    A, B

Libraries used to link example:
    library

Design 'example' was successfully linked.
```

If you have not read all the subdesign or library files into memory, PrimePower attempts to load them automatically during linking. This process is known as an autoloading, which is dependent on the values of the `search_path` and `link_path` variables.

By default, the linker automatically creates black boxes for unresolved references. A black box is essentially an empty cell with no timing arcs. The result is a completely linked design on which analysis can be performed. You can disable automatic black box creation by setting the variable `link_create_black_boxes` to false.

As of 2003.12, PrimePower by default, applies the `-remove_sub_designs` option to the `link_design` command.

Many commands (such as `report_power`) require a linked design and attempt to automatically link the design if you have not done an explicit link. This process is known as an autolink.

Note:

Once the design is linked, the user cannot reset the current design (even if the `-keep_sub_designs` option is specified to the `link` command).

Per-instance Link Paths

By default, the `link` command removes all designs except the current designs to save the memory. An option, `-keep_sub_designs`, can be used to keep all designs in the memory.

You can have PrimePower use different library cells for different instances of the same cell in the design. To do so, set the variable `link_path_per_instance` to a list, with each list element consisting of a list of instances and the corresponding link path to use for those instances. For example:

```
set link_path_per_instance [list \  
    [list {U1 U5 U7} "*"] \  
    [list U1/U2 "*" lsi_10k_fast] \  
    [list U1/U3 "*" lsi_10k_slow]]
```

For more information, see the man page for the variable.

Link Errors

Certain link errors are serious enough to cause the link to fail. When the `link_design` command fails, the output is a design with unresolved references. This design is not ready for power analysis, and some action is necessary to continue.

There are two ways for the linker to fail to resolve a reference:

1. The reference cannot be found. In this case, a black box is created when `link_create_black_boxes` is true.
2. The reference was found, but the linker detected errors in trying to resolve the reference. In this case, a black box is not created, and the link fails.

There are various errors that can cause the link to fail to resolve a reference. The most typical errors are

- The instance simply has too many ports. An example is an instance of BOX that has 6 ports, but the reference has only 5.

- The instance has a pin that does not exist on the reference (library cell or design). A simple example would be an instance of a BOX with pins A, B, and X, where the reference has ports A, B, C, and Z. Generally, the reference can have more ports than the instance. This can be dependent on the source of the netlist. For example, in Verilog, the following instance of the previously described BOX would be fine:

```
BOX u1 (.A(n1), .Z(out));
```

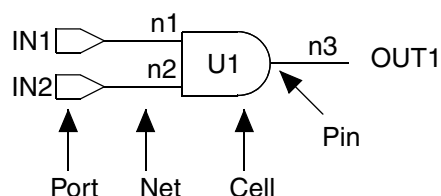
- The instance has scalar pins like `A[0]` and `A[1]`, but the reference has a bus A. Sometimes this indicates a library or subdesign that is out of sync with the top design, which may have been flattened.

Using information from the error messages, you can construct Tcl procedures to display the name and direction of each port on a design or library cell, and each pin on an instance. With these displays, you should be able to detect the problem and determine if the instance or the reference needs to be corrected.

Design Objects

ASIC designs are hierarchical entities composed of objects such as cells, ports, and nets (see [Figure 5-1](#)). To perform detailed power analysis and locate the source of power problems, you need access to these objects. PrimePower lets you create a collection of one or more objects for querying and reporting.

Figure 5-1 Typical ASIC Design Objects



Most PrimePower commands operate on a collection of objects. To create a collection of objects, use the appropriate “get” command: `get_ports`, `get_nets`, `get_clocks`, and so on. The result of a “get” command can be nested within another command that operates on the objects. For example:

```
pp_shell> set_input_delay 2.3 [get_ports IN*]
```

State Objects

PrimePower uses these state objects:

- Current design

The design that is the current top of the hierarchy.

Most objects are referenced relative to the current design.

The `current_design` command sets the working design for many PrimePower commands.

- Current instance

The instance (or hierarchical cell) that is the current scope within the current design. Traverse the hierarchy by changing the current instance. The `current_instance` command sets the working instance object (cell) and enables other commands to be used relative to a specific instance in the design hierarchy.

Netlist Objects

PrimePower commands, attributes, and assertions are directed toward a netlist object. You can create collections of netlist objects.

PrimePower uses these netlist objects:

`cell`

The instances in the design, including those that reference hierarchical blocks and primitive library cells.

`clock`

Clock objects do not exist in a design by default. They are created when you define the clocks for PrimePower. Clocks are also imported from .db format files.

`design`

A design.

`lib`

A library.

`lib_cell`

The cells in a technology library.

`lib_pin`

The pins on library cells.

net

The nets in the current design.

pin

The pins of lower-level cells in the design. Pins can be input, output, or inout (bidirectional).

port

The ports of the current design. Ports can be input, output, or inout (bidirectional).

register

The primitive instances of a design that have clocks defined on them. These instances include flip-flops and latches.

Design Information Reports

Design information reports show information about objects and assertions within a design. PrimePower can display information about

-Design attributes	-Ports
-Cells	-Nets
-Clocks	-Busses

Displaying the Design Attributes

The `report_design` command provides a summary of the current design attributes, including

- Operating conditions
- Wire load models
- Design rules
- Derating factors

Example

To display information about the current design, enter

```
pp_shell> report_design
```

PrimePower displays a report similar to this:

```
*****
Report : design
Design : counter
*****
  Design Attribute                               Value
-----
Operating Conditions:
  operating_condition_max_name      WCCOM
  process_max                       1.5
  temperature_max                   70
  voltage_max                       4.75
  tree_type_max                     worst_case

Wire Load :                               (use report_wire_load for more
information)
  wire_load_mode                    segmented
  wire_load_model_max               70x70
  wire_load_model_library_max       tech_lib
  wire_load_selection_type_max      user-specified
  wire_load_selection_group_max     --
  wire_load_min_block_size          0

Design Rules:
  max_capacitance                   15
  min_capacitance                   --
  max_fanout                        20
  min_fanout                        --
  max_transition                     3.45
  min_transition                     0.1
  max_area                          --

Timing Ranges:
  fastest_factor                     --
  slowest_factor                     --
```

Displaying Clock Information

The `report_clock` command displays information about all clocks in the current design. PrimePower can also report on specific clocks.

Example

To display information about the clocks in a design, enter

```
pp_shell> report_clock
```

PrimePower displays a report similar to this:

```
*****
Report : clock
Design : TOP
*****
Attributes:
  p - propagated_clock
  G - generated clock
```

Clock	Period	Waveform	Attrs	Sources

CLK	20.00	{5 15}		{CLK}
CLKDIV	40.00	{5 25}	G	{INFF1/Q}

Generated Clock	Master Source	Generated Source		Waveform Modification

CLKDIV	C1	INFF1/Q		divide_by(2)

Displaying Cell Information

The `report_cell` command provides information about all cells in the current instance. PrimePower can also display information on a specific cell or set of cells.

This report includes the

- Corresponding cell reference (the library cell name)
- Library name containing the cell
- Area of the cell

Default Report

To create a default report, enter

```
pp_shell> report_cell
```

PrimePower displays a report similar to this:

```
*****
Report : cell
Design : TOP
*****
```

Cell	Reference	Library	Area	Attributes

-				
U1	AN2	tech_lib	2.00	
ff1	FF	tech_lib	7.00	
ff2	FF	tech_lib	7.00	
ff3	FF	tech_lib	7.00	
ff4	FF	tech_lib	7.00	
u4	BOTTOM	TOP	16.00	

Connectivity Report

To report connectivity information for each cell in a design, enter

```
pp_shell> report_cell -connections
```

PrimePower displays a report similar to this:

```
Connections for cell 'ff1':
Reference:      FF
Library:       tech_lib

Input Pins      Net
-----
D              I1
CP            CP

Output Pins     Net
-----
Q             <no_net>
QN          net11
```

Verbose Report

The `-verbose` option and the `-connections` option display additional attributes and show pins on the other side of the net connected to cell pins. For example, enter

```
pp_shell> report_cell -connections -verbose
```


PrimePower displays a report similar to this:

Connections for cell 'ff1':

Reference:FF
Library:tech_lib
Area:7

Input	Pins	Net	Net Driver	Pins	Driver	Pin	Type
D	I1	I1					Input Port
CP	CP	CP					Input Port

Output	Pins	Net	Load	Pins	Load	Pin	Type
Q	<no_net>						
QN	net11	U1/B					Input Pin (OR2)

Port/Net Reports

Parasitics can be obtained via the commands which report information on the ports and nets within the design.

Displaying Port Information

The `report_port` command shows information about ports in the design.

Default Report

By default, the `report_port` command displays the port name, direction, pin capacitance, and wire capacitance for the ports in the current design. To create a default report, enter

```
pp_shell> report_port
```

PrimePower displays a report similar to this:

```
*****
Report : port
Design : TOP
*****
```

Port	Dir	Pin Cap	Wire Cap
C1	in	0.0000	0.0000
C2	in	0.0000	0.0000
D1	in	0.0000	0.0000
D2	in	0.0000	0.0000
OUT1	out	2.3000	0.0000
OUT2	out	2.3000	0.0000

Verbose Report

A verbose report provides more details than a default report. To create a verbose report, enter

```
pp_shell> report_port -verbose
```

PrimePower displays a report similar to this:

```
*****
Report : port
        -verbose
Design : TOP
*****
```

Pin Port	Wire Dir	Cap	Cap
C1	in	0.0000	0.0000
C2	in	0.0000	0.0000
D1	in	0.0000	0.0000
D2	in	0.0000	0.0000
OUT1	out	2.3000	0.0000
OUT2	out	2.3000	0.0000

Port	Max Cap	Min Cap	Max Trans	Min Trans	Max Fanout	Min Fanout	Fanout Load
C1	--	--	--	--	--	--	n/a
C2	--	--	--	--	--	--	n/a
D1	--	--	--	--	--	--	n/a
D2	--	--	--	--	--	--	n/a
OUT1	--	--	--	--	n/a	n/a	0.0000
OUT2	--	--	--	--	n/a	n/a	0.0000

External Fanout		
Port	Number Points	Wire Load Model
C1	0	--
C2	0	--
D1	0	--
D2	0	--
OUT1	3	20x20: tech_lib
OUT2	3	20x20: tech_lib
Input Delay		

Input Port	Min		Max		Related Clock
	Rise	Fall	Rise	Fall	
C1	--	--	--	--	--
C2	--	--	--	--	--
D1	2.10	2.10	2.10	2.10	C1
D2	2.10	2.10	2.10	2.10	C1

Input Port	Resistance		Transition	
	Rise	Fall	Rise	Fall
--	--	--	--	--
C1	--	--	0.00	0.00
C2	--	--	0.00	0.00
D1	--	--	--	--
D2	--	--	--	--

Displaying Net Information

The `report_net` command shows information about nets in the current instance, including

- Fanout and fanin
- Capacitance and resistance
- Pins
- Net connections

PrimePower can also display a specific net in the design.

Default Report

By default, the `report_net` command displays brief information about all nets in the current instance. To create a default report, enter

```
pp_shell> report_net
```

PrimePower displays a report similar to this:

```
*****
Report : net
Design : TOP/u4 (BOTTOM)
*****
Attributes:
  c - annotated capacitance
  r - annotated resistance
```

Net	Fanout	Fanin	Cap	Resistance	Pins	Attributes
CP	4	1	4.00	0.00	5	c
I1	2	1	2.00	0.00	3	c
O1	2	1	2.00	0.00	3	
net11	1	1	1.00	0.00	2	
net12	1	1	1.00	0.00	2	

Total 5 nets	10	5	10.00	0.00	15	
Maximum	4	1	4.00	0.00	5	
Average	2.00	1.00	2.00	0.00	3.00	

Note:

The "c" attribute informs the user that the capacitance value was back-annotated via the `read_parasitics` or the `set_load` command. If it is not present, the value was obtained from the `wire_load` model.

Connectivity Report

The `-connections` option creates a connectivity report for the nets in the design. For example, enter

```
pp_shell> report_net -connections
```

PrimePower displays a report similar to this:

Connections for net 'u4/net11':

Driver Pins	Type
-----	-----
u4/ff1/QN	Output Pin (FF)

Load Pins	Type
-----	-----
u4/U1/B	Input Pin (OR2)

Verbose Report

The `-verbose` option produces a verbose report that displays additional attributes. For example, enter

```
pp_shell> report_net -connections -verbose
```

PrimePower displays a report similar to this:

Connections for net 'u4/net11':

pin capacitance:	1
wire capacitance:	0
total capacitance:	1
wire resistance:	0
number of drivers:	1
number of loads:	1
number of pins:	2

Driver Pins	Type	Pin Cap
-----	-----	-----
u4/ff1/QN	Output Pin (FF)	0

Load Pins	Type	Pin Cap
-----	-----	-----
u4/U1/B	Input Pin (OR2)	1

Bus Report

The `report_bus` command reports information about buses (pins or ports) in the current instance or current design.

To generate a report about buses in the design, enter

```
pp_shell> report_bus
```

PrimePower displays a report similar to this:

```
*****  
Report : bus  
Design : bus_design  
*****
```

Bussed Port	Dir	From	To	Width
-----		-----		-----
acnt	out	15	0	16
baddr	out	15	0	16
bwordreg	out	15	0	16
caddr	out	15	0	16
comreg	out	7	0	8
cs_baddr	in	1	0	2
cs_bword	in	1	0	2
cs_caddr	in	1	0	2
cs_channel	in	3	0	4
cs_cword	in	1	0	2
cwordreg	out	15	0	16
dack	out	3	0	4
din	in	7	0	8
dreq	in	3	0	4
drqreg	out	3	0	4
modreg	out	7	0	8
mskreg	out	3	0	4
srqreg	out	3	0	4
stareg	out	3	0	4
tmpreg	out	7	0	8
wcnt	out	15	0	16

6

Design Parasitics

Design parasitics contribute heavily to the power consumption of a design. By default, PrimePower determines the net parasitics from the wire-load model specified in the Synopsys design library; although several commands are available to allow the user to modify the default parasitic annotation.

Overview

The following sections cover the mechanisms for specifying and reporting design parasitics:

- [Modeling Wire Loads](#)
- [Specifying Port/Net Loads](#)
- [Back-annotated Parasitics](#)

- [Parasitic Reports](#)

Modeling Wire Loads

PrimePower uses wire load models to estimate the capacitance, resistance, and area of nets before floor planning or layout. In the absence of physical design information, PrimePower uses statistically generated wire load models to estimate wire lengths. The wire load models, provided in the technology library, define a fanout-to-length relationship.

PrimePower can automatically select wire load models. See [“Selecting the Wire Load Model Automatically” on page 6-14](#).

PrimePower estimates the wire length of pin-to-pin connections, based on fanout count, then uses that estimate to calculate the effects of cell placement and wire routing.

- Fanout is the total number of pins on the net, excluding the driver pin. The wire load definition contains one or more fanout length pairs.
- If fanout values are not consecutive integers, PrimePower uses linear interpolation to calculate the intermediate `fanout_length` pairs.
- If PrimePower encounters a fanout count greater than the largest `fanout_length` pair, it uses the slope value to extrapolate the wire length.

The wire load is defined by the `wire_load` specification in the technology library.

PrimePower determines which wire load model to use for a design in one of the following ways:

- According to the wire load model you define with the `set_wire_load_model` command.
- According to the wire load indicated by the technology library's `wire_load_selection` group (in the absence of an explicitly set wire load model).
- According to the wire load model identified by the `default_wire_load` attribute in the library (in the absence of both preceding factors).

If none of the preceding are defined, no wire load model is used. If no wire load model is used, the timing information from synthesis is optimistic; that is, your nets have no loading and propagation times. The assumed net resistance and capacitance values will be zero and the net delays and switching power for the net will also be zero. With no wire load model selected, PrimePower does not have complete information about the behavior of your target technology.

You can direct PrimePower to use different wire load models for minimum and maximum timing delay and power analysis.

Calculating the Capacitance of a Net

To calculate the capacitance of a net,

1. Determine the fanout of the net.
2. Look up the length from the `fanout_length` pairs in the `wire_load` model.
3. Calculate the capacitance by multiplying the length by the capacitance coefficient.

Example

```
wire_load("90x90") {  
  
    capacitance : 2.0 ; /* C per unit-length */  
  
    resistance :100.0 ; /* R per unit-length */  
  
    area : 0.5 ; /* net-area per unit-length */  
  
    slope : 1.5 ; /* extrapolation slope */  
  
    fanout_length(1,1) ; /* fanout_length pairs */  
  
    fanout_length(2,2.2);  
  
    fanout_length(3,3.3);  
  
    fanout_length(4,4.4);  
  
}
```

To determine lumped net capacitance, total net resistance, and the design area due to the net, PrimePower multiplies the capacitance, resistance, and area scaling factors by the estimated wire length.

PrimePower calculates the net fanout value as the total number of pins on the net excluding one driver pin. For example, on a net with 2 fanin pins and 3 fanout pins, the fanout value is

$$(2 + 3) - 1 = 4$$

The lumped net capacitance is calculated by multiplication of the estimated wire length by the capacitance factor 2.0. For a fanout value of 4, the last fanout_length defines the wire length as 4.4, resulting in a calculated lumped net capacitance of

$$2.0 * 4.4 = 8.8$$

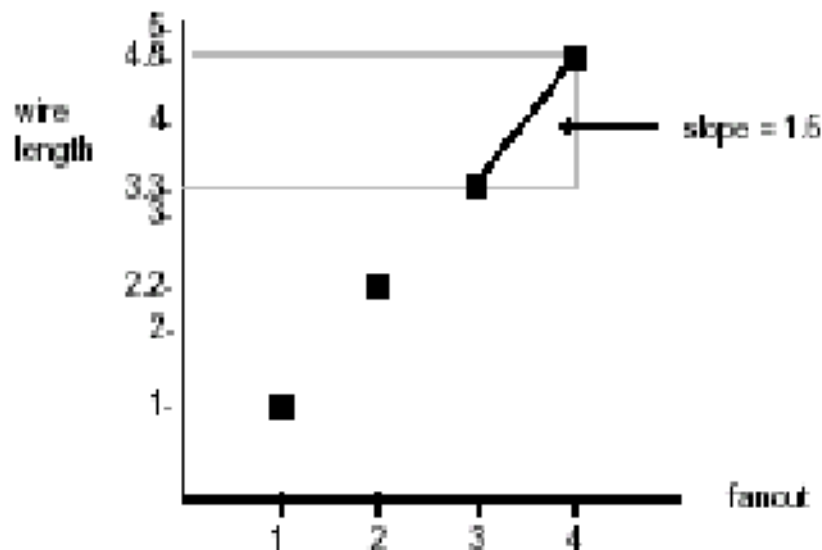
The total net resistance is calculated by multiplication of the estimated wire length by the resistance factor, yielding a value of

$$4.4 * 100.0 = 440.0$$

When calculating pin-to-pin RC delays, PrimePower interprets the total net resistance on the basis of the current operating condition's tree_type attribute.

Figure 6-1 shows how PrimePower calculates wire loads.

Figure 6-1 Wire Load Calculation



Using the previous wire load model for a net with a fanout value of 8 and a slope value of 1.5, PrimePower calculates the wire length and capacitance as follows:

$$\text{length} = 4.4 + ((8 - 4) * 1.5) = 10.4$$

$$\text{capacitance} = 10.4 * 2.0 = 20.8$$

Note:

Define a positive slope value. A positive slope ensures that PrimePower takes into account that adding fanouts to a net increases its length (and therefore its delay and area).

Choosing a Wire Load Model

Which wire load model you choose depends on how the design is ultimately implemented. Hierarchy restrictions imposed on layout results in more-accurate wire load estimates.

Wire load models are typically provided for different sizes of designs based on the observation that same-size blocks generally exhibit similar fanout-to-length relationships. A wire load model must completely contain the design block under consideration.

- If a design is not limited to one contiguous region of the physical chip, the wire load model must reflect that its nets can span the whole chip. In this case, the size of the block that completely contains the design is identical to the size of the chip.
- If a design is limited to a contiguous region of the physical chip, use a wire load model created for designs of the same size. Nets fully enclosed within the design are typically shorter and more predictable than those that potentially span the whole chip.

Another criterion for selecting a wire load model is the degree of pessimism used in its creation. When the potential exists for significantly underestimating the length of a net, use a wire load model that employs pessimism. By using a more pessimistic wire load model in the appropriate situations, you are less likely to have design violations after layout. Consult your Synopsys library supplier for details about how the wire load models provided with the library are created.

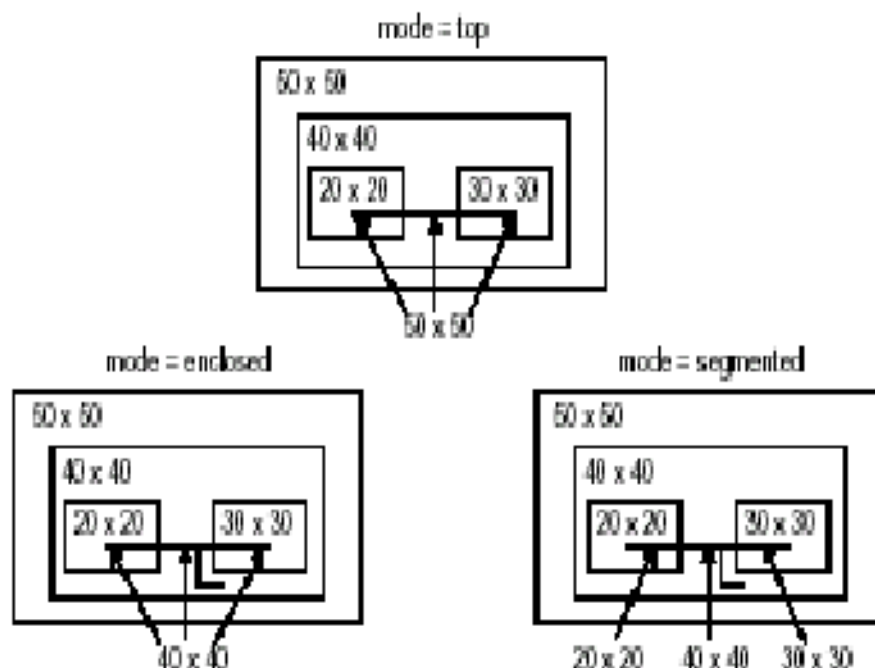
- For large blocks, use wire load models that are more pessimistic in their fanout-to-length estimates than those for small blocks, because the net lengths in large blocks vary more than in small blocks.
- For large fanouts, use fanout-to-length estimates that are more pessimistic than for small fanouts, because the net lengths for high fanout nets are harder to predict accurately.

Wire Load Model Modes

The wire load model mode defines the wire load mode to use for nets in subdesigns. Technology libraries have a `default_wire_load_mode` attribute that defines the default mode. You can override a mode by using the `set_wire_load_mode` command.

The cell area used to select a wire load model for a net is determined by the wire load mode. [Figure 6-2](#) shows the wire load model modes.

Figure 6-2 Wire Load Model Modes



top

Causes the wire load model and cell area for the top level to be used. Wire load models set on subdesigns and subclusters have no effect. The top mode models nets as if the design has no hierarchy; it is pessimistic.

enclosed

Causes the wire load model and the cell area of the smallest design that fully encloses the net to be used. If the design enclosing the net has no wire load model, the design hierarchy is traversed upward until a wire load model is found. The enclosed mode is more accurate than the top mode when cells in the same design are placed in a contiguous region during layout.

segmented

Causes the sum of the cell areas of the designs containing the net to be used. Nets crossing hierarchical boundaries are divided into segments (part of the net is outside the module). Each net segment is estimated from the wire load model of the design containing the segment. If the design enclosing a segment has no wire load model, the design hierarchy is traversed upward until a wire load model is found.

If you want to set the lower blocks in a design to a mode other than top mode, you must set the TOP design to enclosed or segmented mode.

Examples

To set the model `my_model` on a design, enter

```
set_wire_load_model my_model
```

If the current library does not define a default mode, PrimePower searches in the libraries in the `link_library`.

To set the mode for the current design to top so that all nets use the same model (wire load models set on subdesigns are ignored), enter

```
set_wire_load_mode top
```

In a hierarchical design, which is the current design, to cause each net to use the model of the lowest-level block that completely contains that net, enter

```
set_wire_load_mode enclosed
```

In a hierarchical design, which is the current design, to cause nets to be divided into segments (as the nets cross block boundaries) and to estimate each net segment from the wire load model of the block containing the segment, enter

```
set_wire_load_mode segmented
```

Note:

The default value for the `set_wire_load_mode` command is `top`. The `wire_load_mode` attribute cannot be removed. It can only be changed by using the `set_wire_load_mode` command.

Setting a Wire Load Model

The `set_wire_load_model` command can be used to define a wire load model for designs, hierarchical cells, and ports. The wire load model you define with `set_wire_load_model` overrides defaults.

You can use the `set_wire_load_model` command to define an external wire load model for a port. The wire load model of a port can affect the calculated wire load of the net connected to the port (depending on the wire load model for the design).

In the presence of physical hierarchy, wire loads set on subdesigns are ignored. In general, wire loads associated with clusters obey the same mode rules that govern wire loads in the presence of logical hierarchy.

If you set a wire load model on a design, automatic wire load selection is disabled for that design.

The `set_wire_load_model` command does not assume a default wire load mode. You use the `set_wire_load_mode` command to explicitly set the wire load mode before you use the `set_wire_load_model` command. (For additional information about this command, see the man page.)

The syntax is

```
set_wire_load_model -name model_name  
  
[-cluster cluster_name ]  
  
[-library library_name ]  
  
[-min] [-max] [ object_list ]
```

Note:

For more information, see the `set_wire_load_model` man page.

Examples

```
pp_shell> set_wire_load_model -name "60x60"
```

```
Using wire_load model '60x60' found in library  
'my_lib'.
```

To remove the `wire_load_model` attribute, use the `remove_wire_load_model` command. Additional information, refer to the man pages.

Setting a Wire Load Mode

The `set_wire_load_mode` command sets the wire load mode for the current design. For this command, PrimePower performs additional checking to ensure that the minimum block size is not set with wire load modes other than enclosed.

The syntax is

```
set_wire_load_mode mode_name
```

Note:

For more information, see the `set_wire_load_mode` man page.

Example

```
pp_shell> set_wire_load_mode top
```

Setting a Wire Load Minimum Block Size

The `set_wire_load_min_block_size` command sets the minimum block size for a wire. The minimum block size is supported only with the enclosed wire load mode. For this command, PrimePower performs additional checking to ensure that minimum block size is not set with other wire load modes.

The syntax is

```
set_wire_load_min_block_size size
```

size

Minimum block size. Specify a value that is greater than or equal to zero (≥ 0).

To remove the `wire_load_min_block_size` attribute, use the `remove_wire_load_min_block_size` command.

For additional information, refer to the man pages.

Setting a Wire Load Selection Group

The `set_wire_load_selection_group` command sets the wire load selection group used to determine the wire load model when `auto_wire_load_selection` is set to true.

The `set_wire_load_selection_group` command does not assume a default wire load mode. You can use the `set_wire_load_mode` command to explicitly set the wire load mode to enclosed. For detailed information, see the man page for the command.

The syntax is

```
set_wire_load_selection_group  
  
[-min] [-max]  
  
[-library lib_name ]  
  
[-cluster cluster_name ] group_name
```

Note:

For more information, see the `set_wire_load_selection_group` man page.

To remove the `wire_load_selection_group` attribute, use the `remove_wire_load_selection_group` command. For additional information, refer to the man pages.

Defining Wire Load Models or Selection Groups for Hierarchical Cells

To define distinct wire load models for hierarchical cells in the top-level design, define wire load models at the designs and at the instance level. When defining your wire load models remember the following:

- Instance level wire load settings take precedence over design level wire load settings
- PrimePower selects a wire load model based on the total cell area of the design if you designate top as the wire load mode for a design that has no wire load model defined (for the top most level) and if the first library in the link path has a wire load selection table

You can use the `set_wire_load_min_block_size` command to set a minimum block size for the design.

Selecting the Wire Load Model Automatically

If no wire load model is defined for a design, PrimePower automatically selects a default wire load model based on area. The cell area used to select a wire load model for a net is determined by the wire load mode. PrimePower searches the first library on the link path (or the first library in the design's `local_link_library`) for a default wire load.

- If the library contains a `wire_load_selection` group definition, PrimePower uses the cell area of the `current_design` to select the wire load model.
- If the library does not contain a `wire_load_selection` group definition, PrimePower uses the `default_wire_load` for the library.
- If the library contains neither a `wire_load_selection` group nor a `default_wire_load`, PrimePower does not select a wire load model and no wire load model is used.

If you set a wire load model on a design, automatic wire load selection is disabled for that design.

Note:

On large designs, using automatic wire load selection based on area is not recommended, because excessive runtimes might result.

To disable automatic wire load model selection, set the variable

```
auto_wire_load_selection = false
```

Defining the Smallest Block Size

Use the `set_wire_load_min_block_size` command to define the smallest block size to be laid out contiguously in a design. Use the `set_wire_load_min_block_size` command when the logical hierarchy is decomposed into subblocks that have no physical meaning (and should not be considered in selection of a wire load model). PrimePower selects a wire load for each design on the basis of a cell area that is no less than the value defined with the `set_wire_load_min_block_size` command.

A design with an area larger than the range defined in the `wire_load_from_area` definition is assigned the wire load of the next area range. For example, a design with area 120 lies between the defined intervals 0–100 and 150–200, so the design is assigned the higher of the two wire loads, 10x10.

A design must be mapped to determine the total area. For an unmapped design, if the `default_wire_load` attribute is not defined in the library, no wire load is used during mapping.

The wire load is automatically selected before and during a timing operation and design modification (such as with `update_timing`, `report_timing`, `compile`, `translate`, and `reoptimize_design`).

PrimePower selects a wire load model before compile or translate. If the design is not mapped, PrimePower uses either the default or no wire load model.

PrimePower resets the wire load model at the following phases:

- After structuring or mapping and before gate-level optimization
- Before buffering
- At the end of optimization

The last update of the wire load model can create a new violation. In this case, you must rerun `compile -incremental_mapping`.

When the wire load model changes, an informational message is issued in the compile log. For example,

```
Information: Changed wire load model for 'U1' from  
'(none)' to '50x50'. (OPT-170)
```

Reporting Wire Load Models

You can get information about wire load models for designs, clusters, ports, or libraries, using the commands listed in [Table 6-1](#).

Table 6-1 Commands for Reporting Wire Load Information

To report this	Use this command
The wire load models associated with either a design or a cluster	report_wire_load
The wire load model and mode for the current design and the libraries linked to the design	report_design
The wire load associated with a cluster	report_clusters
The wire load model of the output ports of a design	report_port
The wire load models defined in a specified library and wire load selection tables defined in the library	report_lib

Example

To display the selected wire load model in the current design, enter

```
pp_shell> report_design
```

```
*****
```

```
Report : design
```

```
Design : TOP
```

```
Version: v3.4
```

```
Date : Thu 2004
```

```
*****
```

Library(s) Used:

class (File: library/class.db)

Local Link Library:

{library/class.db}

Flip-Flop Types:

No flip-flop types specified.

Latch Types:

No latch types specified.

Operating Conditions:

Name	Library	Process	Temp	Volt	Interconnect	Model

WCCOM	class	1.50	70.00	4.75	worst_case_tree

Wire Loading Model:

Selected automatically from the total cell area.

Name	Library	Res	Cap	Area	Slope	Fanout	Length

10x10	class	0.0000	1.0000	0.0000	0.3110	1	0.5300

Wire Loading Model Mode: top.

Depending on the `wire_load` commands that are set, you see various messages. For example,

Selected from the default (area not fully known).

appears when design area is not known.

Selected manually by the user.

appears when you use the `set_wire_load_model` command to set the value.

Selected from the default.

appears when you do not define a wire load model, using `set_wire_load_model`, and a default is defined in the library.

Example

To display the wire load models defined in the library `my_lib`, enter

```
pp_shell> report_lib my_lib
```

```
*****
```

```
Report : library
```

```
Library: my_lib
```

```
Version: v3.4
```

```
Date : Fri 2004
```

```
*****
```

. . .

Wire Loading Model:

Name	Library	Res	Cap	Area	Slope	Fanout	Length

05x05	my_lib	100.000	1.000	0.000	0.186	1	0.390
10x10	my_lib	100.000	1.000	0.000	0.311	1	0.530
20x20	my_lib	100.000	1.000	0.000	0.547	1	0.860
30x30	my_lib	100.000	1.000	0.000	0.782	1	1.400
40x40	my_lib	100.000	1.000	0.000	1.007	1	1.900
50x50	my_lib	100.000	1.000	0.000	1.218	1	1.800
60x60	my_lib	100.000	1.000	0.000	1.391	1	1.700
70x70	my_lib	100.000	1.000	0.000	1.517	1	1.800
80x80	my_lib	100.000	1.000	0.000	1.590	1	1.800
90x90	my_lib	100.000	1.000	0.000	1.640	1	1.900

Wire Loading Model Mode: enclosed.

. . .

Wire loading model selection:

Selection Name

```
min area max area
```

```
-----
```

```
0 100 "05x05"
```

```
100 200 "10x10"
```

Specifying Port/Net Loads

The `set_load` command sets load capacitance on ports. To accurately time a design, you need to describe the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance.

For more information, see the `set_load` man page.

Example 1

To specify the external pin capacitance of ports, enter

```
pp_shell> set_load -pin_load 3.5 {IN1 OUT1 OUT2}
```

You also need to account for wire capacitance outside the port. For prelayout, specify the external wire load model and the number of external fanout points.

Example 2

For post-layout, specify the external annotated wire capacitance as wire capacitance on the port. For example, enter

```
pp_shell> set_load -wire_load 5.0 {OUT3}
```

To remove port capacitance values, use the `remove_capacitance` command.

Back-annotated Parasitics

PrimePower supports the `read_parasitics` command to back-annotate multiple formats: RSPF, SPEF, and DSPF. It is important to note, that while PrimePower parses distributed RC data, it does not in fact use or store it. For power calculation, PrimePower relies on the lumped capacitance values to determine the associated dynamic power consumption.

Example:

```
read_parasitics wire.spef.
```

For more information, view the man page.

Parasitic Reports

PrimePower provides a number of commands to report the design parasitics. They can be used to verify the back-annotation has been correctly applied.

- Wire Load Reports

Wire load reports provide information about the wire loads you have defined in your design, including

- Cell name and design name
- Wire load model for the cell

- Library that contains the wire load model
- Selection type

Wire Load Information for the Current Design

The `report_wire_load` command provides the wire load information for the current design.

Wire Load Information on Ports

The `report_port` command with its `-wire_load` option provides wire load information on ports.

Example

To see wire load information on the ports in a design, enter

```
pp_shell> report_port -wire_load
```

PrimePower displays a report similar to this:

```
*****
Report : port
        -wire_load
Design : AN2_DESIGN
*****
```

Port	External Number Points	Fanout Wire Load Model
A	0	--
B	0	--
Z	0	70x70: tech_lib

Verifying the Wire Capacitance

The `report_wire` command displays by default only the "net" capacitance and can be used to verify the values on the net with those in the back-annotation file.

The `report_wire` command provides additional options which allow the user to view the total capacitance and the toggle information on the nets as well. The `-toggle` option lists the toggles on each net.

A toggle is any logic-value change on a net and all pins connected to it. A toggle is a concept on the nets of pins.

Toggle with Z:

Toggle counts all transitions among 1, 0, and X. For transitions involving Z, the toggle is defined differently as described below:

- From any logic value changed to Z and then changed back to the original value, there is no toggle counted and there is no power consumed. For example; 1-Z-1 or X-Z-X, no toggle and no power.
- From any logic value change to Z and then changed to a new logic value, only one toggle is counted. It is the toggle from the original one to the new one. Power is calculated in the same way. For example; 1-Z-0, toggle is counted as 1-0, power is counted as 1-0. 1-Z-X, toggle is counted as 1-X, toggle is counted as 1-X, power is counted as 1-X.

Toggle Glitch:

- Glitch is a normal transition with a very narrow pulse width. For example; 0-1-0 will be counted as two toggles which is normal. But glitch power will be scaled by a factor, which is the width of the glitch over the sum of the rise and fall transition time.

Checking for Unannotated Nets

To determine whether all the nets in the design were annotated with the values from the parasitics file; the following options are available:

1. `report_annotated_parasitics -list_not_annotated`

This command can only be used if binary SPEF was used to annotate the design. If binary SPEF is used, PrimePower loads the distributed RCs and calculates the lumped capacitance.

2. Tcl procedure

If a `set_load` command file or ascii SPEF/DSPF was used to annotate the parasitics, users can write their own Tcl procedures such as the one shown below:

```
proc rpt_no_lumped_C {} {  
  
    set all_nets [get_nets * -hierarchical]  
  
    set ba_nets \  
  
        [get_nets * -hierarchical -filter \  
defined(ba_capacitance_max) ]  
  
    set not_ba_nets [remove_from_collection \  
$all_nets $ba_nets]
```

```

        set total_num_nets [sizeof_collection
$all_nets]

        set total_num_ba_nets [sizeof_collection \
$ba_nets]

        set total_num_not_ba_nets
[sized_of_collection $not_ba_nets]

        echo "Total number of nets in design =
$total_num_nets"

        echo "Number of nets with annotated load =
\ $total_num_ba_nets"

        echo "Number of nets missing annotated load
= \ $total_num_not_ba_nets"

        if {$total_num_not_ba_nets > 0} {

            report_net $not_ba_nets

        }; #end if

    }; #end proc

```

This procedure lists the nets which for which the default wire-load model value is used; because the net was not contained in the parasitics file.

7

Setting Design Constraints

Power consumption of a design is influenced by the conditions applied to the design regardless of whether PrimePower is performing dynamic or static analysis. By applying design constraints, users can define the analysis conditions to be applied for power calculation.

Design Conditions

The following sections describe the specification of the analysis conditions:

- [Operating Conditions](#)
- [Input Transition Times](#)
- [Verifying Input Transition Times](#)

- [Specifying Clock Characteristics](#)
- [Reporting Delay Back-Annotation Status](#)
- [Annotating Conditional Delays From SDF](#)
- [Multi-VDD](#)
- [Reporting Power Calculation](#)
- [Synopsys Design Constraints Formatted Script Files](#)
- [Reading an SDF File](#)

Operating Conditions

Integrated circuits exhibit different performance characteristics for different operating conditions. Listed below are the parameters which define the operating conditions:

Temperature

The chip temperature affects device delays. The temperature of the chip depends on factors such as ambient air temperature, power consumption, package type and cooling method.

Supply Voltage(s)

Higher supply voltage usually results in higher switching, internal and leakage power consumption.

Process Derating Factor

The process derating factor is used to scale device parameters resulting from variation in the fabrication process.

Interconnect Model Type

This parameter defines an RC tree topology used to estimate net capacitance. While associated with the operating condition, the net capacitance values can be overwritten with parasitics files annotation.

NLDM/NLPM (non-linear delay model/non-linear power model) libraries are generally characterized for one operating condition. In theory, customers can generate a library for multiple operating conditions which rely on k-factors to modify the power table values based on the operating conditions; but this method is rarely applied. Most vendors provide separate tables for each operating condition.

With *SPPM/SPDM* (scalable polynomial delay model/scalable polynomial power model) however, polynomials are used to define the effects of the operating conditions. A single library can define multiple operating conditions, since the voltage and temperature values are variables in the equation. Additionally, different equations can be used for different process points; which eliminates the need for the process derating factor.

Setting Operating Conditions

The `set_operating_conditions` command sets process, temperature, and voltage conditions for power analysis.

The operating conditions you specify must be defined in a specified library or a library in the link path. Use the `report_lib` command to get a list of the available operating conditions in a technology library before you use the `set_operating_conditions` command.

If no `default_operating_condition` is defined in the libraries, PrimePower applies the values assigned to the nominal temperature and voltage specifications.

Depending on its reference library, the library cells use the nominal values specified in the library to which it belongs. If cell A from library ALIB has a nominal voltage of 1.2, and cell B from library BLIB has a nominal voltage of 1.5, they use different voltages.

PrimePower uses the value of the variable:
`set_default_oc_per_lib` to determine what to use.

By default, this variable is true; which means each cell uses the default from the library to which it belongs. If this is not the desired behavior, you can set this variable to `false`, in which case the operating conditions from the main library applies to all cells.

For more information, see the `set_operating_conditions` man page.

Example 1

To set WCCOM from the `tech_lib` library, enter

```
pp_shell> set_operating_conditions WCCOM -library tech_lib
```

If you do not specify a library, PrimePower searches all libraries in the link path. After you set the operating conditions, you can report or remove operating conditions.

With SPPM libraries, operating conditions can be specified for the entire design or for a particular object list via the `-object_list` option.

Operating Condition Information

These commands report or remove information.

To do this	Use this command
List the operating condition settings for the design.	<code>report_design</code>
Remove operating conditions from the current design.	<code>remove_operating_conditions</code>

Input Transition Times

You can define the transition time at each input port. PrimePower uses this information to calculate the cell power consumption and transition times for the following logic stages.

The `set_input_transition` command defines a fixed transition time for input ports. PrimePower uses the specified transition time in calculating the power of logic driven by the port.

For example; `set_input_transition 0.1 [all_inputs]`

By default, the input transition time is assumed to be 0.

Note:

Assignments to ports, other than inputs, will be flagged as errors.

Verifying Input Transition Times

Reporting the input ramp times can be done with a PrimePower command or by accessing the appropriate attribute information.

Command

```
report_ports -drive
```

Attributes

```
pp_shell> get_attribute[get_ports <portname>]  
input_transition_rise_max <value>
```

```
pp_shell> get_attribute[get_ports <portname>]  
input_transition_fall_max <value>
```

Note:

PrimePower, by default, uses the "max" `input_transition` rise and fall attributes for ramp propagation. You can change this by setting the `pwr_calc_use_slew` variable to min or ave.

Specifying Clock Characteristics

PrimePower uses clocks for setting default switching activity for primary inputs and black-box outputs, propagating switching activity, constructing average cycle waveform, and reporting clock tree power.

You need to specify all clocks used in the design. The clock information includes:

- Period and waveform
- Latency (insertion delay)
- Uncertainty (skew)
- Divided and internally generated clocks
- Clock-gating checks

- Fixed transition time for incomplete clock networks

PrimePower supports analysis of the synchronous portion of a design. PrimePower analyzes paths between registers or ports. For a design with multiple interacting clocks, PrimePower determines phase relationship between the startpoint and endpoint clocks. The clocks can be single phase, multiple phase, or multiple frequency clocks.

Creating a Clock

The `create_clock` command creates a clock at the specified source. A source can be defined at an input port of the design or an internal pin. PrimePower traces the clock network automatically so the clock reaches all registers in the transitive fanout of its source.

A clock you create with the `create_clock` command has an ideal waveform that ignores the delay effects of the clock network. After you create the clock, you must describe the clock network to perform accurate timing analysis.

The `create_clock` command creates a path group having the same name as the clock. This group contains all paths ending at points clocked by this clock.

Example

To create a clock on ports C1 and CK2 with a period of 10, a rising edge at 2, and a falling edge at 4, enter

```
pp_shell> create_clock -period 10 -waveform {2 4} {C1 CK2}
```

The resulting clock is named C1 to correspond with the first source listed. C1 produces the waveform shown in [Figure 7-1](#).

Figure 7-1 C1 Clock Waveform



Starting with the 2002.03 release, PrimePower supports the analysis of multiple clocks propagated to a single register. You can define multiple clocks on the same port or pin by using the `-add` option of the `create_clock` command.

For more information, see the `create_clock` man page.

Automatically Identifying Clock Sources

The `derive_clocks` command automatically identifies clock sources in the design and creates a clock for each source. This is useful for understanding a netlist that has incomplete assertions.

Example

To create clocks with a period of 8 for all sources in the design, enter

```
pp_shell> derive_clocks -period 8 -waveform {2 5}
```

Creating a Virtual Clock

You can use the `create_clock` command to define virtual clocks for signals that interface to external (off-chip) clocked devices. A virtual clock has no actual source in the current design, but you can use it for setting input or output delays. You can also use the virtual clock to specify what frequency your combinational circuit works on.

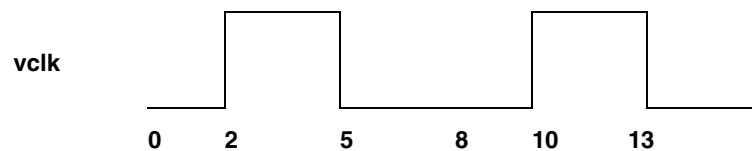
Example

To create a virtual clock named `vclk`, enter

```
pp_shell> create_clock -period 8 -name vclk -waveform {2 5}
```

The `vclk` clock has the waveform shown in [Figure 7-2](#).

Figure 7-2 vclk Clock Waveform



Selecting Clock Objects

The `get_clocks` command selects clocks for a command to use, for example, to ensure that a command works on the CLK clock and not on the CLK port.

For more information, see the `get_clocks` man page.

Example

To report the attributes of clocks having names starting with `PHI1` and a period less than or equal to 5.0, enter

```
pp_shell> report_clock [get_clocks -filter \  
    "period <= 5.0" PHI1*]
```

Applying Commands to All Clocks

The `all_clocks` command is equivalent to the PrimePower `get_clocks *` command. The `all_clocks` command returns a token representing a collection of clock objects. The actual clock names are not printed.

Example

To disable time borrowing for all clocks, enter

```
pp_shell> set_max_time_borrow 0 [all_clocks]
```

Removing Clock Objects

The `remove_clock` command removes a list of clock objects or clocks in the design.

For more information, see the `remove_clock` man page.

Note:

The `reset_design` command removes clocks as well as other information.

Example 1

To remove all clocks whose names start with CLKB, enter

```
pp_shell> remove_clock [get_clocks CLKB*]
```

Example 2

To remove all clocks in the design, enter

```
pp_shell> remove_clock -all
```

Synopsys Design Constraints Formatted Script Files

Synopsys Design Constraints (SDC) formatted script files are Tcl scripts that use a subset of the commands supported by PrimeTime and Design Compiler. There are limitations placed on some of the supported commands. For example, not all options are supported in some cases. Although SDC is a subset of Tcl, it is not a full-function scripting language. It is intended to support the specification of timing constraints across various EDA tools. It is not intended for complex scripting.

Within SDC, there is only one design: the current design. You cannot change the current design with SDC.

SDC files are read into PrimePower with the `read_sdc` command, and are written from PrimePower using the `write_sdc` command. For more information about these commands, see the man pages.

Checking the Syntax of the SDC File

You can check the SDC file and verify that it is syntactically and semantically correct by using the `read_sdc -syntax_only` command. This will validate the script without having any effect on the design.

For more information about SDC, see the *Using the Synopsys Design Constraints Format Application Note*. You can find this Application Note in SOLD and in SolvNet.

Reading an SDF File

The `read_sdf` command reads instance-specific pin-to-pin leaf cell and net timing information from an SDF version 1.0, 1.1, 2.0, 2.1, or 3.0 file, and uses the information to annotate the current design.

Instance names in the design must match instance names in the timing file. For example, if the timing file was created from a design using VHDL naming conventions, the design you specify must use VHDL naming conventions.

Annotating Timing From a Subdesign Timing File

When you specify the `-path` option, the `read_sdf` command annotates the current design with information from a timing file created from a subdesign of the current design.

When you specify a subdesign, you cannot use the net delays to the ports of the subdesign to annotate the current design.

Annotating Load Delay

The load delay, also known as extra source gate delay is the portion of the cell delay caused by the capacitive load of the driven net. Some delay calculators consider the load delay part of the net delay; other delay calculators consider the load delay part of the cell delay. By default, the `read_sdf` command assumes the load delay is included in the cell delay in the timing file being read. If your timing file includes the load delay in the net delay instead of in the cell delay, use the `-load_delay` option with the `read_sdf` command.

Annotating Timing Checks

When the timing file contains the following checks, they are used to annotate the current design.

For this check	The SDF construct is
setup and hold	SETUP, HOLD, and SETUPHOLD
recovery	RECOVERY
removal	REMOVAL
minimum pulse width	WIDTH
minimum period	PERIOD
maximum skew	SKEW
no change	NOCHANGE

Reading the File

The `read_sdf` command reads an SDF file. After reading an SDF file, PrimePower reports the following:

- Number of errors found while reading the SDF file (for example, pins not found in the design)
- Number of annotated delays and timing checks
- Unsupported SDF constructs found in the SDF file, with the number of occurrences of each SDF construct
- Process, temperature, and voltage values found in the SDF file
- Annotated delays and timing checks of the design (with the `report_annotated_delay` and `report_annotated_check` commands)

For more information, see the `read_sdf` man page.

Example 1

This command reads from disk the SDF format file `adder.sdf`, which contains load delays included in the cell delays and uses its information to annotate the timing on the current design.

```
pp_shell> read_sdf -load_delay cell adder.sdf
```

Example 2

These commands read the timing information of instance `u1` of design `MULT16` from the disk file `mult16_u1.sdf`, and annotates the timing on the design `MY_DESIGN`. The load delay is included in the net delays.

```
pp_shell> current_design MY_DESIGN  
pp_shell> read_sdf -load_delay net -path u1 mult16_u1.sdf
```

Example 3

This command reads timing information and annotates the current design with the worst timing when the timing file has different timing conditions for the same pin pair. The load delay is assumed to be included in the cell delay.

```
pp_shell> read_sdf -cond_use_max boo.sdf
```

Example 4

This command reads minimum and maximum timing information and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delay, PrimePower uses delays annotated for the minimum condition. When reporting maximum delays, PrimePower uses delays annotated for the maximum condition.

```
pp_shell> read_sdf -analysis_type bc_wc boo.sdf
```


Example 5

This command reads minimum and maximum timing information from two separate SDF files and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delays, PrimePower uses delays annotated for the minimum condition. When reporting maximum delays, PrimePower uses delays annotated for the maximum condition.

```
pp_shell> read_sdf -analysis_type bc_wc \  
           -min_file boo_bc.sdf -max_file boo_wc.sdf
```

Removing Annotated Timing Checks and Delays

To remove timing checks annotated with the `read_sdf` command, use the `reset_design` or `remove_annotated_check` command.

To remove delays read and annotated with the `read_sdf` command, use the `reset_design` or `remove_annotated_delay` command.

Reporting Delay Back-Annotation Status

Because SDF files are usually large (about 100 MB or larger) and contain many delays, it is a good idea to verify that all nets are back-annotated with delays (and timing checks, where applicable).

Reporting Annotated or Nonannotated Delays

You can check and identify nets that have not been back-annotated with delays in SDF. PrimePower checks and reports on cell delays and net delays independently. When reporting net delays, PrimePower considers three types of nets:

- Nets connected to primary input ports
- Nets connected to primary output ports
- Internal nets that are not connected to primary ports

This distinction is important because according to the SDF standard, only internal nets are annotated in SDF.

The `report_annotated_delay` command reports the number of annotated and nonannotated delay arcs.

For more information, see the `report_annotated_delay` man page.

Example

To produce a report of annotated delays, enter

```
pp_shell> report_annotated_delay
```

PrimePower displays a report similar to this:

Delay type	Total	Annotated	NOT Annotated
-----+-----+-----+-----+			
cell arcs	16	14	2
cell arcs (unconnected)	5	5	0
internal net arcs	3	3	0
net arcs from primary inputs	11	11	0
net arcs to primary outputs	4	4	0
-----+-----+-----+-----+			
	39	37	2

Reporting Annotated or Nonannotated Timing Checks

You can create a report showing the number of annotated timing checks of all cell timing arcs within your design.

For more information, see the `report_annotated_check` man page.

Example

To produce an annotated timing check report, enter

```
pp_shell> report_annotated_check
```

PrimePower displays a report similar to this:

```
*****
report: annotated_check
Design: my_design
*****
```

	Total	NOT Annotated	Annotated
cell setup arcs	2368	2368	0
cell hold arcs	2368	2368	0
cell recovery arcs	676	676	0
cell removal arcs	0	0	0
cell min pulse width arcs	135	135	0
cell min period arcs	822	822	0
cell max skew arcs	716	716	0
cell nochange arcs	0	0	0
	7085	7085	0

Annotating Conditional Delays From SDF

Delays and timing checks specified in an SDF file can be conditional. The SDF condition is usually an expression based on the value of some inputs of the annotated cell. The way PrimePower annotates these conditional delays depends on whether the Synopsys library specifies conditional delays.

- If the Synopsys library contains conditional arcs, all conditional delays specified in SDF file are annotated. The condition string specified in the Synopsys library with the `sdf_cond` construct must exactly match the condition string in the SDF file.
- If the Synopsys library does not contain conditional arcs (there is no `sdf_cond` construct in the Synopsys library), the maximum or minimum delays of all conditional delays from SDF are

annotated. To specify whether to annotate the minimum or maximum delays, use the `-cond_use_max` or `-cond_use_min` option of the `read_sdf` command.

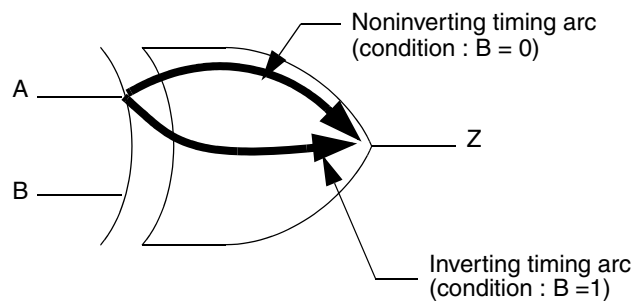
If your library contains state-dependent delays, using a Synopsys library containing conditional arcs enables more accurate annotation from the SDF.

PrimePower uses the condition specified in the SDF only to annotate the delays and timing checks to the appropriate timing arc specified in the Synopsys library. If you have conditional timing arcs in your SDF, and your library is defined correctly to support conditional arcs, you can use case analysis (or constant propagation) to enable the desired conditional arc values.

Example

Consider the example 2-input XOR gate in [Figure 7-3](#).

Figure 7-3 Example of State-Dependent Timing Arcs

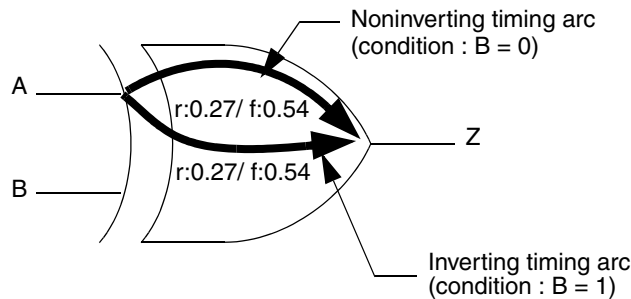


The SDF for the delay from A to Z looks like this:

```
(INSTANCE U1)
(DELAY
  (ABSOLUTE
    (COND B (IOPATH A Z (0.21) (0.54) ) )
    (COND ~B (IOPATH A Z (0.27) (0.34) ) )
  )
)
```

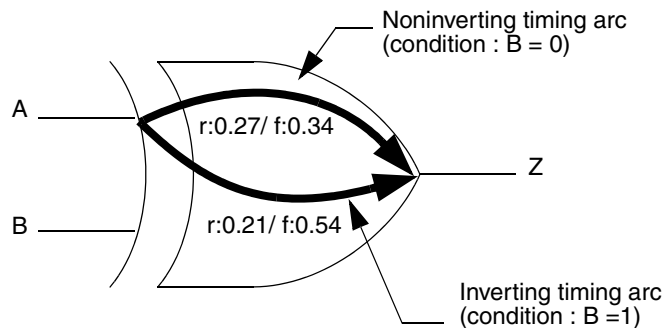
If the Synopsys library contains no conditions, the annotation from SDF uses the worst-case delay for all timing arcs from A to Z. Mapping from the library the timing arc that corresponds to a given condition is not possible. In this case, the annotated delays are as shown in [Figure 7-4](#).

Figure 7-4 Annotated Delays When the Synopsys Library Contains No Conditions



If the Synopsys library contains conditions, the annotation can identify which timing arc corresponds to the SDF condition. In this case, the annotated delays are as shown in [Figure 7-5](#).

Figure 7-5 Annotated Delays When the Synopsys Library Contains Conditions



Note:

An IOPATH statement in the SDF file annotates all arcs between two pins. If an IOPATH statement follows a COND IOPATH statement, it overwrites the former statement.

The IOPATH delay for the A to Z arc varies depending on whether the B input is a 1 or 0. To select the B = 0 delays, set a case analysis on the B input pin. For example, enter

```
pp_shell> set_case_analysis 0 [get_pins U1/B]
```

Note:

It is possible for a constant to propagate to a pin that selects a conditional arc. This can occur through normal logic constant propagation from a tie-high or tie-low condition, or through a `set_case_analysis` that was set on another pin that propagates a constant to the pin.

Multi-VDD

When calculating power for designs with multiple voltage rails, PrimePower checks the value of the attribute `output_signal_level` to calculate the switching power of the net.

PrimePower supports the ability to set the values of single and multi-rail cells. Users can modify the default rail voltages of single-rail or multi-rail cells. The values specified are used to calculate the correct energy.

The `set_rail_voltage` command for a particular cell, causes PrimePower to override the default voltage. This is particularly useful for what-if analysis. Another application for the `set_rail_voltage` is to apply operating conditions to hierarchical blocks, and then back-annotate rail voltages due to IR drop on individual cells to determine the effects.

Reporting Power Calculation

For users interested in understanding how the power was calculated, PrimePower supports the `report_power_calculation` command. Shown below is the syntax:

```
report_power_calculation pin_cell_or_net_list
    [-state_condition boolean_eq_of_pins]
    [-path_source pin_name]
    [-rise | -fall]
    [-negative_unateness]
    [-verbose]
    [-nosplit]
```

With this command, users can verify the power data in the technology libraries or for debugging the sources of power consumption at the leaf-cell level. It can be applied to both statistical activity based or event-based power analysis, but the output differs slightly in that the statistical activity flow includes the toggle rate and signal probability; whereas, the output from the VCD flow is based on single events.

Note:

In order to view the details of the power calculation, the Synopsys libraries must be compiled with the `library_features` attribute (`report_power_calculation`). Without this library attribute, the output of the `report_power_calculation` command will not provide the power table indexes used to calculate the power.

Leakage Power Calculation

When the user selects the power calculation of an instance without the `-verbose` option, only the leakage power calculation is reported.

Example:

```
report_power_calculation [get_cell cm01]
```

```
-----  
Cell Leakage Power Calculation
```

```
cell: cm01 (iv22)  
state condition: !a
```

```
State Dependent Leakage Power = 1.440e-10 W  
cell leakage power = leakage power value * state probability
```

```
Leakage Power Value = 0.0002750  
library model: scalar  
value = 0.0002750  
(no scaling since the library has no leakage power k-factors)
```

```
State Probability = 0.5237      (user annotated)
```

```
=====
```

```
Cell Leakage Power Calculation
```

```
cell: cm01 (iv22)  
state condition: a
```

```
State Dependent Leakage Power = 8.811e-11 W  
cell leakage power = leakage power value * state probability
```

```
Leakage Power Value = 0.0001850  
library model: scalar  
value = 0.0001850  
(no scaling since the library has no leakage power k-factors)
```

```
State Probability = 0.4763      (user annotated)
```

```
=====
```

Cell Leakage Power Calculation

```
cell: cm01 (iv22)
state condition: default
```

```
Leakage Power = 0.000      W
cell leakage power = leakage power value * state probability
```

```
Leakage Power Value = 0.0004853
library model: scalar
value = 0.0004853
(no scaling since the library has no leakage power k-factors)
```

```
State Probability = 0.000      (estimated)
```

To isolate the power consumption for particular states, the user can include the `-state` option.

Internal Power

Shown below is a sample output for viewing the internal power consumption of an inverter:

```
report_power_calculation
```

Pin Internal Power Calculation

```
cell: cm01 (iv22)
pin: y
path source: a
state condition: default
```

```
Path Dependent Rise Pin Internal Power = 1.825e-06 W
pin internal power = internal energy * pin toggle rate
```

```
Rise internal energy per transition = 0.1916
library model: NLPM
table variables:
  X = input_net_transition = 0.000
  Y = total_output_net_capacitance = 5.012
```

```

relevant portion of lookup table:
              (X)  0.4000      (X)  1.2000
(Y)  0.4100      (Z)  0.1227      (Z)  0.1920
(Y)  0.7400      (Z)  0.1188      (Z)  0.1653

```

```

Z = A + B*X + C*Y + D*X*Y
A =  0.0788      B =  0.1220
C =  0.0225      D = -0.0862

```

```

Z = 0.191621
(no scaling since the library has no internal power k-factors)

```

```

Path Dependent Rise Pin Toggle Rate = 0.009522 (user annotated)

```

```

Path Dependent Fall Pin Internal Power = 7.722e-07 W
pin internal power = internal energy * pin toggle rate

```

```

Fall internal energy per transition = 0.07372
library model: NLPM
table variables:
  X = input_net_transition = 0.000
  Y = total_output_net_capacitance = 5.012
relevant portion of lookup table:
              (X)  0.4000      (X)  1.2000
(Y)  0.4100      (Z)  0.0122      (Z)  0.0729
(Y)  0.7400      (Z)  0.0086      (Z)  0.0489

```

```

Z = A + B*X + C*Y + D*X*Y
Z = 0.0737192
(no scaling since the library has no internal power k-factors)

```

```

Path Dependent Fall Pin Toggle Rate = 0.01047 (user annotated)
-----

```

Again, users can isolate the power information by specifying the state or the transition (rising or falling).

Net Switching Power

To view the switching power calculation the user can specify the desired net to view. The command: `report_power_calculation out1` produces the following report:

```
report_power_calculation out1
```

Net Switching Power Calculation

net: out1
driver: cm01/y

Switching power = 0.0002425 W
net switching power = switching energy * net toggle rate

Switching Energy Per Transition = 12.13
switching energy = $0.5 * \text{capacitance} * \text{voltage}^2$
total net capacitance = 5.012
voltage = 2.200

Net Toggle Rate = 0.02000 (user annotated)

Note: If the `set_current_rail` command is applied, only the power calculation for the specified rail will be output.

8

Statistical Activity Based Power Analysis

PrimePower is capable of performing statistical activity based power analysis based on toggle rates to quickly calculate the average power of the design. The annotated activity can come from default toggle rates, switching activity specifications, or in the form of simulation generated SAIF files. Zero-delay simulation is used to propagate switching activity to unannotated nets to calculate the power consumption for the complete design. This chapter describes the input activity options for statistical activity based power analysis and how PrimePower uses the information to calculate the average power and cycle-based power waveforms.

Overview

This chapter contains the following sections:

- [Statistical Activity Based Analysis](#)

- [Switching Activity](#)
- [Vector-free/Default Activity](#)
- [Switching Activity](#)
- [Toggle Rate](#)
- [Annotating Switching Activity With set_switching_activity](#)
- [Annotating Switching Activity Using SAIF Files](#)
- [Estimating Unannotated Switching Activity](#)
- [Correlation](#)
- [Complex Cells](#)
- [Debugging the Annotation](#)
- [Case Analysis](#)
- [Power Calculation](#)

Statistical Activity Based Analysis

Statistical activity based power analysis in PrimePower is performed in multiple stages.

1. PrimePower checks the switching activity on all the nets in the design. The switching activity can be obtained from simulation or user annotation using the SAIF and switching activity annotation commands described in the following sections.

2. PrimePower then propagates the switching activity on any unannotated nets, and assigns default annotation where applicable until toggle rates are determined for every net. This propagation is done via internal zero-delay simulation when the user issues the `calculate_power` command.
3. Based on the activity on the outputs and inputs of every leaf cell, PrimePower uses heuristics to calculate the Boolean difference for each power arc to determine the contribution of each power arc. The power arcs are obtained from the library power table. Ultimately, the contribution for each arc is calculated to determine the total power for each cell.

User options control each stage of the analysis and are described throughout the chapter.

Switching Activity

The power of a design depends on the switching activity of the design nets and cell pins. The required switching activity information can be annotated on design objects like nets, ports, pins and cells, and then used by the `calculate_power` command during power analysis. We first describe the types of switching activity that can be annotated to design objects, and in the next sections we look into the commands that can be used for switching activity annotation.

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 probability that the value of the design object has logic value 1. The toggle rate is the rate at which the design object switches between logic values 0 and 1.

- State-dependent (SD) toggle rates on input pins of leaf cells. 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/or path-dependent (SDPD) toggle rates on output pins of leaf cells. The internal power characterization of output pins can be state-dependent and/or path-dependent. Output pins of cells with SDPD characterization can be annotated with SDPD toggle rates.
- State-dependent static probability on leaf cells. Cell leakage power can be characterized using state-dependent leakage power tables. Such cells can be annotated with state-dependent static probability.

Switching activity can be annotated by:

- Using the `set_switching_activity` command on individual design objects.
- Generating one or more Switching Activity Interchange Format (SAIF) files using RTL or gate-level simulation, and reading the SAIF files using the `read_saif` or `merge_saif` commands.
- Relying on default switching activity.

Switching activity annotation can be analyzed using the `report_saif` command and removed using the `reset_switching_activity` command. These commands are described in later sections.

Vector-free/Default Activity

If no simulation data is available, users can rely on the built-in default toggle rate to analyze the power consumption.

The default is specified with the variables:

```
pwr_default_toggle_rate (default 0.5)
pwr_default_static_probability (default 0.5)
pwr_default_tr_reference_clock [fastest|related] \
(default fastest)
```

This toggle rate is assigned to all primary inputs, tri-state output, black-box output, and propagated throughout the rest of the design.

Toggle Rate

The toggle rate of the primary inputs and other starting points such as black-box outputs are assigned values which correspond to $1/2$ the clock frequency * `pwr_default_toggle_rate` variable value (0.5).

No Clock Specification

If no clocks have been defined, the default clock period is assumed to be the default library time unit (generally 1 ns). Therefore, it is strongly recommended that users specify the frequency of operation by defining the clocks with the `create_clock` command. If no clock exists, users can define virtual clocks with this command as well. See the section [“Specifying Clock Characteristics” in Chapter 7](#).

With Clock Specification

By default, the variable `pwr_default_tr_reference_clock` is set to `related`. In this case, PrimePower locates the related clocks for these ports and assigns a toggle rate relative to the appropriate clock. For example

- Primary input D1 feeds a flip-flop U1 clocked by `clk1` (period 10 / $tr = 0.2$)
- Primary input D2 feeds a flip-flop U2 clocked by `clk2` (period 20 / $tr = 0.1$)

The switching activity for D1 and D2 will vary since the related clocks differ. The clock-based $toggle\_rate = (1/2 * \text{clock toggle rate} * pwr\_default\_toggle\_rate)$. Assuming the default value is assigned to `pwr_default_toggle_rate`: 0.5 then

- The toggle rate for D1 = $(0.5 * 0.2 * 0.5) = 0.05$
- The toggle rate for D2 = $(0.5 * 0.1 * 0.5) = 0.025$

If the `pwr_default_toggle_rate` is set to 0.1:

- The toggle rate for D1 = $(0.5 * 0.2 * 0.1) = 0.01$
- The toggle rate for D2 = $(0.5 * 0.1 * 0.1) = 0.005$

To determine the related clock, PrimePower checks the SDC constraints. If input transitions are provided relative to a clock; then it will assume the specified clock is the related clock. Otherwise, PrimePower will trace the fanout of the port to a point where the clock is bound. If more than one clock is found, the fastest of the related clocks will be used.

If no related clock is found; the fastest design clock will be used to determine the switching activity. Users can also set the variable `pwr_default_tr_reference_clock` to fastest; in which case the fastest design clock will be used to determine the toggle rate of the starting point ports/pins.

In this case, both D1 and D2 would be assigned the same default toggle rate based on the fastest clock: `clk1`

- The toggle rate for D1 = $0.5 * 0.2 * 0.5 = 0.05$
- The toggle rate for D2 = $0.5 * 0.2 * 0.5 = 0.05$

The following attribute, `related_clock <related_clock>`, will be placed on the appropriate nets/ports. In the above example, the value of `related_clock` on D1 would be `clk1`; and for D2 would be `clk2`.

Static Probability

The static probability defines the percentage of time the signal is at a high state. By default, the value is 0.5 which means the signal is high for half the time and low for half the time.

The default values may be appropriate for data bus lines, but may not be appropriate for some signals such as `reset` or `test_enable` signals.

Annotating Switching Activity With `set_switching_activity`

The more accurate the switching activity, the more accurate the power analysis. Wherever possible, users without simulation-generated toggle-rate information should provide switching activity of the primary inputs which mimics the true behavior.

The `set_switching_activity` command can be used to annotate design objects with the different types of switching activity.

Simple switching activity can be annotated using the `-static_probability`, `-toggle_rate` and `-period` arguments. For example:

```
set_switching_activity [get_net net1] \  
-static_probability 0.2 -toggle_rate 10 -period 1000
```

This specifies that the value of the net `net1` is logic 1 for 20% 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 target library time unit. The `-period` argument is optional and a default value of 1 is used when it is not specified.

State dependent toggle rates can be annotated using the `-state_dep` argument, as shown in this example:

```
set_switching_activity [get_pin ff1/Q] -toggle_rate 0.01 -  
state_dep "D"  
set_switching_activity [get_pin ff1/Q] -toggle_rate 0.03 -  
state_dep "! D"
```

This 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. The `-rise_ratio` argument can also be used with state dependent toggle rates to specify the ratio of rise transitions to fall transitions for the specified state. For example:

```
set_switching_activity [get_pin xor1/Y] -toggle_rate 0.01 -  
state_dep \  
"A" -rise_ratio 0.9
```

This specifies that the pin xor1/Y toggles 0.01 times when the cell is in state "A", and that 90% of these toggles are rise toggles.

Path dependent toggle rates can be annotated using the `-path_dep` argument. The following:

```
set_switching_activity [get_pin and1/Y] -toggle_rate 0.02 -  
path_dep "A"  
set_switching_activity [get_pin and1/Y] -toggle_rate 0.00 -  
path_dep "B"
```

Specifies that the pin and1/Y toggles 0.02 times due to a toggle on the input pin A, but never toggles due to a toggle on B. Toggle rates that are both state and path dependent can be specified using the `-state_dep` and `-path_dep` options together.

State-dependent static probabilities can be annotated using the `-state_dep` argument. The following:

```
set_switching_activity [get_cell and1] -static 0.1 -  
state_dep "A & B"  
set_switching_activity [get_cell and1] -static 0.7 -  
state_dep "A & !B"  
set_switching_activity [get_cell and1] -static 0.2 -  
state_dep "! A"
```

Specifies that the cell and1 is at state "A & B" for 10% of the time, at state "A & !B" for 70% of the time, and at state "!A" for 20% of the time.

To assign toggle rates which are based on the related clocks, users can apply the `-related_clocks` option. The following example

```
set_switching_activity -select regs -toggle_rate .1 \  
-related_clocks {clk1 clk2}
```

specifies that all registers in the design have a toggle rate of 0.1 relative to their related clock.

To assign scale factors, the `-tr_scale_factor` option can be used. The following example

```
set_switching_activity -select cgs -related_clocks \  
{clk1 clk2} -tr_scale_factor 0.4
```

specifies the output of the clock-gating cells should toggle at 0.4 times the frequency of their related clocks.

Note:

When setting the toggle rate on registers and clock-gating cells, users should first apply the command to the registers, followed by the clock-gating cells since these are in fact registers as well.

More details on the `set_switching_activity` command can be found in the following syntax description.

```
set_switching_activity  
    [-static_probability sp_value]  
    [-toggle_rate tr_value]  
    [-period period_value]  
    [-clock clock_name]  
    [-state_dep state_condition]  
    [-path_dep path_sources]
```

```
[-rise_ratio ratio_value]
[-select select_types]
[-hier]
[-instances instances]
[object_list]
[-verbose]
[-tr_scale_factor factor]
[-related_clocks clocks]
```

`-static_probability sp_value`

Specifies the value of the static_probability switching activity. `sp_value` represents the proportion of the time the signal is at the logic state 1. For example, an `sp_value` of 0.25 indicates that the signal is in the logic state 1 for 25% of the time.

`-toggle_rate tr_value`

Specifies the value of the toggle rate switching activity. `tr_value` is a floating point number that represents the number of 0->1 and 1->0 transitions that the signal makes during a period of time. The period can be specified with the `-period` option. Alternatively, a related clock can be annotated using the `-clock` argument, and the specified toggle rate will be relative to the related clock period.

`-tr_scale_factor factor`

This option is an alternative to specifying the toggle rate. Instead, with the

`-clock` or `-related_clock` option, users can specify a scale factor. For example, if the clock toggle rate is 0.1; then specifying a `tr_scale_factor` on the registers of 0.1 would cause them to have a toggle rate of 0.001.

`-state_dep state_condition`

Specifies the state condition when annotating state-dependent toggle rates on pins or state-dependent static probabilities on cells. State dependent toggle rates can be annotated when the

internal power of the library cell pin is characterized with state-dependent power tables. State-dependent static probabilities can be annotated when the cell leakage power is characterized with state-dependent power tables. The state condition specified with this argument must be logically equivalent to a state condition in the internal / leakage power characterization.

`-path_dep path_sources`

Specifies the path sources when annotating path-dependent toggle rates on pins. This is used when the library cell pin has path-dependent internal power. The path source(s) specified with this argument must be the same as those in the internal power characterization.

`-rise_ratio ratio_value`

Specifies the ratio of rise transitions to total transitions for the specified toggle rate when annotating pins that are characterized with both rise and fall internal power. The `ratio_value` argument is a floating point number between 0.0 (all transitions are falling) and 1.0 (all transitions are rising). You need to specify a toggle rate in order to use this option. The default value is 0.5.

`-period period_value`

Specifies the time period for which the number of transitions given in the toggle rate `tr_value` occur; The time units for this value are those specified in the target technology library. When this argument is used, the `tr_value` specified by the `-toggle_rate` argument is divided by the given `period_value`; The resulting toggle rate is then annotated to the objects given to the `set_switching_activity` command. If the `-period` argument is not specified, a default `period_value` of 1.0 is assumed.

`-clock clock_name`

Specifies a clock to which `tr_value` is related. This clock is referred to as the object's related clock. When specified, the related clock is annotated to the design object(s) and during power calculations, the annotated toggle rate value is divided related clock period. The resulting toggle rate is then used for calculating power. The clock period can be changed between different runs of `report_power` and the resulting change in the toggle rate of the design objects will be used automatically without re-annotating the switching activity.

`-related_clock clocks`

When applied, the switching activity options will be applied to the selected items whose related clock is included in the list.

`-select select_types`

Specifies a list of object types that will be used for implicitly selecting the objects that will be annotated with the specified switching activity. The `select_types` argument can be a list of one or more of the following object types: `cgs` (clock-gating cells), `black_boxes` (black_box outputs), `regs` (sequential cell outputs), `tris` (tristate cell outputs), `inputs` (input design ports / hierarchical instance pins), `outputs` (output design ports / hierarchical instance pins), `inout` (inout design ports / hierarchical pins) and `nets` (nets). When the `-select` argument is used all the objects in the current instance (current design, if no current instance is set) that satisfy the selection criteria will be annotated with the specified switching activity. If the `regs` selection type is used, both the non-inverting (Q) and inverting (QN) sequential cell outputs are annotated. The annotated toggle rate on the inverting outputs is the same as that on the non-inverting outputs. The static probability on the inverting outputs is $1.0 - sp\_val$ where `sp_val` is the specified static probability.

`-hier`

Used with the `-select` argument and specifies that all the objects in all the hierarchy that satisfy the selection criteria will be annotated. If not specified only the top level objects in the current instance will be considered.

`-instances instances`

Used with the `-select` argument and specifies that all the objects in the given list of instances that satisfy the selection criteria will be annotated. This option can also be used with the `-hier` option.

`object_list`

Specifies a list of nets, pins, ports or cells in the current design on which the `static_probability` and `toggle_rate` switching activity values are to be set. Note that the `object_list` argument for explicitly specifying the objects to be annotated, and the `-select` argument for implicitly specifying the objects to be annotated, are mutually exclusive.

`-verbose`

Provides a more verbose output when switching activity is annotated.

Annotating Switching Activity Using SAIF Files

Better yet, is to provide the switching activity generated from simulation. Users have the option of providing either RTL level or gate-level SAIF files to PrimePower for analysis.

Generating SAIF

You can generate SAIF files from an HDL simulation or use the UNIX `vcd2saif` utility provided with PrimePower.

Using HDL Simulation

The Synopsys VCS-MX simulator is capable of generating SAIF directly with built-in tasks. For other HDL simulators, the SAIF generation PLI must be linked and the testbench must include the appropriate PLI commands for outputting the toggle information in the SAIF format.

Refer to the *Power Compiler User Guide* for more information on SAIF generation from HDL simulation.

Using the vcd2saif Utility

Not all simulators read SAIF files. However, because many simulators support the VCD format, you can use the `vcd2saif` utility to convert both RTL and gate-level VCD data into the SAIF format. This utility supports Verilog, VHDL, and mixed language VCD outputs from VCS, VCS-MX, Verilog-XL, MTI, and NC-SIM. The `vcd2saif` utility also supports the VHDL construct "generate". The supported formats for the input VCD file into `vcd2saif` can be compressed into `.gz`, `.Z`, `.vpd` and `.fsdb` formats.

This utility can be used with or without a forward SAIF file.

The `vcd2saif` utility does not support SDPD.

The following syntax for the vcd2saif utility can be used for RTL simulation (without a forward SAIF file) and gate-level simulation:

```
vcd2saif -i vcd_file -o bsaif  
[-instance path ...]  
[-format lang] [-testbench lang]  
[-verilog_instance path] [-vhdl_instance path]  
[-no_div] [-keep_leading_backslash] [-time]
```

Execute `vcd2saif -h` for the complete list of options.

RTL SAIF

If no gate-level simulation data is available; users can provide RTL-level SAIF files which contain the switching activity of the primary inputs, hierarchical ports, and synthesis invariant elements such as sequential elements.

When RTL-level SAIF files are provided, it is possible that RTL objects have been renamed during synthesis.

In order to map the RTL objects to the gate-level objects users must issue the following command:

```
set_rtl_to_gate_name -rtl <rtl_name> -gate <gate_name>
```

You can also use the output of Formality `report_matched_point` by applying the `-names_file file_name` option to the `read_saif` command.

Gate-level SAIF

The same `read_saif` command is used to parse the gate-level SAIF file, although users may apply different options. Assuming the gate-level SAIF contains activity annotation for all elements in the design, PrimePower can accurately calculate the average power consumption of the design.

SAIF File Commands

PrimePower provides the following commands to read, reset, and report switching activity in the SAIF file format.

```
read_saif
  -input file_name -instance_name name
  [-ignore ignore_name]
  [-ignore_absolute ig_absolute_name]
  [-exclude exclude_file_name]
  [-exclude_absolute ex_absolute_file_name]
  [-names_file name_changes_log_file]
  [-scale scale_value] [-unit_base unit_value]
  [-khrate khrate_value]
  [-rtl_direct]
  [-verbose]
```

`-input file_name`

Specifies the name of the SAIF file to be read.

`-instance_name name`

The `read_saif` command assumes that the current design is instantiated as an instance in the testbench used to generate the SAIF file. The current design therefore appears as an instance in the SAIF file. The `-instance_name` argument specifies the name of the instance of the current design as it appears in the SAIF file. The `read_saif` command annotates all sub-instances in the hierarchy of the specified instance, and

annotates the instance itself. You must enter each instance name completely. For example, if the current design is instantiated in the simulation environment as TOP/DUT/U1, use -instance TOP/DUT/U1 and not -instance TOP/DUT/U. You cannot end the path with the hierarchy separator (/).

`-ignore ignore_name`

Specifies the name of an instance in the SAIF file for which switching activity is to be ignored. The `read_saif` command ignores switching activity within the hierarchy of that instance in the SAIF file. The `ignore_name` argument can be a hierarchical name separated by a slash (/). The `-instance_name` argument is applied first. The `read_saif` command then strips off the prefix of a net name that matches name before deciding whether this net name is under the hierarchy specified by the `-ignore` option.

`-names_file name_changes_log_file`

Specifies a previously-created log file, usually the output of the DC command: `change_names -log`, to be used as the input to the `read_saif` command. Whereas the design file can contain multiple name changes, the SAIF file contains only the original object names. Thus, the object search by name might fail to find objects whose names have been changed. The `name_changes_log_file` contains a list of `change_names` commands for cells, ports, and nets in the current design, and thus provides mapping information so that the `read_saif` command can identify the correct objects, even if the names have changed. The `name_changes_log_file` is a text file, so it can also be manually manipulated. The file format is similar to the output of `change_names -verbose`.

`-ignore_absolute ig_absolute_name`

Functions the same as the `-ignore` option, except that it is not affected by the `-instance_name` argument. The `read_saif` command determines whether a net name is under the `-ignore` hierarchy before applying the `-instance_name` argument.

`-exclude exclude_file_name`

Specifies the name of a file that contains a list of names to be ignored. The `exclude_file_name` argument is used when you want to ignore switching activity for several instances. The file must contain each `ignore_name` on a separate line, without the `-exclude` option. As with the `-ignore` option, the ignore names are recognized after the `-instance_name` argument.

`-exclude_absolute ex_absolute_file_name`

Has the same effect as the `-exclude` option, except that the file contains a list of absolute names that are ignored that are not affected by the `-instance_name` argument.

`-scale scale_value`

Specifies the value of the scaling factor for the base time unit. Allowed values for `scale_value` are 1 (the default), 10, or 100. The time unit is usually obtained automatically from the link library.

`-unit_base unit_value`

Specifies the base time unit. Allowed values for `unit_value` are ns (the default), us, or ps.

`-khrate khrate_value`

Specifies a default derating factor value to be used for inertial glitches in the SAIF file. If `-khrate` is not specified, a default derating factor of 0.5 is used.

`-rtl_direct`

Indicates that the SAIF file was generated from RTL simulation without reading in a RTL forward SAIF file. When using this option, make sure that the SAIF file obtained from simulation was done on an RTL design, and that no RTL forward SAIF files were used. Examples of such RTL non-forward SAIF file flows include: When using the power PLI/FLI, the `read_rtl_saif` command was not performed, and the net monitoring was set to `rtl_on` (using the `set_gate_level_monitoring` command in the PLI, and the `set_net_monitoring_policy` command in FLI); when using the `vcd2saif` utility, the `-rtl` argument was not used.

`-verbose`

Specifies verbose mode and for instance gives verbose warnings on design objects in the SAIF file that could not be annotated on the design.

Reading SAIF Files Using `merge_saif`

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 is an example of how the `merge_saif` command can be used. We assume that the design has three modes: standby, slow and fast; and that the SAIF files `standby.bsaif`, `slow.bsaif` and

fast.bsaif are generated for these modes. Depending on the expected usage of the design, we give the following weighting to each SAIF file:

```
standby.saif 80%
slow.bsaif   5%
fast.bsaif   15%
```

The SAIF files can then be read using the following command:

```
pp_shell-t> merge_saif -input_list \
                  {-input standby.saif -weight 80 \
                  -input slow.bsaif   -weight 5  \
                  -input fast.bsaif   -weight 15 } \
                  -instance tb/i
```

The `-output` argument of the `merge_saif` command can be used to generate a SAIF file containing the weighted sum of the switching activities.

After `merge_saif` reads each individual SAIF file, it uses a switching activity propagation mechanism to estimate the switching activity of design nets that are not included in the SAIF file. One can therefore use the following:

```
pp_shell-t> merge_saif -input_list {-input rtl.bsaif \
                  -weight 100} -instance tb/i -output estimate.bsaif
```

To generate a gate-level SAIF file with estimated switching activity information from an RTL SAIF file.

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

Currently, SAIF files with SDPD information cannot be used by the `merge_saif` command.

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

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

```
merge_saif
  -input_list {-input filename -weight weight ... }
  -instance_name inst_name
  [-output merged_saif_name]
  [-simple_merge]
  [-ignore ignore_name]
  [-ignore_absolute ig_absolute_name]
  [-exclude exclude_file_name]
  [-exclude_absolute ex_absolute_file_name]
  [-unit_base unit_value] [-scale scale_value]
  [-khrate khrate_value]
  [-rtl_direct]
  [-verbose]
```

`-input_list {-input filename -weight weight ... }`

Specifies the name and the corresponding weight of the SAIF file list. The argument of the `-input_list` option contains a list of { `-input filename -weight weight` }, where `-input` and `-weight` are keywords. In addition, each weight must be between 0 and 100, and the sum of all weights is equal to 100. For example, { `-input gate_back_1.saif -weight 20 -input gate_back_2.saif -weight 80` } means that the files `gate_back_1.saif` and `gate_back_2.saif` are weighted by 20% and 80% respectively.

`-simple_merge`

Indicates that the switching activity propagation mechanism will not be used to estimate the switching activity of design nets missing from the SAIF files. This can be used without loss of switching activity accuracy when all SAIF files are to annotate 100% of the nets, ports and pins of the current design. Use the `report_saif` command to verify that each SAIF file annotates 100% of the current design.

`-output merged_saif_name`

Specifies the name of the output merged SAIF file.

Removing Switching Activity Annotation

Switching activity annotation can be removed from individual design objects using the `reset_switching_activity` command. This example:

```
pp_shell-t> reset_switching_activity objects
```

Removes the simple and SDPD switching activity annotation from the specified objects.

In order to view the details of the power calculation, the Synopsys libraries must be compiled with the `library_features` attribute (`report_power_calculation`).

Without this library attribute, the output of the `report_power_calculation` command will not provide the power table indexes used to calculate the power.

When no options are provided to the `reset_switching_activity` command it removes all previously annotated and propagated switching activity for the complete design hierarchy.

To selectively remove switching activity, users can apply the `reset_switching_activity` command options listed below:

```
reset_switching_activity
    -static_probability
    -tran
    -state state_condition
    -path path sources
    -nohier
    -instance instance_list
    -object_list
    -verbose
```

Refer to the man pages for more details on these options.

It is recommended that switching activity information from previous switching activity annotation is removed using the `reset_switching_activity` command before reading new SAIF files.

Writing SAIF File Out of PrimePower

PrimePower can write out SAIF file containing user annotated and propagated switching activity.

Table 8-1 Options for write_saif command

Argument	Description
-output	Required option. Specify the name of the file to be generated.
-instances	Specify optional list of instances for which SAIF file is created. If not specified, the current instance is the default.
-no_hier	Default behavior is to create SAIF file for all the objects under the specified or default instance. Use this option to turn off.
-rtl	When used, this option generates a SAIF file containing switching activity for synthesis invariant objects. When this option is used on an unmapped design, the SAIF file is similar to an RTL SAIF file generated by RTL simulation flow.
-propagated	This option propagates the activity and writes out SAIF containing both annotated and estimated activity; When not used, the <code>write_saif</code> command generates a SAIF file that has only user annotated activity. If no activity is annotated by the user, then no values are captured in the file unless this option is used
-exclude_sdpd	Exclude state- and path-dependent toggle rates and static probabilities

Estimating Unannotated Switching Activity

PrimePower needs switching activity information on all design nets and SDPD information on all design cells and pins in order to calculate power. Switching activity that is not user annotated will be estimated automatically before power is calculated.

This is performed in three stages:

- Design nets whose switching activity can be calculated accurately or cannot be propagated are set to some default values. We say that such nets are default annotated.

- The user annotated and default annotated switching activities are then 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 then used to derive the non-annotated SDPD switching activity.

Annotating Design Nets with Default Switching Activity Values

Design nets are annotated with default switching activity values when the switching activity can be accurately derived or when the switching activity cannot be estimated using the propagation mechanism described below. The first type of nets include nets driven by clocks since the switching activity information can be accurately derived from the clock waveform. For the second type of nets, it should be noted that the propagation mechanism uses the functionality of design cells to propagate the input switching activity to the cell outputs. Black-box cells have unknown functionality, and therefore the switching activity of block-box outputs cannot be derived using the propagation mechanism. Black-box outputs that are not user annotated will be annotated with a default value.

The following lists all the different types of design nets that are annotated by default values:

- Nets driven by constants: A default 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 default values for the toggle rate and static probability are derived from the clock waveform.

- Nets driving or driven by buffers: If the buffer input or output net is user or default annotated, then the unannotated buffer output or input is default annotated with the switching activity values on the annotated input or output.
- Nets driving or driven by inverters: If the inverter input or output net is user or default annotated, then the unannotated inverter output or input is default annotated with the same toggle rate value, and with the inverted static probability value. If the annotated static probability value is sp then the inverted static probability value is $1.0 - sp$.
- 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 default annotated with the same toggle rate value and with the inverted static probability value.
- Primary inputs and outputs of black-box cells: The switching activity of primary inputs and black-box outputs cannot be propagated. Default switching activity depending on the value of the `pwr_default_static_probability` and `pwr_default_toggle_rate` variables is used.

The default static probability value is the value of the `pwr_default_static_probability` variable. The default toggle rate value is the value of the `pwr_default_toggle_rate` multiplied by the related clock frequency. The related clock can be specified using the `-clock` argument of the `set_switching_activity` command. If no related clock is specified on the net, the clock with the highest frequency is used. Both variables have an initial default value of 0.5.

Propagating Switching Activity

The switching activity of design nets that are not user or default annotated are then derived using a propagation mechanism. This mechanism is basically a zero delay simulator. 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 unannotated cell outputs.

The number of simulation steps performed by this mechanism depends on the analysis effort argument 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 unannotated value is estimated by the propagation mechanism.

Deriving SDPD Switching Activity

If an RTL SAIF file or a gate-level SAIF file without SDPD information is used to annotate the design switching activity, then PrimePower needs to estimate the required SDPD switching activity information. After obtaining the simple switching activity (from user annotation, or by switching activity propagation), PrimePower will estimate the SD static probability information for every cell, and the SDPD toggle rate information for every cell pin. This information is obtained from the switching activities of each cell input and output pins. Although the

SDPD estimation mechanism produces fairly accurate power calculations, for the most accurate power results, use gate-level SAIF files with SDPD information.

Correlation

While propagating switching activity through the design, PrimePower makes certain statistical assumptions. However, the logic states of gates' inputs can have interdependencies that affect the accuracy of any statistical model.

Such interdependency of inputs is called correlation. Correlation affects the accuracy of PrimePower propagation of toggle rates. Because accurate analysis depends on accurate toggle rates, correlation also affects the accuracy of power analysis.

PrimePower 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, PrimePower has no access to information about correlation external to the design. If correlation exists between the primary inputs of the design, PrimePower does not recognize the correlation.

Complex Cells

If your design has black boxes, such as complex cells, RAM, ROM, or macrocells, annotate switching activity at the outputs of these elements.

Debugging the Annotation

PrimePower provides several commands to debug the activity annotation. The `set_switching` commands and the user provided SAIF files are parsed as soon as the commands are issued.

If there are problems with mapping and the annotation is not as expected, PrimePower provides the following commands with which to view the annotated information used for power analysis:

- `get_switching_activity`
- `report_saif`

The `report_saif` command can be used to display information on the annotated switching activity. A call to `report_saif` displays a report showing how much of the design objects are annotated with user annotated switching activity, default switching activity and propagated switching activity.

```
pp_shell  report_saif
```

```

*****
Report : Saif
        -gate
Design : mac
Version: X-2005.06
Date   : February 22, 2009 10:48 am
*****

```

Switching Activity Statistics for "mac"

Object Type	User	Default	Propagated	No	
Activity (%)	Annotated (%)	Annotated (%)	Annotated (%)	Activity (%)	
	Total				
Nets	1858(100.00%)	0(0.00%)	0(0.00%)	0(0.00%)	1858
Power Arcs	4093(63.43%)		0(0.00%)	0(0.00%)	
2360(36.57%)	6453				

Signal Probability Statistics for "mac"

Object Type	User	Default	Propagated	No	
Probability (%)	Annotated (%)	Annotated (%)	Annotated (%)	Probability (%)	
	Total				
Nets	1858(100.00%)	0(0.00%)	0(0.00%)	0(0.00%)	1858
Leakage States	0(0.00%)		0(0.00%)	0(0.00%)	
1339(100.00%)	1339				

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

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

The following attributes can be accessed to get the value of toggle rate and signal probability:

```
pp_toggle_rate_value  
pp_toggle_rate_assert_flag  
pp_glitch_rate_value  
pp_glitch_rate_assert_flag  
pp_signal_probability_value  
pp_signal_probability_assert_flag
```

Case Analysis

Case analysis lets you perform power analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design. Case analysis concepts are described in the following sections:

- [Performing Case Analysis](#)
- [Constant Propagation Based on Cell Logic](#)
- [Correlation](#)
- [Complex Cells](#)
- [Setting Case Analysis Values](#)
- [Reporting Case Analysis Values](#)
- [Removing Case Analysis Values](#)
- [Constant Propagation Log File](#)
- [Power Calculation](#)

Performing Case Analysis

Case analysis lets you perform power analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design. Setting a case analysis with a logic constant propagates the constant forward through the design, then disables paths where the constant is propagated. Setting a case analysis with a logic transition (rising or falling) eliminates certain paths by limiting the transitions considered during analysis.

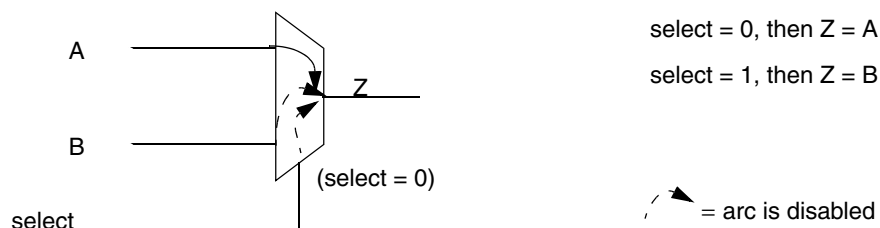
When you perform case analysis, keep the following points in mind:

- Case analysis propagates the constant values specified on nets forward (but not backward) through the design.
- Case analysis does not affect sequential cells if the library was compiled with Library Compiler v3.3 or earlier.
- Propagation of constants across the sequential cells is by default disabled and can be enabled by setting the variable `case_analysis_sequential_propagation` to `always`.

Example 1

The multiplexer in shown below has two signals (A and B) going in, one signal (Z) going out, and 0 is set on a select signal coming into the multiplexer. The select disables the arc from B to Z because the constant is blocking the data from B to Z.

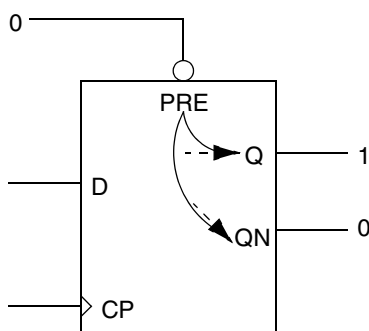
Figure 8-1 Constant Blocking of Data



Example 2

When case analysis propagates a constant to a sequential element's asynchronous preset or clear pin, the sequential cell outputs also propagate the constant 1 or 0, as shown below.

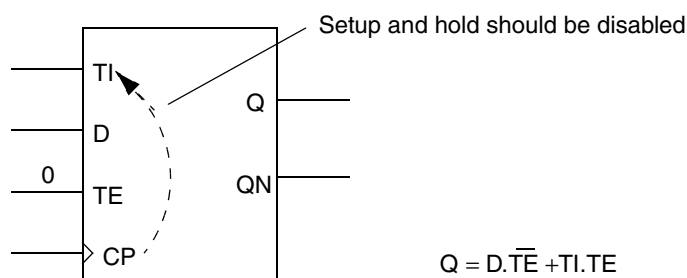
Figure 8-2 Constant Propagation Through an Asynchronous Input



Example 3

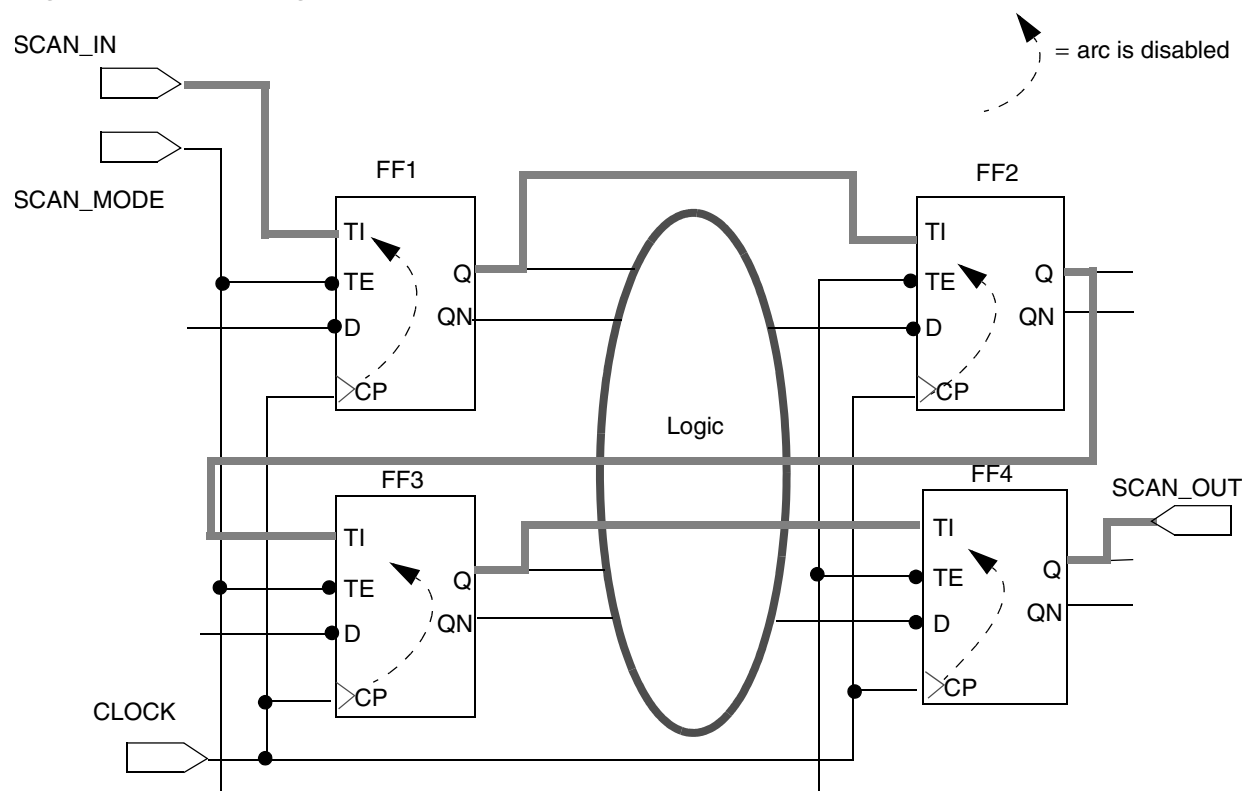
When your design uses scan flip-flops, case analysis is useful for disabling the scan chain. To do this, set the scan mode control pin to the constant value that disables the test mode. PrimePower propagates the test-mode constant value to the scan-mode pin of each scan flip-flop. Case analysis can determine from the constant values which timing arcs to disable, as shown in below.

Figure 8-3 Case Analysis Disabling Timing Checks



The most common use of case analysis is to disable a scan chain, as shown in below. Setting the SCAN_MODE port to 0 disables the scan chain.

Figure 8-4 Using Case Analysis to Disable a Scan Chain



By setting a logic constant of 0 on the SCAN_MODE port, PrimePower disables the scan chain that starts at SCAN_IN and ends at SCAN_OUT. PrimePower disables the scan chain because the logic 0 set on the SCAN_MODE port propagates to the FF1/TE, FF2/TE, FF3/TE, and FF4/TE pins. A logic constant on the TE pin of each scan flip-flop disables the setup and hold arcs from CP to TI because of the sequential case analysis on the FF1, FF2, FF3, and FF4 cells.

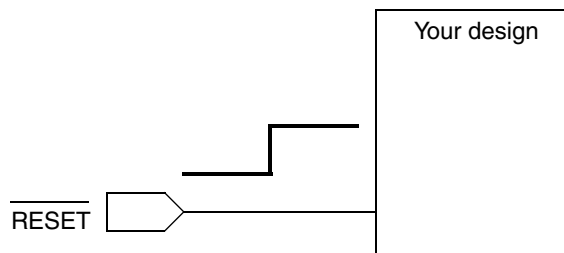
To set case analysis with a 0 value, enter

```
pp_shell> set_case_analysis 0 [get_ports "SCAN_MODE"]
```

Example 4

Setting a case analysis for a transition can be useful for analyzing the behavior of a circuit for a specific situation. For example, in the circuit shown below, a rising-edge transition on the reset input brings the device out of the reset mode.

Figure 8-5 Case Analysis for a Rising Edge



To analyze the timing of the circuit coming out of reset mode, you are only concerned about rising edges on the reset input, not logic 0, logic 1, or falling edges. To set case analysis in this situation, enter

```
pp_shell> set_case_analysis rising [get_ports RESET]
```

Constant Propagation Based on Cell Logic

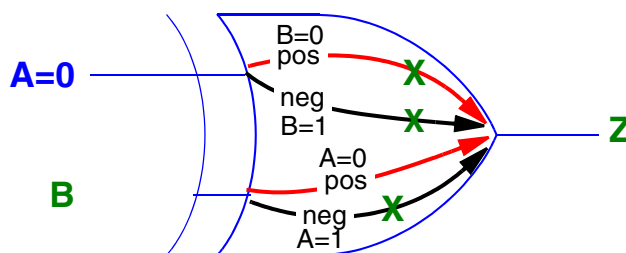
Case analysis can disable timing arcs and propagate case values for the following types of cells:

- Combinational cells with the `function` attribute specified in the library description
- Sequential cells with Boolean State Table (BST) specified in the library description
- Cells with timing and power arcs that have state-dependent conditions specified using the `when` statement in Library Compiler. (See the Library Compiler user guides for the correct usage of the `when` statement and state-dependent delays.)

If a cell has any input at a logic constant case value, arcs that could never propagate a transition are disabled. If any other arcs have a `when` condition, those conditions are evaluated. The `when` arcs are disabled if the `when` expression evaluates to false. Note that the `when` statement cannot enable an arc that has been disabled by case analysis.

The figure below shows how arcs of an XOR gate are disabled when using the `when` statement in the library description. When $A = 0$, all arcs connected to A, and negative_unate arcs from B to Z are disabled. Note, there might be several arcs of the same sense between the two pins.

Figure 8-6 Disabled Arcs Using the when Statement



The only enabled arcs are positive_unate arcs from **B** to **Z**.
(X identifies disabled arcs)

Setting Case Analysis Values

The `set_case_analysis` command performs case analysis. It specifies that a port or pin is at a constant logic value 1 or 0, or is considered with a rising or falling transition. For more information, see the `set_case_analysis` man page.

Example 1

To perform power analysis for system mode by setting the case analysis for the test port to constant logic 0, enter

```
pp_shell> set_case_analysis 0 Test
```

Example 3

You can use case analysis in combination with mode analysis. For example, use case analysis to specify that certain internal logic is a constant value because the RAM block is in read mode:

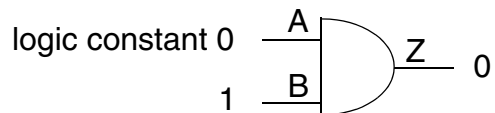
```
pp_shell> set_case_analysis 1 U1/U2/select_read
```

As shown in the following examples, if there are logic constants and case analysis on inputs of a cell, and the logic constants are set to any controlling logic value of that cell, then the logic constant is propagated from the output; otherwise case analysis is propagated.

Example 4

If you have an AND gate and input pin A has logic constant 0 (controlling the logic value of the AND gate), the logic constant 0 is propagated from the output pin, irrespective of the value on B.

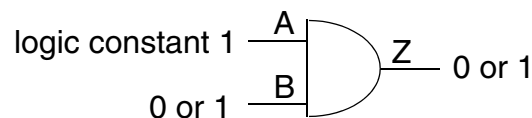
For example:



Example 5

If you have the same AND gate, but pin A has logic constant 1, it is not a controlling logic value; the logic constant is not propagated to the output pin.

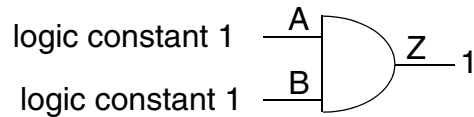
For example:



Example 6

If you have the same AND gate and both pins A and B have logic constant 1 (noncontrolling value), then a logic constant of 1 is propagated from the output of the gate.

For example:



Reporting Case Analysis Values

The `report_case_analysis` command reports the case analysis values.

Example

To report the case analysis values, enter

```
pp_shell> report_case_analysis
```

PrimePower displays a report similar to this:

```
*****
Report : case_analysis
Design : false_path
*****
```

Pin name	Case analysis value
test_port	0
U1/U2/core/WR	1

The `report_case_analysis -all` shows both the logic constants and case analysis set on the design.

For example,

```
pp_shell> report_case_analysis -all
*****
Report : case_analysis
        -all
Design : TOP
Version: 2004.12
Date   : Wed Oct 15 11:23:37 2004
*****
```

Pin name	Logic constant value

U3/Logic0/output	0

Pin name	User case analysis
value	

HRS	1
ALARM	1
MINS	1

Removing Case Analysis Values

The `remove_case_analysis` command removes case analysis values.

Example

To remove case analysis values from the `test_port` design, enter

```
pp_shell> remove_case_analysis test_port
```

To suppress the propagation of logic constants (including those set with case analysis and pins tied to logic high or low), set the variable `disable_case_analysis` to true. You can use this feature when you want to write scripts for Design Compiler that can be synthesized correctly.

Constant Propagation Log File

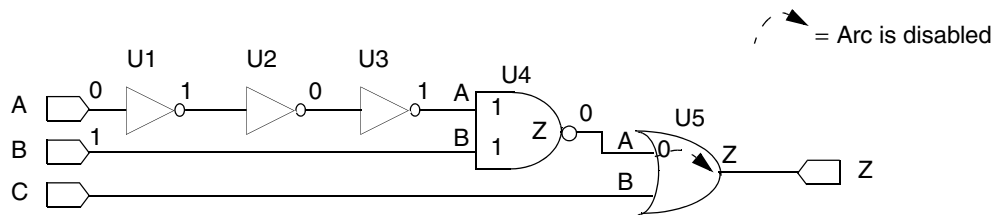
When PrimePower performs constant propagation, it disables timing arcs at the boundary of the domain where a constant propagates. For this reason, when PrimePower disables a timing arc, knowing the exact origin of the constant value propagated on a given pin of the design might be difficult.

To help you determine the cause of a timing arc disabled because of constant propagation, you can set the `case_analysis_log_file` variable to a file name. The constant propagation process is logged to the named file. You can also view the timing arcs disabled after constant propagation with the `report_disable_timing` command.

Example

This command sequence sets case analysis and reports the disabled timing arcs to the log file `my_design_cnst.log` for the example circuit.

```
pp_shell> set_case_analysis 0 {get_port "A"}
pp_shell> set_case_analysis 1 {get_port "B"}
pp_shell> set case_analysis_log_file my_design_cnst.log
```



In the example, the only arc that PrimePower needs to disable is the arc in U5 from U5/A to U5/Z. PrimePower needs to disable this arc because U5/A is at constant value 0 and no constant is propagated to U5/Z. The constant propagation log file my_design_cnst.log looks like this:

```
*****
Report : case_analysis propagation
Design : my_design
Version: 2004.12
*****
```

1.1 Forward propagation on NET pins (level 1)

```
-----
Propagation of logic constant '0' from pin 'A' on net 'A':
> pin 'U1/A' is at logic '0'
Propagation of logic constant '1' from pin 'B' on net 'B':
> pin 'U4/B' is at logic '1'
```

1.2 Forward propagation through CELLS (level 1)

```
-----
Cell 'U1' (libcell 'IV') :
input pin U1/A is at logic '0'
> output pin 'U1/Z' is at logic '1'
```

2.1 Forward propagation on NET pins (level 2)

```
-----
Propagation of logic constant '1' from pin 'U1/Z' on net
'w1':
> pin 'U2/A' is at logic '1'
```

2.2 Forward propagation through CELLS (level 2)

```
Cell 'U2' (libcell 'IV') :  
  input pin U2/A is at logic '1'  
  > output pin 'U2/Z' is at logic '0'
```

3.1 Forward propagation on NET pins (level 3)

```
Propagation of logic constant '0' from pin 'U2/Z' on net  
'w2':  
  > pin 'U3/A' is at logic '0'
```

3.2 Forward propagation through CELLS (level 3)

```
Cell 'U3' (libcell 'IV') :  
  input pin U3/A is at logic '0'  
  > output pin 'U3/Z' is at logic '1'
```

4.1 Forward propagation on NET pins (level 4)

```
Propagation of logic constant '1' from pin 'U3/Z' on net  
'w3':  
  > pin 'U4/A' is at logic '1'
```

4.2 Forward propagation through CELLS (level 4)

```
Cell 'U4' (libcell 'ND2') :  
  input pin U4/A is at logic '1'  
  input pin U4/B is at logic '1'  
  > output pin 'U4/Z' is at logic '0'
```

5.1 Forward propagation on NET pins (level 5)

```
Propagation of logic constant '0' from pin 'U4/Z' on net  
'w4':  
  > pin 'U5/A' is at logic '0'
```

5.2 Forward propagation through CELLS (level 5)

6. End of Logic Constant Propagation

The result of a `report_disable_timing` command looks like this:

Cell or Port	From	To	Flag	Reason
U5	A	Z	c	A = 0

Power Calculation

The actual power calculation is performed when the user issues the `calculate_power` command. This command initially determines if all the nets have been annotated and for any user-defined cases. If necessary, it applies default values and performs propagation so that toggle rates are assigned to all the nets within the design before analyzing the power.

A number of variables and command line options control the power calculation. These are described in the sections below.

Power Calculation Variables

The following variables control some aspects of the power calculation for both event- and toggle-based analysis:

```
pwr_match_xstate = "0"
```

The `pwr_match_xstate` variable is used in evaluating the "when" function in the power tables. By default, this value is set to 0, which means that if the inputs specified in the when condition are X, they will be treated as 0 when determining which power table to access.

```
pwr_xpower_scale_factor = "0.5"
```

By default, transitions to and from X are assumed to be 1/2 the power consumption of a normal transition. Increasing the value to 1 results in a more pessimistic power report.

```
pwr_calc_use_slew [min|max|ave]
```

This variable defines which ramp to pick when calculating the transition times for the nets within a design. By default, PrimePower uses the max value.

```
pwr_limit_extrapolation true|false (default false)
```

By default, PrimePower extrapolates indefinitely, if the data point for internal power lookup is out-of-range. If however, the variable is set to `true`, then PrimePower limits the extrapolation.

Table 8-2 *Power tables*

nlpm power table	PrimePower stops extrapolation at one additional index grid.
sppm power table	PrimePower stops extrapolation at the specified range.

Users may want to limit the extrapolation for more accurate power values, if there are many high-fanout nets (such as clock or global reset nets).

Calculate Power Options

The `calculate_power` command has a number of options which apply to the calculation of power based on toggle rates.

By default, if no VCD activity is provided, PrimePower calculates the average power for the duration of the SAIF file; or until there is convergence when no SAIF data is provided.

If both VCD activity and SAIF activity is provided, then PrimePower, by default, uses the VCD activity and performs event-based analysis.

To force toggle-based activity, the user can apply the following options:

- `saif` instructs PrimePower to use the SAIF file instead of the VCD activity.
- `vcd2saif` instructs PrimePower to internally convert the VCD data to SAIF data and to perform toggle-rate analysis.

The user can later apply the `write_saif` command to write out the SAIF file.

To calculate the power for specific instances, the `-instance list` option can be applied. Power calculation for other instances is not performed.

To generate an average cycle waveform, the user must apply the `-waveform` option to the `calculate_power` command. The commands and options are described in the section below.

Average Cycle Waveform Data

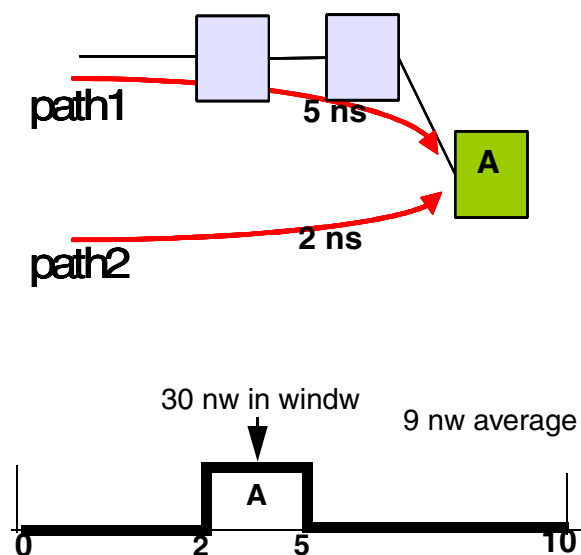
If the `-waveform` option is applied to `calculate_power`, PrimePower will generate an average waveform based on all existing clocks. It will apply common factoring to generate the appropriate waveform period. For example, if `clk1` has a period of 2 ns and `clk2` has a period of 3 ns; then PrimePower will generate average waveforms with a period of 6 ns. Users can specify the `-clocks` option to instruct PrimePower to generate an average cycle waveform for all the clocks specified in the list.

The average cycle waveform is based on the average power over one clock cycle or common base period of multiple clocks. It uses the fact that toggling (hence the power) on each net occurs at different timing windows within a clock cycle.

For each leaf-cell, a timing window is created by calculating the difference in the input path delay and pin transition times for all the paths to all the leaf-level cells. Cells with many paths, are likely to have a large timing window.

Ultimately, the sum of all the leaf cells is used to generate an average cycle waveform which helps users identify the approximate location and value of the power peak per cycle.

Figure 8-7 Average Power Waveform



- Referring to cell A as an example
 - Average power is 9 nw
 - Timing window created from the difference between the input path. In this case it is $(5-2) = 3$ ns
 - Cycle is 10 (as defined by the clock)
 - Power in its window is $9 \times 10 / (5-2) = 30$

The following items need to be considered before using this feature:

- The clocks have to have been defined with the `create_clock` command.
- The specified clocks should be synchronous.
- The specified clocks should have a fairly small common base period.

Use the `set_waveform_options` to control the waveform output. The value for `-interval` can be very small, and is default to 0.01 ns. The value for `-interval` is actually resolution.

A new variable `pwr_avg_wav_arrival_distribution` is introduced.

Note:

The `-effort high` option has no effect on the average waveform generation.

This variable affects the calculation of the power waveform. By default, the variable is set to `even`. In this case, PrimePower assumes the power consumption for a particular cell is distributed evenly across the timing window as shown in the example above.

Users can set the value to `triangle`, in which case the signal arrival time points are assumed to occur within the timing window, so that a triangular power waveform is calculated. This option is more computationally intensive, but may produce more accurate waveform results.

The following commands can affect the waveform:

- `set_clock_transition`: affects the clock transition
- `set_propagated_clock`: affects the clock latencies
- `set_input_transition`: affects the input port transition
- `set_input_delay`: affects arrivals at the input port
- `set_case_analysis`: select mode, also affects timing windows
- Other SDC commands affect transition or timing windows

9

Event-based Power Analysis

PrimePower event-based analysis relies on the gate-level simulation activity to perform event-based power calculation. Detailed time-based power information is generated enabling the user to determine both average and peak power.

Overview

This chapter describes the commands and user options for performing accurate event-based power analysis within PrimePower. It covers the following topics:

- [Event-based Power Analysis](#)
- [Activity File Formats](#)
- [Power Calculation](#)

- [Interrupting and Resuming Power Calculation](#)
- [Power Waveforms](#)

Event-based Power Analysis

PrimePower's event-based power analysis produces accurate time-based information when event-based simulation activity is provided. The simulation data is specified with the `read_vcd` command and processed when the user issues the `calculate_power` command.

The processing of the event based data involves the following steps:

1. PrimePower calculates the ramp and load for every cell-level input pin within the design.
2. PrimePower processes the events in the simulation activity file to determine the total energy for that event.
3. PrimePower sums the energy for each leaf-level cell to determine the average and peak power consumption.

Details on power calculation are provided in [Chapter 1, "Introduction."](#) The sections that follow provide information on the commands applied to read the simulation data and calculate power.

Activity File Formats

The PrimePower analysis engine processes VCD and VPD activity files. However, users can input other activity file formats which are converted within PrimePower to VCD with the appropriate utility. To

determine the format, PrimePower relies on the file extension. It then applies the corresponding conversion utility which must be in the user's search path.

The table below lists the formats, extension and conversion utilities supported by PrimePower.

File type	Extension	Utility
VCD	default file type	none
VPD	.vpd	none
compressed VCD	.Z	uncompress
gzipped VCD	.gz	gunzip
FSDB	.fsdb	\$SYNOPSYS/bin/fsdb2vcd

Note:

If the file suffix is not one of those listed, PrimePower assumes the file is a VCD file and reads the data directly.

Activity files are specified with the following command regardless of the file format:

Syntax

```
read_vcd [-path prefix] -strip_path [path prefix]
[-pipe command] [-ignore_scope_type] [-zero_delay]
filename
```

To correctly parse the activity file, a number of options may be required. Described below are the options for performing specific user tasks.

Mapping the Testbench Instance to the Design Module

The activity file is generally created at the testbench level, where the design-under-test is an instance within the HDL testbench hierarchy. In order to map the testbench instance to the design module specified as the current design, the user must apply the `-strip_path` option.

For example: If the testbench TB instantiates the design module TOP as U1; then the following `-strip_path` option should be specified:

```
read_vcd -strip_path TB/U1 .....
```

Note:

Even if the top-level design was not specified in the `$dumpvars` task, the VCD file still lists the complete hierarchy starting with the testbench.

Handling Large Activity Files

There are a number of choices regarding the handling of large amounts of activity data. PrimePower does support the processing of VCD files > 2GB on all platforms, but some users may wish to avoid generating and storing large files.

Users can generate compressed formats such as FSDB or gzipped VCD to reduce the activity file size. Within PrimePower, the data from the compressed files is converted to the VCD format and piped to the analysis engine. The converted data is not stored.

Another option is to pipe the activity data directly from the HDL simulation to PrimePower. In this case, the activity data is fed directly from the HDL task within the simulation; to the power analysis engine instead of being saved into a file. To do this, the users must follow the steps below:

1. The HDL simulation must be setup to generate an activity file. For example, if generating a large VCD file from a Verilog simulation; the testbench must include the \$dumpvars / \$dumpfile PLI tasks to generate the VCD data.
2. The user must specify the commands or runscript to invoke the HDL simulation with the -pipe option. For example, if the user has a runscript: `run_vcs` to execute a VCS simulation; which contains the following command:

```
vcs -R -f arguments -l log
```

you can invoke the simulation within PrimePower to pipe the VCD data directly by adding either of the following options to the `read_vcd -pipe` option:

```
-pipe run_vcs
```

or

```
-pipe "vcs -R -f arguments -l log" ...
```

Handling Activity Data From Mixed-language Simulations

Activity data generated from mixed-language simulation may contain net and instance references whose case is incompatible with the design data loaded into PrimePower.

To address these issues, PrimePower provides the following variables to control the handling of mixed-language simulation activity files.

- `pwr_read_vcd_ignore_case` (default: `false`)

This variable controls how PrimePower matches the net and cell names from the VCD file and those from the Verilog design. By default, the value is set to `false`, and PrimePower considers the case in the VCD file when matching instance and net names in the design.

Users may want to set this to `true`, when reading VCD files generated from mixed-language simulations wherein the VCD file generated from a VHDL testbench may contain case-insensitive names. Some simulators may output VCD files which contain all lowercase or all uppercase. The case-sensitivity needs to be ignored to map to the Verilog net names in the design.

- `pwr_vhdl_always_ignore_case` (default: `true`)

This variable controls how PrimePower processes VCD files generated from simulation of a VHDL design. By default, because VHDL is case insensitive, the value is `true` and the matching of nets in the VCD file is done without regard to the case.

For mixed designs, users may need to apply this variable to handle mapping to Verilog portions of the design which are case-sensitive.

Handling VCD Files With Non-module Scopes

Some VCD files contain scopes such as tasks and functions. The PrimePower VCD file parser by default, does not understand hierarchy other than module-level hierarchy and fails to recognize VCD hierarchy defined by functions and tasks.

If the VCD file contains such hierarchical elements, the user must apply the `-ignore_scope_type` option to `read_vcd` in order for PrimePower to correctly interpret the VCD hierarchy.

For example if the VCD file contains \$scope entries such as:

```
$scope task tb_t1 $end  
$scope function tb_f1 $end
```

Users need to apply the `-ignore_scope_type` option. This is not common in gate-level environments.

Processing Multiple Activity Files

PrimePower only processes one activity file at a time. If multiple `read_vcd` commands are applied, only the last activity file name is stored, and processed during the `calculate_power` command. Since the annotation of the simulation values occurs during the `calculate_power` command; only the annotation in the last file specified is applied.

If users perform multiple simulations and have multiple activity files for a particular design; these can be analyzed within PrimePower by issuing the commands to read, process and report power for each activity file separately.

For example; if the user has two VCD files for a particular design:

test1.vcd and test2.vcd

The following commands can be used to determine the power of each test:

```
set vcd_list {test1.vcd test2.vcd}
for each vcd_file $vcd_list {
  read_vcd $vcd_file -strip_path ....
  calculate_power
  report_power -file ${vcd_file}_pwr
}
```

The power consumption for the vector sets would be stored in test1.vcd_pwr.rpt and test2.vcd_pwr.rpt respectively.

Zero-delay VCD Files

If the VCD file is generated from a zero-delay simulation; the user will need to provide Synopsys design constraints as well as specifying the

-zero_delay option to the read_vcd command to ensure accurate calculation of the peak power and waveform data.

The -zero_delay option causes PrimePower to calculate the timing windows for each instance within the user specified clock cycle in order to distribute the power consumed by the transition. Refer to the section in the [“Average Cycle Waveform Data”](#) section in [Chapter 8](#), “for more information on how the average cycle waveform is generated.

Debugging Problems With the VCD File

Errors with processing the activity file data are not uncovered until the user issues the `calculate_power` command. At this point, PrimePower maps the VCD hierarchy to the current design and annotates the values. Below, you can find a description of types of errors which could occur:

1. The VCD hierarchy does not match that of the current design.

In this case, PrimePower issues warnings for all the nets in the design which cannot be found in the VCD file.

For example, the following type of warning will be generated:

```
Warning: The net XXX is not covered by VCD file
"verilog.dump" {sim - 220}
```

The mismatch generally occurs if the `read_vcd -strip_path` option This type of error occurs, either because the `-strip_path` option was incorrectly specified; or the VCD file does not contain all levels of hierarchy to the leaf-level cell OR the VCD scopes are of types other than modules.

To debug this type of problem the following command can be used to check the VCD file hierarchy.

```
report_vcd_hier [-pipe command] [-full_path] [-find
pattern] [-ignore_scope_type]
[filename]
```

The `-find` option is useful in locating the level of hierarchy at which the cell resides. For example to locate the instance `add_23`; the command: `report_vcd_hier -find add_23` can generate the hierarchical path for the desired instance:

```
top/mac/add_23
```

The `-full_path` option outputs the hierarchical information with the full path which can be easily cut and pasted into the `read_vcd -strip_path` argument.

2. Syntax errors during parsing of the activity file.

Problems with parsing the activity file can be due to a number of reasons.

- The file format may not be one of those listed. If an EVCD file is supplied, PrimePower generates syntax errors. PrimePower does not support the enhanced VCD file format.
- The conversion utility may not match the activity file format type or version. PrimePower invokes the appropriate utilities to convert the activity data to VCD where necessary. The utility must be in the users search path and match the version of the data.

Power Calculation

When the activity file provided includes time-based information, PrimePower performs event-based analysis when the `calculate_power` command is issued. It analyzes the energy for every event to calculate the average power for every instance in the design. Users can also generate power waveforms and view the peak power with the appropriate options.

The following sections describe the variables and command options which affect the power calculation.

Variables for Power Calculation

```
pwr_table_internal_energy_only = "true"
```

The `pwr_table_internal_energy_only` variable indicates whether power tables in the technology library should include switching power or not.

By default the value is set to `true`. PrimePower assumes the library power tables contain only the internal energy, which is calculated in the following manner during characterization:

Formula 1

```
Internal_energy_rise =  
    Total_energy_rise - leakage_energy - switching_energy  
Where the switching_energy is  $CV^2$ 
```

Formula 2

```
Internal_energy_fall =  
    Total_energy_fall - leakage_energy
```

When the variable is set to `false`, PrimePower assumes the library internal energy tables contain values in the form of

Formula 3

```
Internal_energy_rise =  
    Total_energy_rise - leakage_energy - 0.5  
switching_energy
```

Formula 4

```
Internal_energy_fall =  
    Total_energy_fall - leakage_energy - 0.5  
switching_energy
```

Where the switching_energy is CV^2

Since switching energy (CV^2) is incurred only on the rising edge; the tables values in this case do not reflect the true internal energy.

$$E_{int\_table}(rise) = E_{int}(rise) + 1/2CV^2$$
$$E_{int\_table}(fall) = E_{int}(fall) - 1/2CV^2$$

The variable does not affect the average power as multiple events even out the difference, but it does affect the power waveforms particularly for small designs and vector sets. For optimal peak power analysis, set this variable correctly.

The best way of finding out the formula used in the characterization is to consult the library vendor. A workaround is to inspect the falling power table. If most of them have negative numbers, it is likely that the formulas 3 and 4 are used in characterization.

The examples below describes, in detail, how the variable affects PrimePower calculating total power consumption of each toggle.

When the `pwr_table_internal_energy` variable is set to `false`, PrimePower assumes formulas 3 and 4 were used for characterization.

Formula 1

$$\begin{aligned} \text{total\_energy\_rise} \\ = \text{internal\_energy\_rise} + \text{leakage\_energy} + \text{switching\_energy} \end{aligned}$$

Formula 2

$$\begin{aligned} \text{total\_energy\_fall} \\ = \text{internal\_energy\_fall} + \text{leakage\_energy} \end{aligned}$$

Formula 3

```
total_energy_rise
    = internal_energy_rise + leakage_energy + 0.5
switching_energy
```

Formula 4

```
total_energy_fall
    = internal_energy_fall + leakage_energy + 0.5
switching_energy
```

When the `pwr_table_internal_energy` variable is set to `true`, PrimePower assumes formulas 1 and 2 were used for characterization.

Event Based Initial X-State Handling

When processing VCD data, PrimePower processes every event. Many VCD files contain X-states at initialization when the input values are being propagated.

The variable `pwr_count_initial_xpower true|false` (default is `false`) determines whether PrimePower calculates the power during initialization or not.

By default, the variable is set to `false` and PrimePower ignores the switching and internal power consumed by the transition of nets from an unknown to a known state. If however, users are concerned with the power at initialization, the value can be set to `true` and the power peak produced as the circuit initializes can be included in the power analysis.

Unmatched-States

When PrimePower processes the events in the activity file, events for which there is no matching condition in power table may occur. To view the percentage of matching tables, users must apply the `–statistics` option to the `calculate_power` command. In this case PrimePower outputs statistical information which includes:

```
X % tables on output pins matched
X % tables on input  pins matched
```

Most cells do not have input-pin based tables resulting in 100% pin matching. However, for output cells, if the event does not match any of the state definitions in the power table, the event detracts from the percentage of tables on output pins which matched. For these events, by default, PrimePower assumes no power is consumed and ignores the event.

Users however, can modify the value of the variable

```
pwr_estimate_power_for_unmatched_event true|false (default
false)
```

to instruct PrimePower to come up with an estimate of the power consumption for that event based on the average value of all the available tables for that output pin.

Using the `calculate_power` Command Options

The `calculate_power` command provides users with multiple options to control the time interval, specific instances, and the type of information to store for reporting purposes. These options are described below:

Default Behavior

By default, the `calculate_power` command calculates the energy for every event within the simulation activity file, and determines the average power consumption for every instance within the design. The results are stored as attributes and can be viewed within the PrimePower GUI or via the reporting commands described in the Power Reporting chapter.

Time/Instance/Mode Options

If the user is interested in only particular instances or in particular time windows; the following options are available to allow the user to focus on only specific portions of the design or specific segments of the simulation:

- `-instance {list}` allows the user to specify the desired instances

Example: `calculate_power -instance {U1 U2}`

- `-time {list}` allows the user to specify the simulation time windows for which power is to be calculated. If the end of simulation time is unknown, users can specify a time value of "-1" to denote the last event in the simulation activity file.

Example: `calculate_power -time {100 -1}`

This informs PrimePower to calculate the power for the design starting at time 100 until the end of the simulation activity file.

- `-mode {expression}` allows the user to describe a boolean expression during which power is to be calculated.

Checking for Out-of-Range Loads and Transition Times

If the design contains high-fanout nets with large loads, this can result in transition times which are beyond the range specified in the library power models. By default, PrimePower extrapolates the energy value when the indexes are out of range which results in very high power consumption for instances in the high-fanout network.

To verify that the loads and transition times are within the range specified in the library model users can apply the following `calculate_power` options:

- `-statistics`

This option causes PrimePower to output the following statistical information:

Table 9-1 Output of statistical information

Information	Description
Total number of 2-input equivalent gates	based on 3 pins/gate
Total number of states	# of transitions
Found X% x states	% X states
Found X% glitch states	% glitch transitions
Found X% tables on input pins matched	% of input transition tables that matched
Found X% tables on output pins matched	% of output transitions with matching tables

Table 9-1 Output of statistical information (Continued)

Information	Description
Found X% "out of range" ramp violations	% of out of range ramps
Found X% "out of range" load violations	% of out of range loads

Ideally, there should be no out of range ramp and load violations and all pins should have matching tables. Namely, the % out of range violations should be 0 and the % tables on pins matched should be 100.

If the user has an out of range ramp and load violations; the following options can be specified to generate more detailed information regarding the violating instances and how far out of range the values are.

- `-verbose`

This option causes PrimePower to generate a file `primepower.err` (by default) which lists all the pins whose indexes are out of the power table range.

- `-err <filename>`

Shown below is a sample `primepower.err` file:

```
a_lm_reg[0] (ldf1a6) had ramp violation with rise ramp =
0.511479,
fall ramp = 0.298800
a_lm_reg[10] (ldf1a6) had ramp violation with rise ramp
= 0.298800,
fall ramp = 0.511479
U0_1_11 (or2c1) had load violation with load = 0.021120
U0_1_13 (or2c1) had load violation with load = 0.022120
```

To rename the file, users can optionally specify the `-err` `<filename>` option to generate a file with a different name.

Interrupting and Resuming Power Calculation

When performing event-based calculation, PrimePower allows users to interrupt the power analysis. Users can view the results up to that point or apply the CPU resources to other jobs before resuming the power analysis.

To interrupt the simulation, users can issue a Ctrl-C after the `calculate_power` command has been issued and processing of VCD data has begun.

Note:

If Ctrl-C is issued during update timing, it will stop the process, and the user cannot resume. Ctrl-C must be issued once the processing of the time stamps has begun.

To interrupt power calculation, the user must issue the command:
`resume_power_calculation [-force]`

Example:

```
calculate_power
```

Ctrl-C

```
report_power -file interrupt1  
save_session interrupt1  
resume_power_calculation
```

The total power calculated once the analysis is complete, includes the power analysis before and after the interruption.

If the user modifies any of the input options prior to the resumption of the power analysis, he must issue the `-force` option to continue power calculation. By default, if the parasitics, timing etc. is modified before resuming, power analysis will not be resumed.

Example:

```
calculate_power
```

Ctrl-C

```
set_load 0.0 *  
resume_power_calculation -force
```

If the user has selected `calculate_power -waveform`, then the waveform data prior to the interrupt is stored separately from the data generated after the interrupt. The waveform files are given `_1`, `_2` etc. extensions.

Note:

Ctrl-C cannot be used to interrupt the analysis if the `pp_shell` command was issued with a pipe to the `tee` command:

Example: `pp_shell -f pp_vcd.tcl | tee pp_vcd.log`

If the user issues Ctrl-C, the `tee` will quit and the user cannot resume the analysis.

Power Waveforms

To save the power values over time, users can apply the `-waveform` option to the `calculate_power` command. In this case, PrimePower stores the power for every event within a given time interval (default 1 ns) in a waveform file which can be viewed within turboWave.

Listed below are the default waveform options:

- `format: fsdb`
- `filename: primepower.fsdb`
- `interval: 1 ns`
- `hierarchy : (all except leaf level)`

To modify the default options, users can apply the `set_waveform_options` command prior to `calculate_power`.

The syntax is shown below:

```
set_waveform_options [-file filename] [-leaf] [-hier level]
[-format fsdb | out | rpt ] [-interval sampling_period]
-effort high | low
```

Peak Power

In order to store or view the peak power consumption, the `-waveform` option must be applied to the `calculate_power` command. This command causes PrimePower to store the power for every event within a given time interval (default 1 ns) in the waveform file. The largest value and the time at which it occurred is included in the report file generated with the `report_power` command.

Users interested in only the peak power can limit the amount of waveform data saved by setting the waveform option: `-hier 1`. In this case, only the waveform data for the top level hierarchy is saved, but the peak power for the complete hierarchy is reported in the PrimePower report file when the `report_power` command is issued.

Another option is to apply the `-format rpt` option to the `set_waveform_options` command. In this case, no waveform data is generated, but peak values are included in the power report.

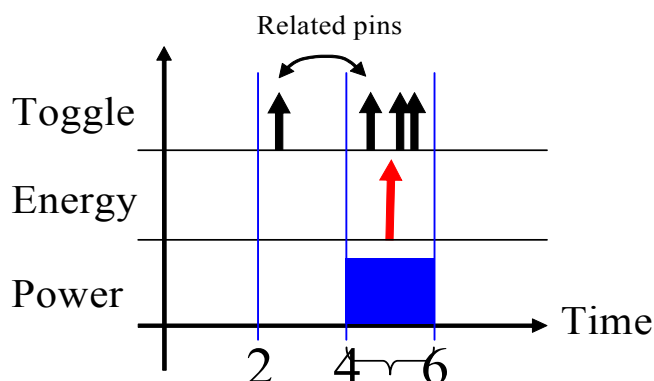
Options for Power Waveform Calculation Algorithm

PrimePower provides two algorithms for calculating power waveforms. You can select the desired algorithm with the `-effort` option. If not specified, PrimePower applies the `-effort low` option, which invokes the default algorithm that relies heavily on the interval. It is, however, recommended that you select the `effort -high` option for more accurate peak power.

The default algorithm sums the energy consumed by every event within a specific interval and averages it out over the interval time. Larger intervals result in smaller peaks while smaller intervals result in larger peaks. [Figure 9-1](#) shows the behavior for low-effort peak power analysis, given:

```
set_waveform_options -interval 2
update_power -waveform
```

Figure 9-1 Low-Effort Peak Power Analysis



The energy of all the events within every interval is summed and divided by the time interval. The waveform therefore shows the average power of each interval. The larger the interval, the smaller the peak, and vice versa.

The algorithm invoked with the `-effort high` option causes PrimePower to average the energy for every event across the path delay from the related input to the output, and across the input transition time for input pin power. By default, the interval is derived from the VCD time scale so that the path delay can be accurately averaged.

Essentially, PrimePower divides path delay into intervals corresponding to the VCD resolution. The power consumption for the transition is averaged across these intervals.

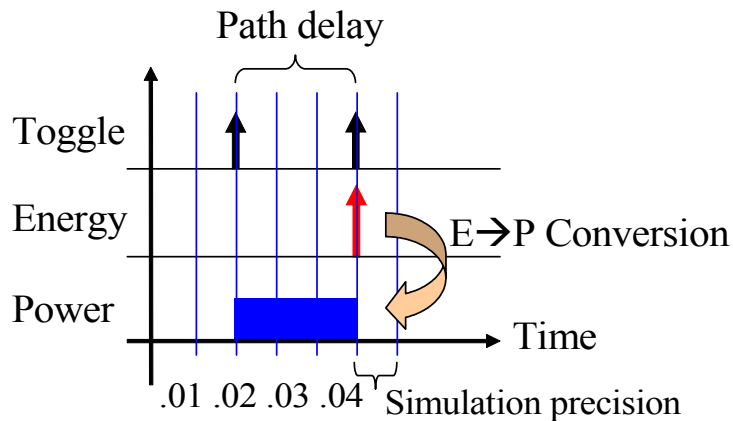
You can override the default interval with the `-interval` option, but this is not recommended.

Figure 9-2 shows the behavior for high-effort peak power analysis, given:

```
$timescale 10ps
```

where the resolution is derived from the VCD file.

Figure 9-2 High-Effort Peak Power Analysis



The energy for an event is distributed evenly over the path delay or transition time for the input power pin. Instead of averaging it across the interval in which the output event occurred, the energy is distributed over the path delay that better mimics the true behavior.

10

Generating Reports

PrimePower can generate a wide range of reports that provide information about the power consumption.

Overview

The following sections describe the mechanisms for generating power reports:

- [Default Power Report](#)
- [New Power Report](#)
- [Optional Reports](#)
- [PrimeRail Reports](#)
- [Custom Report Generation](#)

- **Statistics**

The primary command for generating power dissipation reports is `report_power`. Once the user has successfully performed the power analysis with the `calculate_power` command, the `report_power` command is issued to view the results. Only the time and instances specified in the `calculate_power` command can be reported.

Shown below is the syntax for the `report_power` command:

```
report_power [-file file_prefix]
              [-instance list]
              [-threshold value]
              [-leaf]
              [-hier level_number]
              [-cell]
              [-net]
              [-input_net]
              [-sortby sort_type]
              [-nodisplay]
```

Default Power Report

By default, the `report_power` command outputs the power consumption of all the instances in the design (except for the leaf-level) to standard out as well as to the default file:

`primepower.rpt`.

The report file can also provide power dissipation data (power = energy/time) for the desired instances by using the `report_power -instance` option.

Typically, you use the power dissipation report to identify general problem areas in a design. You can narrow your analysis using PrimePower's other options.

By default, the power dissipation report includes eight types of power data:

- Total power—total event-based and leakage power
- Event-based power—total power consumption due to event-based sources such as internal power, switched capacitances, glitches, and X-transitions
- Leakage power—reverse-biased junction leakage and subthreshold leakage
- Switching power—power dissipated into charging and discharging capacitive loads during logic change
- Internal power—event-based power consumed inside a cell
- X-state transition power—power dissipated when the signal changes from or to the X-state
- Glitch power—power dissipated into detectable glitches at the nets
- Logic states—total number of states per instance

The following example shows a sample power dissipation report.

```

                                PrimePower (TM)
Version X-2005.06 for sparcOS5 -- May 22, 2005
Copyright (c) 2000-2005 by Synopsys, Inc.
                                ALL RIGHTS RESERVED

*****
Report:                          average power
Design:                          mac
Library File(s):                 ./lib/snps/core_typ.db
Clock Name: default  Clock Period: -1ns
Reporting Threshold: 0 Watt
Date:                            May 22, 2005
*****
```


Leakage Power = reverse-biased junction leakage +
subthreshold leakage

Level	Instance_Name (Cell_Name)				#_of_States	
Total	Dynamic	Leakage	Switching	Internal	X-	
tran	Glitch					
Power	Power	Power	Power	Power	Power	Power
in Watt	in Watt	in Watt	in Watt	in Watt	in Watt	in
Watt	in Watt					
	(% of Tot)	(% of Tot)	(% of Dyn)	(% of Dyn)	(% of Dyn)	(%
of Dyn)	(% of Dyn)					
----- Instances -----						
*-----	0	pp_root (.)	1353917			
2.804e-03	2.804e-03	2.594e-07	1.054e-03	1.750e-03		
0.000e+00	0.000e+00					
	(99.99%)	(0.01%)	(37.59%)	(62.41%)	(	
0.00%)	(0.00%)					
-*-----	1	mac (mac)	1353917			
2.804e-03	2.804e-03	2.594e-07	1.054e-03	1.750e-03		
0.000e+00	0.000e+00					
	(99.99%)	(0.01%)	(37.59%)	(62.41%)	(	
0.00%)	(0.00%)					
--*-----	2	mac/mult_21 (mac_DW02_mult_16_16_0)	528317			
8.582e-04	8.580e-04	1.491e-07	4.595e-04	3.986e-04		
0.000e+00	0.000e+00					
	(99.98%)	(0.02%)	(53.55%)	(46.45%)	(	
0.00%)	(0.00%)					
---*-----	3	mac/mult_21/U1/U9720 (mac_DW01_add_25_0)				
110743						
1.255e-04	1.255e-04	1.605e-08	8.423e-05	4.126e-05		
0.000e+00	0.000e+00					
	(99.99%)	(0.01%)	(67.12%)	(32.88%)	(	
0.00%)	(0.00%)					
--*-----	2	mac/add_23 (mac_DW01_add_33_0)	288202			
3.386e-04	3.386e-04	2.362e-08	2.174e-04	1.212e-04		
0.000e+00	0.000e+00					
	(99.99%)	(0.01%)	(64.20%)	(35.80%)	(	
0.00%)	(0.00%)					
----- End of report -----						

New Power Report

In version X-2005-06, users can optionally produce power reports in a format compatible with that produced by Power Compiler. The new format is enabled in the following ways:

1. The user can set the variable `pwr_report_power_format` to `new`. By default, the value is set to `old`. When set to `new`, the output of the `report_power` command will be in the new format.
2. The user can issue the `report_power_new` command. Shown below are the options to the `report_power_new` command.

```
report_power [-net] [ -cell]
              [-only cell_or_net_list]
              [-hier] [-hier_level level_value]
              [-flat] [-threshold thresh_value]
              [-include_input_nets]
              [-nworst number] [-sort_mode mode]
              [-histogram [-exclude_leq le_val |
                           -exclude_geq ge_val]]
              [-categorized]
              [-related_clocks clock_list]
              [-nosplit]
```

By default the new format displays only the top-level power consumption. Shown below is a sample report generated with the `report_power new.rpt` command:

Figure 10-1 Sample report generated from the `report_power_new` command

```
*****
Report : power
Design : cntrl6
Version: X-2005.06-Beta2
Date   : Fri May 20 11:33:01 2005
*****
Simulation Periods: 75 ns - 3375 ns

Sampling Interval: 0.001 ns

Library(s) Used:
mylib.db (File: /usr/mylib.db)
Operating Conditions: nom Library: mylib.db
Wire Loading Model Mode: top
Wire Load Model: mywlm
<no wire load model is set>

Power-specific unit information :
  Voltage Units = 1 V
  Capacitance Units = 1 pf
  Time Units = 1 ns
  Dynamic Power Units = 1 W
  Leakage Power Units = 1 W

Cell Internal Power = 3.044e-06 (32%)
Net Switching Power = 6.480e-06 (68%)
-----
Total Dynamic Power = 9.523e-06 (100%)
Cell Leakage Power  = 4.383e-07
```

The `report_power_new` options can be used to view additional information. Shown below is a sample `report_power -hier` output.

Figure 10-2 Sample report generated from the `report_power_new -hier` command

Hierarchy	Power	Power	Power	Power	%
cntr16	6.48e-06	3.04e-06	4.38e-07	9.96e-06	100.0
add_13 (cntr16_DW01_inc_16_0)	1.42e-06	2.73e-07	2.26e-07	1.92e-06	19.3
Hierarchy	Peak Power	Peak Time	Glitch Power	X-tran Power	
cntr16	2.32e-02	2412.570-2412.571			
add_13 (cntr16_DW01_inc_16_0)	1.67e-02	1200.142-1200.143	0.000	0.000	
			0.000	0.000	

Optional Reports

There are several types of optional reports that the `report_power` command can produce. These reports include

- Cell report
- Leaf report
- Waveform report
- Net report

Cell-type Power Dissipation Report

The cell-type power report is appended to the default power report (`*.rpt`). It provides the same type of data as the default power dissipation report, except that it displays information by cell type instead of by instance. This extra content is produced only when the `-cell` option is specified in the `report_power` command.

```
report_power -cell
```

Figure 10-3 shows a sample cell-type power report.

Figure 10-3 Sample cell-type power report

Cell_Name						

Total Power in Watt	Dynamic Power in Watt (% of Tot)	Leakage Power in Watt (% of Tot)	Switching Power in Watt (% of Dyn)	Internal Power in Watt (% of Dyn)	X-tran Power in Watt (% of Dyn)	Glitch Power in Watt (% of Dyn)

xor_gate						
1.9836e-05	1.9836e-05 (100.00%)	2.4384e-19 (0.00%)	9.6221e-06 (48.51%)	1.0214e-05 (51.49%)	1.3089e-06 (6.60%)	0.0000e+00 (0.00%)
or_gate						
4.1711e-06	4.1711e-06 (100.00%)	9.0720e-20 (0.00%)	2.7902e-06 (66.89%)	1.3809e-06 (33.11%)	6.2091e-07 (14.89%)	0.0000e+00 (0.00%)
and_gate						
2.1404e-06	2.1404e-06 (100.00%)	4.9920e-20 (0.00%)	8.9838e-07 (41.97%)	1.2420e-06 (58.03%)	5.0078e-07 (23.40%)	0.0000e+00 (0.00%)

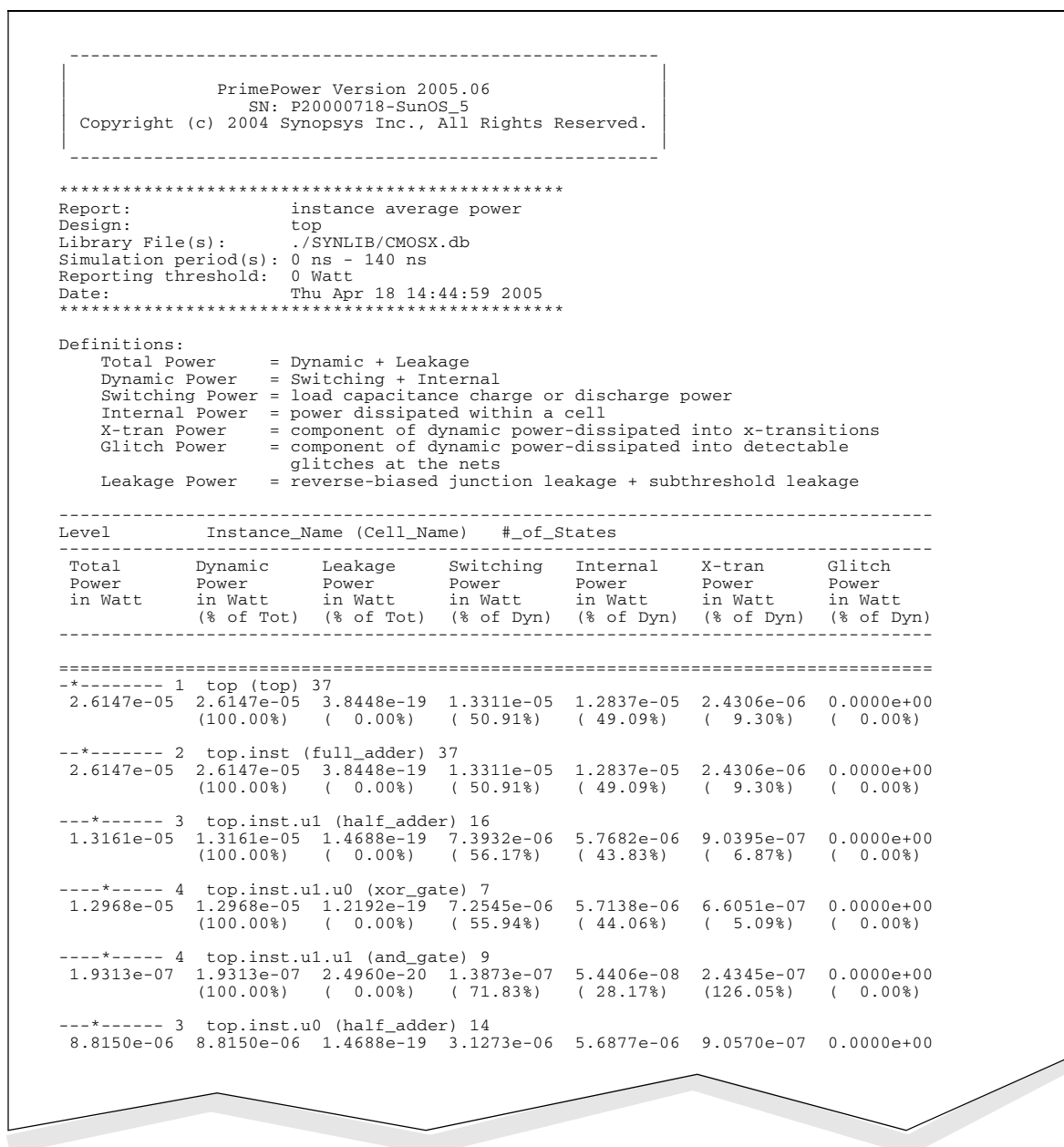
In the above example, the power values represent the sum of all the xor_gate, or_gate, and and_gates in the design.

Leaf Power Dissipation Report

The Power Dissipation Report (*.rpt) shows leaf instance power dissipation, in addition to power dissipation for instances. You obtain this information by specifying the `-leaf` option in the `report_power` command.

```
report_power -leaf
```

Figure 10-4 Sample leaf power report



Reporting to the leaf level is useful particularly when leaf cells are scattered throughout the hierarchy or when performing detailed analysis.

Net Power Report

Shown below is a portion of a net switching power report generated using SAIF annotation. When VCD (event based activity) is applied, no attributes are specified.

Attributes

- a - Switching activity information annotated on net.
- d - Default or propagated switching activity information on net.

Net	Total Net Load in pF	Static Prob.	Toggle Count per ns	Switching Power in Watt	Attrs
Logic0	0.000	0.000	0.0000	0.0000	a
z[32]	0.003	0.350	0.0026	1.066e-08	a
z[31]	0.003	0.526	0.0033	1.795e-08	a
z[30]	0.003	0.507	0.0027	1.152e-08	a
z[29]	0.000	0.448	0.0054	3.700e-09	a
z[28]	0.002	0.513	0.0110	3.920e-08	a
z[27]	0.005	0.521	0.0217	1.744e-07	a
z[26]	0.001	0.509	0.0435	5.452e-08	a
z[25]	0.001	0.519	0.0480	4.026e-08	a
z[24]	0.001	0.490	0.0356	6.600e-08	a
z[23]	0.005	0.511	0.0460	4.051e-07	a
z[22]	0.001	0.497	0.0414	3.481e-08	a
z[21]	0.006	0.483	0.0489	4.530e-07	a
z[20]	0.000	0.514	0.0377	1.600e-08	a
z[19]	0.001	0.490	0.0466	3.908e-08	a
z[18]	0.000	0.494	0.0440	1.866e-08	a
z[17]	0.007	0.508	0.0487	5.879e-07	a
z[16]	0.005	0.483	0.0377	3.338e-07	a
z[15]	0.013	0.500	0.0491	1.005e-06	a
z[14]	0.001	0.496	0.0390	4.184e-08	a
z[13]	0.003	0.513	0.0496	2.163e-07	a
z[12]	0.001	0.501	0.0352	3.684e-08	a
z[11]	0.001	0.497	0.0425	6.287e-08	a
z[10]	0.008	0.476	0.0563	6.934e-07	a
z[9]	0.015	0.475	0.0291	7.066e-07	a
z[8]	0.010	0.477	0.0296	4.619e-07	a
z[7]	0.006	0.534	0.0293	2.630e-07	a

z[6]	0.013	0.540	0.0297	6.052e-07	a
z[5]	0.021	0.505	0.0307	1.040e-06	a
z[4]	0.021	0.415	0.0320	1.084e-06	a
z[3]	0.014	0.500	0.0352	7.765e-07	a
z[2]	0.014	0.326	0.0416	9.244e-07	a
z[1]	0.013	0.503	0.0831	1.749e-06	a

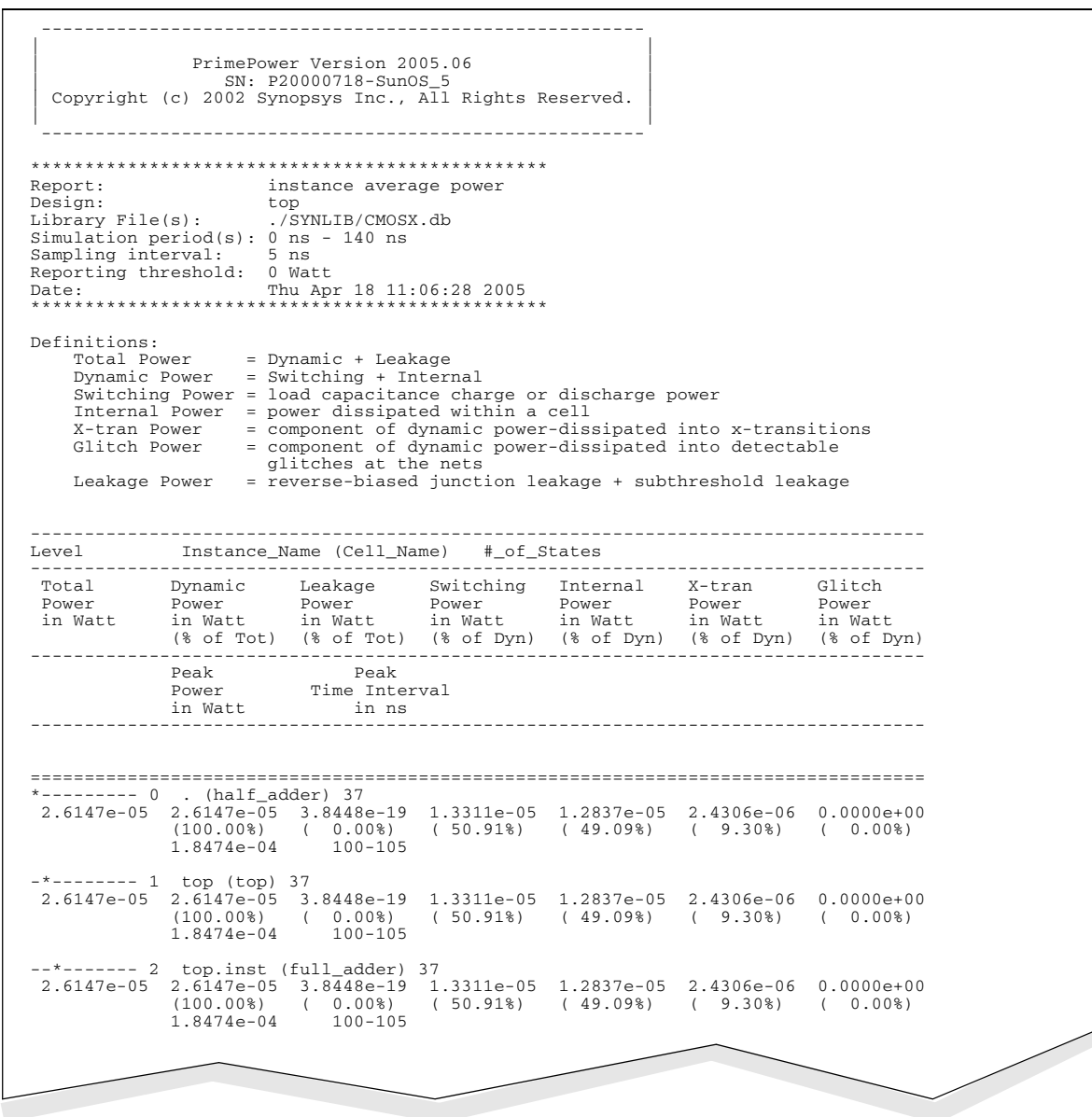
By default, PrimePower does not include primary input net power. The switching power of primary inputs can be viewed by specifying the `-input_net` option to the `report_power` command. Their power consumption is listed separately and is not included in the total power.

Waveform Report

The Power Versus Time Report (`*.out` or `*.fsdb`) shows instance-by-instance power consumption during specified simulation time intervals, and the Power Dissipation Report (`*.rpt`) shows peak power for each instance among these sampling intervals. Use this report to pinpoint problems during particular time segments within each instance of a design.

The `*.out` file, `*.fsdb` files, and the peak power information are produced only if you specify the `-waveform` option for the `calculate_power` command. [Figure 10-5](#) shows a sample power dissipation report with peak power.

Figure 10-5 Sample power dissipation report with peak power



Viewing Waveform Data

You can view waveform data using turboWave. Enter the following command at the UNIX command line to invoke turboWave:

```
%turboWave
```

To view the waveform data in turboWave:

Choose *File > Open* and select the desired file.

This loads the desired *.fsdb* or *.out* file. You can select the desired signals from within turboWave by choosing the *Signal > Get All Signals* from the menu bar.

Scaling Waveform Data

By default, turboWave scales all the individual waveforms to a default height. To view the waveforms scaled to the total power consumption do the following:

1. Select all the signals in the Signal Column.
2. Choose *Analog > Zoom Value* from the turboWave menu bar.
3. On the form that appears, select Full and then Close.

Clock-related Power

PrimePower provides commands which allow the user to view the clock tree and connect register power. If the clock tree has not been inserted, users can apply the following command to estimate its power consumption:

```
estimate_clock_tree_power
```

If it has been inserted, PrimePower provides the following commands to report the clock tree and register power consumption.

- `report_clock_tree_power`
- `report_register_power`
- `report_power -related_clock`

These commands are available provided the user first defines the clocks in the design with the `create_clock` command.

Example: `create_clock -period 1 clk`

The `create_clock` command requires the user specify a period. This period is not used for power calculation if the clock activity is provided in a SAIF or VCD file.

Clock Tree Estimation

PrimePower now supports the ability to estimate the additional power incurred by a clock tree prior to its insertion. This feature is enabled with a new command, `estimate_clock_tree_power`, which provides users with more accurate analysis of the total design power when clock tree synthesis has not yet been performed. Prior to issuing the `estimate_clock_tree` command, the you must indentify the design clocks.

You can perform clock tree estimation before power calculation when you have statistical activity (either via SAIF or the `set_switching_activity` command). When VCD data is provided, however, the `estimate_clock_tree_power` command must be issued after the `calculate_power` command. Internally, the VCD activity on the clock tree is converted to SAIF and used to estimate the clock tree power.

Based on user constraints and buffer-type specification, PrimePower builds a virtual balanced clock tree and calculates its power consumption. The following guidelines are used to build the clock-tree:

- The delay from the root to the leaf of the tree is minimized.
- The driven registers are put at the same and lowest level of the tree.
- The deviation of buffer fanouts at the same tree level is minimized.
- For clock-gating cells, separate trees are built based on the fanout of the clock-gating cell using the same rules as specified above.

The output load of each buffer is calculated using the wire load model; and the input transition of each buffer is propagated from the root. The switching activity is derived from the clock specification or from the SAIF file.

The output of the command includes the power consumption as well as detailed information on the pseudo-clock tree.

Command Syntax .

Described below is the command options which have been added to PrimePower to support the clock-tree estimation project.

```
estimate_clock_tree_power
```

```
-buffer <lib_cell>
```

This option specifies the library cell to be used as the insertion buffer.

```
-max_fanout <fanout>
```

By default the value is 32; but users can modify the maximum fanout.

```
-wlm_lib <library_name>
```

Specifies the library from which to obtain the specified wire_load_model.

```
-wlm <wire_load_model>
```

Users can select the desired wire_load_model. Otherwise, the tool applies the default library wire_load_model.

```
-input_transition <ramp>
```

By default, the value is derived from the set_clock_transition command. Users can override it with this option.

```
-clock <clock>
```

Specifies the clock for which to estimate the power. If not specified all clocks will be assumed.

```
-include_register
```

Informs the tool to include the driven registers in the power analysis.

`-ignore_clock_gating`

Informs the tool to ignore clock-gating cells when building the clock tree and estimating the power.

Command Output.

Shown below is a sample output of the `estimate_clock_tree_power` command with the following options: `-buffer [get_lib_cell mylib.db/BUF1]` `-max_fanout 4`. Detailed information regarding the clocks, driven registers, and timing propagation is provided as well as the power.

```
*****
Report : estimated clock tree power
Design : reg16
Version: X-2005.06
Date    : Tue May 10 08:41:09 2005
*****

Library(s) Used:
  mylib.db (File: /usr/mylib.db)
Operating Conditions: <lib_default>      Library:mylib.db
Wire Load Model: wlm1

Buffer Used:      BUF1      Library:      mylib.db
Buffer Max Fanout: 4

Power-specific Unit Information:
  Voltage Units = 1V
  Capacitance Units = 1pf
  Time Units = 1ns
  Power Units = 1W

CLOCK: clk1      (source: clk1)
-----
-----

      Clock Period:      20.000000
```


Clock Toggle Rate: 0.100000
Clock Static Prob: 50.000000%

Operating Voltage: 5.000000
Clock Input Transition: 0.000000 (rise) 0.000000 (fall)

Number of Driven Regs: 16
Number of Existing CT Cells: 0
Number of Buffer Inserted: 5
Depth of Clock Tree: 2

Clock Tree Latency		min	ave	max

	rise	0.0654	0.0654	0.0654
	fall	0.0648	0.0648	0.0648
Clock Tree Output Transition		min	ave	max

	rise	0.0205	0.0205	0.0205
	fall	0.0139	0.0139	0.0139

Power Estimation:

Cell Internal Power	=	5.855e-06
Net Switching Power	=	1.120e-04

Total Dynamic Power	=	1.179e-04
Cell Leakage Power	=	3.965e-08

Clock Tree Power

The `report_clock_tree_power` command returns the power consumption of the clock tree. By default, it returns the power consumption of all the clock elements in the design.

Users with multiple clocks can report the clock tree power for each clock separately.

Examples include

- `create_clock -period 1 clk1`
- `create_clock -period 1 clk2`
- `report_clock_tree_power -clock clk1 > clk1.rpt`
- `report_clock_tree_power -clock clk2 > clk2.rpt`

In this case, the clock tree power for each clock is output to a separate file.

Register Power

`report_register_power`

To obtain the power consumption of the registers, use this command.

To obtain a report for the registers clocked by a particular clock, use the

`-clock (name)` option.

Related Clock Power

The option, `-related_clocks`, generates a clock-related power report. The related clock is selected based on the clock domain the object belongs to.

Scan Chain Power

The following command is used to report the power consumption of the scan registers and buffers or inverters connected to scan enable pins.

```
report_scan_chain_power <scan enable pins>
```

Note:

The design must contain the test-mode data and the VCD file should include the activity which has test mode operations.

PrimeRail Reports

PrimePower is used within PrimeRail to calculate power. The PrimePower output is used to perform IR analysis in PrimeRail.

The following commands (currently hidden) are used to generate PrimeRail reports:

- VCD event-based analysis

```
calculate_power_rail <filename>
```

- Statistical activity based analysis

```
write_rail <filename>
```

Custom Report Generation

PrimePower now provides a host of attributes which reference power information within the design and the libraries. [Table 10-1 on page 10-22](#) shows the attribute and the object type to which it can be assigned.

These attributes are accessible only after the `calculate_power` command has been issued.

Table 10-1 Power Attributes

Class	Attribute
net	pp_toggle_count pp_power pp_glitch_rate_value pp_glitch_rate_assert_flag pp_toggle_rate_value pp_toggle_rate_assert_flag pp_signal_probability_value pp_signal_probability_assert_flag pp_net_power related_clock
pin	pp_glitch_rate_assert_flag pp_glitch_rate_value pp_signal_probability_assert_flag pp_toggle_rate_value pp_toggle_rate_assert_flag related_clock
cell (**)	pp_number_of_states pp_total_power pp_switching_power pp_internal_power pp_dynamic_power pp_xtran_power pp_glitch_power pp_glitch_rate_value pp_glitch_rate_assert_flag pp_leakage_power pp_peak_power (***) pp_peak_start_time (****, *****) pp_peak_end_time (****, *****)

Table 10-1 Power Attributes (Continued)

Class	Attribute
design (**)	pp_number_of_states pp_total_power pp_switching_power pp_internal_power pp_dynamic_power pp_xtran_power pp_glitch_power pp_glitch_rate_value pp_glitch_rate_assert_flag pp_leakage_power pp_peak_power (****) pp_peak_start_time (****, *****) pp_peak_end_time (****, *****)
lib_cell (*****)	pp_total_power pp_switching_power pp_internal_power pp_dynamic_power pp_xtran_power pp_glitch_power pp_leakage_power
Note	
**	Only monitored cells have power attributes. That is, only the cells that have been calculated power for have power attributes.
***	Only current design has power attributes. Corresponding to root_cell. Current design has power attributes only when root_cell is monitored (default).

Table 10-1 Power Attributes (Continued)

Class	Attribute
****	pp_peak_xxx attributes only available when -waveform is specified.
*****	peak time interval = {pp_peak_start_time, pp_peak_end_time}
*****	Only instantiated lib_cells have power attributes.

Users can write their own procedures and rely on the built-in object collections to access the desired information.

Shown below is a sample procedure, which outputs a custom report for power consumption of all registers in the design.

```

proc report_register_power {} {
    set cells [all_registers]
    if { [sizeof_collection $cells] == 0 } {
        echo "Error: cannot find any registers.\n"
    } else {
        set t_total      0.0
        set t_dynamic    0.0
        set t_leakage    0.0
        set t_switching  0.0
        set t_internal   0.0
    }
    # print the header
    echo "*****"
    echo "          Register Power Report          *"
    echo "*****\n"
    echo [format "%10s %10s %10s %10s %10s %s" \
        "Total" "Dynamic" "Leakage" "Switching" "Internal"
        "Register Cell"]
    echo "-----"
    # print the body
    foreach_in_collection cell $cells {
        set name      [get_object_name $cell]
        set total     [get_attribute $cell pp_total_power]
        set dynamic   [get_attribute $cell pp_dynamic_power]
    }
}

```

```

        set leakage [get_attribute $cell pp_leakage_power ]
        set switching [get_attribute $cell pp_switching_power]
        set internal [get_attribute $cell pp_internal_power]
        echo [format "%10.3e %10.3e %10.3e %10.3e %10.3e %s" \
            $total $dynamic $leakage $switching $internal $name]
        set t_total      [expr $t_total + $total]
        set t_dynamic    [expr $t_dynamic + $dynamic]
        set t_leakage    [expr $t_leakage + $leakage]
        set t_switching  [expr $t_switching + $switching]
        set t_internal   [expr $t_internal + $internal]
    }
#    print the total
echo "-----"
    echo [format "%10.3e %10.3e %10.3e %10.3e %10.3e TOTAL" \
        $t_total $t_dynamic $t_leakage $t_switching
        $t_internal]
    }
}
-----

```

Statistics

To isolate the instances which contribute to out of range load and ramp violations; the user can rerun PrimePower with the `calculate_power \`
`-statistics -verbose` option.

In this case, an ASCII `primepower.err` file will be generated which lists all the instances which are out of range; and the time at which the library access was made.

Shown below is a sample `primepower.err` file

```
cm02 (iv22) had ramp violation with rise ramp + 30.893909,  
fall  
ramp = 2.546273  
cm01 (iv22) had ramp violation with rise ramp + 00.000000,  
fall  
ramp = 0.000000  
cm01 (iv22) had ramp violation with load = 5.012000
```

The file list the rise and fall ramps and load values which are out of range.

Values which are out of range cause PrimePower to extrapolate and could lead to inaccuracy.

11

Multi-rail Power Analysis

PrimePower provides commands which enable the user to identify the power consumption of the design per rail. By default, PrimePower reports the total power consumption. In order to report power per rail, PrimePower needs to know the power rails in the design as well as the rails in the library. As of the 2005.06 release, PrimePower allows users to group cells based on the design power rails. Users can report the power consumption per design rail, rather than per library rail. In addition, PrimePower now supports power analysis of designs that include modules that are powered on and off. Rather than assuming the power supply to the cells is always on, PrimePower now takes into account the power control signal for accurate analysis of power down modes.

This chapter describes the power analysis of a multi-power rail design. It covers the following topics:

- [Multi-voltage Design Power Report](#)

- Multi-voltage Cells
- Reporting Power Rail Mapping
- Power-gating Support
- Sample Script
- Dealing with the Multi-voltage Libraries

Multi-voltage Design Power Report

To report power by rails, the cells in a design should be grouped by the power rails. For multi-voltage cells; such as level shifters, the library power rails need to be linked to the design power rails.

The following is an example of grouping cells:

```
create_power_rail_mapping V1 -instance "G1 G2 G3" \  
create_power_rail_mapping V2 -instance "B1 B2" \  
create_power_rail_mapping V3 -instance "R1
```

In the example above, the G1 refers to one of the green instances and R1 refers to the red cell. Other instances are named in the same fashion. V1 is one power rail in the design.

Once the design is grouped, the power report will cover only the power of the selected rails. Here is a simple example of an MVdd design analysis:

```
link  
create_power_rail_mapping \  
    V1 -instance "G1 G2 G3" \  
set_current_rail "V1"  
calculate_power  
report_power
```

In the above example, the `set_current_rail` command, allows you to select the rails for which to report power. You can select one rail or a list of power rails. PrimePower only reports the sum of the selected power rails. Without this command, all the rails' power will be reported together. This command only affects the `calculate_power` command and must be used before it.

Shown below are examples of how to use the `set_current_rail` command to view the power consumption per rail for a design which contains two power rails: VDD1 and VDD2:

```
Example: set_current_rail VDD1
        calculate_power
        report_power -file pp_rail_VDD1
        set_current_rail VDD2
        report_power -file pp_rail_VDD2
```

To calculate the power consumption per rail, users must issue the `calculate_power` command after each `set_current_rail` command. The `report_power` command output will contain the power consumption ONLY for the last specified rail. The rails for

which power is calculated is defined in the Power Rail heading of the report file as shown in the sample file `pp_rail_VDD2.rpt` shown below:

Figure 11-1 Sample report file

```
*****
Report:          average power
Design:          d
Library File(s): ./MULTI-VDD/power_level.db
Clock Name: default Clock Period: 1ns
Power Rail:      VDD2
Reporting Threshold: 0 Watt
Date:            May 22, 2005 5:49 pm
*****

.....
Definitions:
Total Power      = Dynamic + Leakage
Dynamic Power    = Switching + Internal
Switching Power  = load capacitance charge or discharge power
Internal Power   = power dissipated within a cell
X-tran Power     = component of dynamic power-dissipated into x-transitions
Glitch Power     = component of dynamic power-dissipated into detectable
                   glitches at the nets
Leakage Power    = reverse-biased junction leakage + subthreshold leakage
-----
Level      Instance_Name (Cell_Name)  #_of_States
-----
Total      Dynamic   Leakage   Switching   Internal   X-tran   Glitch
Power      Power      Power      Power      Power      Power      Power
in Watt    in Watt    in Watt    in Watt    in Watt    in Watt    in Watt
           (% of Tot) (% of Tot) (% of Dyn) (% of Dyn) (% of Dyn) (% of Dyn)
-----
-----
Instances
*----- 0  pp_root (.) 40000
6.183e-06 6.183e-06 8.505e-11 1.800e-07 6.003e-06 0.000e+00 0.000e+00
           (100.00%) ( 0.00%) ( 2.91%) ( 97.09%) ( 0.00%) ( 0.00%)
-----
-*----- 1  d (d) 40000
6.183e-06 6.183e-06 8.505e-11 1.800e-07 6.003e-06 0.000e+00 0.000e+00
           (100.00%) ( 0.00%) ( 2.91%) ( 97.09%) ( 0.00%) ( 0.00%)
-----
--*----- 2  d/U11 (BUF16_2) 2000
2.670e-06 2.670e-06 1.500e-12 0.000e+00 2.670e-06 0.000e+00 0.000e+00
           (100.00%) ( 0.00%) ( 0.00%) (100.00%) ( 0.00%) ( 0.00%)
-----
--*----- 2  d/U5 (AND2_5) 3000
4.314e-07 4.314e-07 9.300e-12 1.714e-08 4.143e-07 0.000e+00 0.000e+00
           (100.00%) ( 0.00%) ( 3.97%) ( 96.03%) ( 0.00%) ( 0.00%)
-----
```



Multi-voltage Cells

Grouping cells by rail cannot solve all the problems related to the multi-voltage design. Suppose you have a multi-voltage cell that has Vdd1 and Vdd2 power supplies, as follows:

```
pin (Y) {  
    internal_power() {  
        power_level : VDD1 ;  
        power (power_r1) {  
            values ("1.934150, 2.148130");  
        }  
    }  
    internal_power() {  
        power_level : VDD2 ;  
        power (power_r1: {  
            values ("1.934150, 2.148130");  
        }  
    }  
}
```

For each toggle, the total power consumption includes the contribution from difference power supplies. This contribution can only be distinguished in the library during the characterization.

To retrieve the information in the library, the power rails in the design and library need to be linked. Use the `create_power_rail_mapping -library_rail` command to map the power rails in the library, as follows:

```
dc_shell> create_power_rail_mapping V3 \  
          -library_rail Vdd0 -inst "R1"
```

Reporting Power Rail Mapping

If there are MVdd cells in a design, you may want to know the power rails in the library before creating the power rail mapping between the rails in the design and in the libraries. The `report_power_rail_mapping` command will do this. After linking the design and the libraries, this command reports the libraries and their power rails.

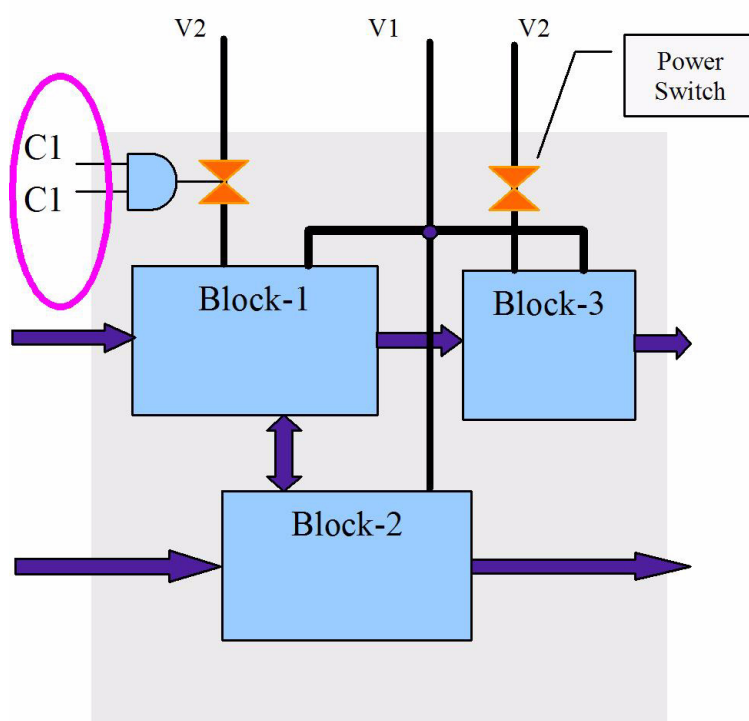
After grouping the design instances to the appropriate power rails, this command reports the mapping information as well. The `report_power_rail_mapping` command can be used before or after power rail mapping.

Power-gating Support

Power gating is an effective leakage power saving technology. It saves power by cutting off the power supply to certain parts of the design dynamically. If a block is not used for a while, its power supply can be cut off.

The diagram below shows a design with multiple power rails. The V2 rail, which feeds Block-3 and portions of Block-1, can be shut off when those instances are not used.

Figure 11-2 Design with multiple power rails



Monitoring the Power Off Signal

The power switch status affects the power consumption inside the blocks. When these blocks contain retention registers, the state-saving balloon latch within the register cells will be powered, while the rest of the block power is disabled, thus consuming no power. In this case, the leakage power is minimal and the dynamic power is 0. While the power switch is on, the block works normally. There is both dynamic and leakage power consumption inside the block.

Knowing the power switching on or off status is important in order to analyze the power correctly. The `-power_off` switch is used to capture the power switch status. See the example below.

```
create_power_rail_mapping V2 -instance "B1 B2" -power_off \
```


"C1 & C2"

PrimePower follows certain rules to handle the `power_off` signal. When the switching activity is from VCD and the power off condition is true, PrimePower knows that the power supply is in shut off state and no power will be consumed. This applies to both leakage and dynamic power. When the power off condition is false, the regular power analysis will be performed.

When the switching activity is from SAIF, the static probability of the power off condition is used to scale the leakage as default.

If the activity information does not capture the actual toggles during power off, users can set the variable `pwr_scale_dyn_at_power_off` to true. PrimePower will scale the dynamic power as well as the leakage power.

Sample Script

Assume power rail V2 is power gated. The second power rail mapping command is mandatory. It defines the power off condition and the power rail that will be controlled by power gating. After using that command, you can directly use `calculate_power` and `report_power` commands. You can check the rail mapping as well.

```
set_link_library "merged.db"
set_link_library "* lib1.db"
set link_path_per instance \
    [list [list {instance1} {* lib2.db}]
    [list {instance3} {* lib3.db}]]

read_verilog $myvlog

current_design $...mydesign
link
```

```

read_vcd -strip_path tb/top vcd.dump
report_power_rail_mapping

create_power_rail_mapping V1 -library_rail Vdd1 \
    inst "B1 B2 B3"
create_power_rail_mapping V2 -library_rail Vdd2 \
    -power_off "c2 & c2" -instance "r1"

report_power_rail_mapping

set_curretn_rail V1
calculate_power
report_power

```

By default, PrimePower will disable reporting if power rail commands are issued after the `calculate_power` command. Users can override this behavior by setting the variable `pwr_force_reporting` to `true`.

Dealing with the Multi-voltage Libraries

The Synopsys' multi-voltage design flow uses three types of libraries; multiple NLDM, SPDM, and merged NLDM. PrimePower supports all of them. In order for the power analysis to be performed properly, we need to be aware of special requirements for each type of library.

Multiple NLDM Library

In multiple NLDM libraries, the different libraries have the same cell names. A multi-voltage design using this type of library may have cells with the same name, but different VDDs. Power analysis requires the cells to be linked to the proper libraries. PrimePower

allows the linking of different cells to different libraries by using the `link_path_per_instance` variable as shown in the script in the previous section.

SPDM Library

The same cell name with different domains represented in a polynomial. The linking of this type of library is similar to the merged NLDM library.

12

Command Interface and Tcl

The PrimePower command interface is based on the Tcl command language.

Overview

Tcl concepts and the command interface are described in the following sections:

- [Tcl Syntax and PrimePower Commands](#)
- [Redirecting and Appending Output](#)
- [Using Command Aliases](#)
- [Interrupting Commands](#)
- [Using Scripts](#)

- [Listing and Running Previous Commands](#)
- [Suppressing Warning and Informational Messages](#)
- [Using Variables](#)
- [Using Collections](#)
- [Using Lists](#)
- [Flow Control](#)
- [Using Procedures](#)
- [Tcl Advanced Features](#)
- [Tcl Packages and Autoload](#)
- [Regular Expressions](#)
- [Extending Procedures](#)
- [TclPro Toolkit](#)
- [Object Attributes](#)
- [Using Attributes](#)
- [Attribute Names and Usage](#)

Tcl Syntax and PrimePower Commands

The PrimePower user interface is based on Tcl version 8.2. Using Tcl, you can extend the PrimePower command language by writing reusable procedures.

The Tcl language has a straightforward syntax. Every Tcl script is viewed as a series of commands, separated by a new-line character or semicolon. Each command consists of a command name and a series of arguments.

There are two types of PrimePower commands:

- Application commands
- Built-in commands

Each type is described in the following sections. Other aspects of Tcl version 8.2 are also described.

If you need more information about the Tcl language, consult books on the subject in the engineering section of your local bookstore or library.

Abbreviating Commands and Options

Application commands are specific to PrimePower. You can abbreviate application command names and options to the shortest unambiguous (unique) string. For example, you can abbreviate the `get_attribute` command to `get_attr` . Conversely, you cannot abbreviate most built-in commands.

Command abbreviation is meant as an interactive convenience. You should not use command or option abbreviations in script files, however, because script files are then susceptible to command changes in subsequent versions of the application. Such changes can make abbreviations ambiguous.

The variable `sh_command_abbrev_mode` determines where and whether command abbreviation is enabled. Although the default value is `Anywhere`, in the site startup file for the application, you can set this variable to `Command-Line-Only`. To disable abbreviation, set `sh_command_abbrev_mode` to `None`.

If you enter an ambiguous command, PrimePower attempts to help you find the correct command.

Example

The `set_wire` command as entered here is ambiguous:

```
pp_shell> set_wire
Error: ambiguous command 'set_wire' matched 4 commands:
      (set_wire_load_min_block_size, set_wire_load_mode,
set_wire_load_model ...) (CMD-006)
```

PrimePower lists up to three of the ambiguous commands in its error message. To list all the commands that match the ambiguous abbreviation, use the help function with a wildcard pattern. For example,

```
pp_shell> help set_wire_*
set_wire_load_min_block_size # Set minimum block size for
auto
wire load selection
set_wire_load_mode          # Set wire load mode for current design
set_wire_load_model         # Set wire load model on designs, hier
cells,
or ports
set_wire_load_selection_group # Set wire load selection
group
```

Using Tcl Special Characters

The characters listed in [Table 12-1](#) have special meaning for Tcl in certain contexts.

Table 12-1 Special Characters

Character	Meaning
\$	De-references a variable.
()	Used for grouping expressions.
[]	Denotes a nested command.
\	Used for escape quoting.
" "	Denotes weak quoting. Nested commands and variable substitutions still occur.
{ }	Denotes rigid quoting. There are no substitutions.
;	Ends a command.
#	Begins a comment.

Basic System Commands

[Table 12-2](#) lists some common tasks and the basic system commands for performing them.

Table 12-2 Common Tasks and Their System Commands

To do this	Use this
List the current PrimePower working directory.	<code>pwd</code>
Change the PrimePower working directory to a specified directory or, if no directory is specified, to your home directory.	<code>cd <i>directory</i></code>
List the files specified, or list all files in the working directory if no arguments are specified. Requires an <code>sh</code> program in your path.	<code>ls <i>directory_list</i></code>
Search for a file using <code>search_path</code> .	<code>which <i>filename</i></code>
Generate the current date and time.	<code>date</code>
Execute an operating system command. You should always use the <code>exec</code> command before using the <code>sh</code> command. This Tcl built-in command has some limitations. For example, it doesn't perform any expansion. For more information, see the <code>exec</code> man page.	<code>exec <i>command</i></code>
Execute an operating system command. Unlike <code>exec</code> , this command performs file name expansion. Requires an <code>sh</code> program in your path.	<code>sh <i>command</i></code>
Return the value of an environment variable.	<code>getenv <i>name</i></code>
Set the value of an environment variable. Any changes to environment variables apply only to the current PrimePower process and to any child processes launched by the current PrimePower process.	<code>setenv <i>name value</i></code>
Display the value of one or all environment variables.	<code>printenv <i>variable_pattern</i></code>

Using the Result of a Command

PrimePower commands return a result, which is interpreted by other commands as strings, booleans, integers, and so forth. With nested commands, the result can be used as

- A conditional statement in a control structure
- An argument to a procedure
- A value to which a variable is set

Example

Here is an example using a result:

```
if {[expr $a + 11] <= $b} {  
    echo "Done"  
    return $b  
}
```

Using Built-In Commands

Most built-in commands are intrinsic to Tcl. Their arguments do not necessarily conform to the PrimePower argument syntax. For example, many Tcl commands have options that do not begin with a dash, but do have a value argument.

Example

The Tcl `string` command has a `compare` option that you use like this:

```
string compare string1 string2
```

PrimePower Extensions and Restrictions

Generally, PrimePower implements all the Tcl built-in commands. However, PrimePower adds semantics to some Tcl built-in commands and imposes restrictions on some elements of the language. Here are the differences:

- The Tcl `rename` command is limited to procedures you have created.
- The Tcl `load` command is not supported.
- You cannot create a command called `unknown`.
- The auto-exec feature found in `tclsh` is not supported. However, `autoload` is supported.
- The Tcl `source` command has additional options: `-echo` and `-verbose`, which are nonstandard to Tcl.
- The `history` command has additional options, `-h` and `-r`, nonstandard to Tcl, and the form `history n`. For example, `history 5` lists the last five commands.
- The PrimePower command processor processes words that look like bus (array) notation (words that have square brackets, such as `a[0]`), so that Tcl does not try to execute the index as a nested command. Without this processing, you would need to rigidly quote such array references, as in `{a[0]}`.
- Always use braces (`{}`) around all control structures and procedure argument lists. For example, quote the `if` condition as follows:

```
if {!( $a > 2) } {  
    echo "hello world"  
}
```

Redirecting and Appending Output

You can direct the output of a command, procedure, or a script to a specified file in two ways:

- Using the `redirect` command
- Using the traditional UNIX redirection operators (`>` and `>>`)

The UNIX style redirection operators cannot be used with built-in commands. You must use the `redirect` command when using built-in commands.

You can use either of the following two commands to redirect command output to a file:

```
redirect temp.out {report_wire n*}  
report_wire n* > temp.out
```

You can use either of the following two commands to append command output to a file:

```
redirect -append temp.out {report_wire clk* }  
report_wire clk* >> temp.out
```

Note:

The Tcl built-in command `puts` does not respond to redirection of any kind. Instead, use the PrimePower command `echo`, which responds to redirection.

Using the redirect Command

In an interactive session, the result of a redirected command that does not generate a Tcl error is an empty string. For example,

```
pp_shell> redirect -append temp.out { history -h }
pp_shell> set value [redirect blk.out {plus 12 34}]
pp_shell> echo "Value is <$value>"
Value is <>
```

Screen output from a redirected command occurs only when there is an error. For example,

```
pp_shell> redirect t.out {set_input_transition I1 .5}
Error: Errors detected during redirect
      Use error_info for more info. (CMD-013)
```

This command had a syntax error because IO is not a floating-point number. The error is in the redirect file.

```
pp_shell> exec cat t.out
Error: value 'I1' for option '-transition' not of type
'float'
(CMD-009)
```

The `redirect` command is more flexible than traditional UNIX redirection operators. The UNIX style redirect operators `>` and `>>` are not part of Tcl and cannot be used with built-in commands. You must use the `redirect` command with built-in commands.

For example, you can redirect `expr $a > 0` only with

```
redirect file {expr $a > 0}
```

With `redirect` you can redirect multiple commands or an entire script. As a simple example, you can redirect multiple `echo` commands:

```
redirect e.out {
    echo -n "Hello"
    echo "world"
}
```

Getting the Result of Redirected Commands

Although the result of a successful `redirect` command is an empty string, you can get and use the result of the command you redirected. You do this by constructing a `set` command in which you set a variable to the result of your command, and then redirecting the `set` command. The variable holds the result of your command. You can then use that variable in a conditional expression.

For example,

```
redirect p.out {
    set rvs [read_verilog a.v]
}
if {$rvs == 0} {
    echo "read_verilog failed! Returning..."
    return
}
```

Using the Redirection Operators

Because Tcl is a command-driven language, traditional operators usually have no special meaning unless a particular command (such as `expr`) imposes some meaning. PrimePower commands respond to `>` and `>>` but, unlike UNIX, PrimePower treats the `>` and `>>`

as arguments to the command. Therefore, you must use white space to separate these arguments from the command and the redirected file name. For example,

Keep in mind that the result of a command that does not generate a Tcl error is an empty string. To use the result of commands you are redirecting, you must use the `redirect` command.

```
echo $my_variable >> file.out; # Right
echo $my_variable>>file.out; # Wrong!
```

The UNIX style redirect operators `>` and `>>` are not part of Tcl and cannot be used with built-in commands. You must use the `redirect` command with built-in commands.

Using Command Aliases

You can use aliases to create short forms for the commands you commonly use. For example, this command duplicates the function of the `dc_shell include` command when using PrimePower:

```
pp_shell> alias include "source -echo -verbose"
```

After creating a new alias, you can use it by entering this command:

```
pp_shell> include commands.pp
```

When you use aliases, keep the following points in mind:

- PrimePower recognizes an alias only when it is the first word of a command.

- An alias definition takes effect immediately but lasts only until you exit the PrimePower session. To save commonly used alias definitions, store them in the .synopsys_pp.setup file.
- You cannot use an existing command name as an alias name; however, aliases can refer to other aliases.
- Aliases cannot be syntax checked. They look like undefined procedures. For more information about syntax checking, see the section called “TclPro Toolkit” in the “Tcl Advanced Topics” chapter of the *PrimeTime User Guide: Advanced Timing Analysis*.

Interrupting Commands

If you enter the wrong options for a command or enter the wrong command, you can interrupt command processing by pressing Control-c.

The time the command takes to respond to an interrupt (to stop what it is doing and return to the prompt) depends on the size of the design and the function of the command being interrupted.

If PrimePower is processing a script file and you interrupt one of the script's commands, script processing is interrupted and PrimePower does not process any more commands in the script.

If you press Control-c three times before a command responds to your interrupt, pp_shell is interrupted and exits with this message:

```
Information: Process terminated by interrupt.
```

There are a few exceptions to this behavior, which are documented with the applicable commands.

Using Scripts

You can use the `source` command to execute scripts in PrimePower. A script file, also called a command script, is a sequence of `pp_shell` commands in a text file.

The syntax is:

```
pp_shell> source [-echo] [-verbose] scriptfile_name
```

By default, the `source` command executes the specified script file without showing the commands or the system response to the commands. The `-echo` option causes each command in the script to be displayed as it is executed. The `-verbose` option causes the system response to each command to be displayed.

Within a script file you can execute any PrimePower command. The script can be simple ASCII or gzip compressed.

The `source` command is also used to load bytecode-compiled scripts. For more information, see the section called “TclPro Toolkit” in the “Tcl Advanced Topics” chapter of the *PrimeTime User Guide: Advanced Timing Analysis*.

Adding Comments

You can add block comments to script files by beginning comment lines with the pound sign (`#`).

Add inline comments using a semicolon to end the command, followed by the pound sign to begin the comment. For example,

```
#  
# Set the new string  
#  
set newstr "New"; # This is a comment.
```

Controlling Script Processing When Errors Occur

By default, when a syntax or semantic error occurs while executing a command in a script, PrimePower discontinues processing the script. There are two variables which you can use to change the default behavior: `sh_continue_on_error` and `sh_script_stop_severity`.

To force PrimePower to continue processing the script no matter what, set `sh_continue_on_error` to `true`. This is usually not recommended because the remainder of the script may not perform as expected if a command fails due to syntax or semantic errors (for example, an invalid option).

To get PrimePower to stop the script when certain kinds of messages are issued, use the `sh_script_stop_severity` variable. This is set to `none` by default. Set it to `E` in order to get the script to stop on any message with error severity. Set it to `W` to get the script to stop on any message with warning severity.

Note:

The `sh_script_stop_severity` variable has no effect if the `sh_continue_on_error` variable is set to `true`.

Finding Scripts Using the `search_path` Variable

By default, the `source` command treats its `script_file_name` argument as an absolute file name.

To cause PrimePower to find the `script_file_name` using the search path, set the `sh_source_uses_search_path` variable to true. (The default is false.)

For example,

```
set sh_source_uses_search_path true
```

Logging Source File Contents

When you source a script, the `source` command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment.

To disable this logging, set the `sh_source_logging` variable to false. (The default is true.) For example,

```
set sh_source_logging false
```

Listing and Running Previous Commands

The `history` command lists the commands used in a `pp_shell` session. With no arguments, the `history` command lists the last *n* commands that you entered. The default is 20.

The `history` command is complex and can generate various forms of output. This section mentions some commonly used features.

The syntax is

```
history [keep count] [-h] [-r] [number_of_entries]
```

For more information, see the `history` man page.

To review the last 20 commands you entered, enter

```
pp_shell> history
1 source basic.tcl
2 read_db middle.db
.
.
18 current_design middle
19 link
20 history
```

To change the length of the history buffer, use the `keep` option. For example, this command specifies a history of 50 commands:

```
pp_shell> history keep 50
```

To limit the history list to three commands and to display them in reverse order, enter

```
pp_shell> history -r 3
20 history info 3
19 link
18 current_design middle
```

You can also redirect the output of the `history` command to create a command script:

```
pp_shell> redirect my_script {history -h}
```

Rerunning Previously Entered Commands

You can rerun and recall previously entered commands by using the exclamation point (!) operator.

[Table 12-3](#) lists the shortcuts you can use to rerun commands. Place these shortcuts at the beginning of a command line. They cannot be part of a nested command or a script.

Table 12-3 Shortcut for Rerunning Commands

To rerun	Use
The last command	!!
The <i>n</i> th command from the last	!-n
The command numbered <i>n</i> (from a history list)	!n
The most recent command that started with <code>text</code> (<code>text</code> must begin with a letter or underscore (<code>_</code>) and can contain numbers.)	!text

Examples

First recall the `current_design` command, by entering

```
pp_shell> history
1 source basic.tcl
2 read_db middle.db
3 current_design middle
4 link
5 history
```

Then rerun the third command, by entering

```
pp_shell> !3
```

Modifying the Previous Command Before Rerunning

You can modify and rerun the previous command executed using the `^x^y` form.

This recalls the last command, replaces all instances of `x` with `y`, then executes the command.

This is different from many UNIX shells, which only replace the first instance of `x` with `y`. For example, the following command was typed incorrectly, and was easily corrected:

```
pp_shell> sorce sort.tcl
Error: unknown command 'sorce' (CMD-005)
pp_shell> ^sorce^source
source sort.tcl
1
pp_shell>
```

Use more characters to help isolate the change. In the previous example, changing `sor` to `sour` would have modified the command, but it also would have changed the argument to `sourt.tcl`, which was not intended.

Suppressing Warning and Informational Messages

PrimePower provides commands for suppressing and redisplaying warning and informational messages.

Note:

You cannot suppress error messages and fatal error messages.

[Table 12-4](#) summarizes the commands for controlling the output of warning and informational messages. You can use these commands within a procedure (for example, to turn off specific warnings). If you suppress a message *n* times, you must unsuppress the message the same number of times to enable its output again. Use these commands carefully. For more information, see the man pages.

Table 12-4 Commands for Controlling Message Output

To do this	Use this command
Disable on-screen reporting of one or more messages	<code>suppress_message</code>
Enable on-screen reporting of previously disabled messages	<code>unsuppress_message</code>
Display the currently suppressed message IDs	<code>print_suppressed_messages</code>
Disable on-screen reporting of messages after a specified number of occurrences	<code>set_message_info</code>
Get information on the total number of messages generated, the number of occurrences of a specified message, or the limit set on reporting of a message	<code>get_message_info</code>

Using Variables

PrimePower variables can be divided into two categories: application variables and user-defined variables. The application variables are standard variables that control the behavior of PrimePower. User-defined variables are created by the user for any desired purpose.

[Table 12-5](#) summarizes commands that manage variables.

Table 12-5 Commands for Managing Variables

To do this	Use this
Assign a value to a variable	The <code>set</code> command. For example, <code>set x 27</code>
List variables with their current values	The <code>printvar</code> command, <code>echo \$variable_name</code> , or <code>set variable_name</code>
Substitute the value of a variable	The dollar sign (<code>\$</code>). For example, <code>set x 27</code> <code>set y \$x</code>
Remove user-defined variables	The <code>unset</code> command. For example, <code>unset x</code> Note: You cannot remove application variables.

Example 1

These commands set and remove variables:

```
pp_shell> set search_path ". /usr/synopsys/libraries"
. /usr/synopsys/libraries
```

```
pp_shell> set adir "/usr/local/lib"
/usr/local/lib
```

```
pp_shell> set my_path "$adir $search_path"
/usr/local/lib . /usr/synopsys/libraries
```

```
pp_shell> unset adir
```

```
pp_shell> unset my_path
```

Example 2

You can set variables to the result of a command:

```
pp_shell> set x [get_ports *]
```

Listing Variables

The `printvar` command list variables. [Table 12-6](#) summarizes the `printvar` command and its options.

Table 12-6 Commands for Listing Variables

To list this	Use this
All variables, alphabetically with their current values	<code>printvar</code>
All variables that match a specified pattern, alphabetically with their current values	<code>printvar pattern*</code>
A specified variable with its current value	<code>printvar variable_name</code> <code>echo \$variable_name set variable_name</code>
Only PrimePower variables	<code>printvar -application</code>
User-defined variables	<code>printvar -user_defined</code>

Example

To list the path variables, enter

```
pp_shell> printvar *path
link_path          = "*" lib.db"
search_path        = ". /usr/designs /usr/libraries"
sh_source_uses_search_path = "false"
```

User-Defined Variables

PrimePower supports any number of user-defined variables. Variables are either scalar or arrays.

The syntax of an array reference is:

```
array_name(element_name)
```

Arrays can be managed using the `array` command. See the referenced Tcl documentation for more information on the `array` command.

The following examples show how to set, unset, and examine variables:

```
pp_shell> set x 3
3
pp_shell> set A(1) 27
27
pp_shell> set A(hello) there
there
pp_shell> array names A
1 hello
pp_shell> unset x
pp_shell>
```

By default, when you create a new variable by using the `set` command at the `pp_shell` prompt, a message appears:

```
pp_shell> set new 2
```

```
Information: Defining new variable 'new'. (CMD-041)
```

This message can be useful in interactive mode because it warns you when you create a new variable unintentionally. For example, if you intend to set a system variable but mistype it slightly, the warning message tells you that you created a new variable instead of setting the existing system variable.

The following variables control the generation of CMD-041 messages:

- `sh_new_variable_message` (default: true)
- `sh_new_variable_message_in_proc` (default: false)
- `sh_new_variable_message_in_script` (default: false)

By default, the CMD-041 message is generated in interactive mode, but not during execution of procedures or scripts (when you are unlikely to be looking for messages). To enable generation of these messages during execution of procedures or scripts, set the variables appropriately. Note that doing so can cause a significant increase in runtime. For details, see the man pages for the variables.

Using Collections

PrimePower builds an internal database of the netlist and the attributes applied to the database. This database consists of several classes of objects, such as designs, libraries, ports, cells, nets, pins, and clocks. Most PrimePower commands operate on these objects.

A collection is a group of objects exported to the Tcl user interface. Collections have an internal representation (the objects), and sometimes a string representation. The string representation is generally used only for error messages.

You can create collections of objects, then apply a set of commands to interact with those collections. Collections can be homogeneous (contain objects of one type) or heterogeneous (contain objects of many types).

Note:

In some ways creating collections of objects in PrimePower is significantly different from doing so in Design Compiler. For an explanation of the differences, see the chapter called “Using PrimeTime With Design Compiler” in the *PrimeTime User Guide: Advanced Timing Analysis*.

The collection commands are divided into three categories:

- Commands that create collections of objects for use by another command
- Commands that manipulate collections
- Commands that query objects for you to view

You can use wildcards and filtering criteria to narrow the focus of a collection. You can store collections in variables for use in setting attributes, or for performing custom reporting.

Creating Collections

In PrimePower, you can pass a collection to a command by using the result of a command that creates the collection or by using a variable that contains a collection.

Primary Commands That Create Collections

[Table 12-7](#) lists the primary commands that create collections of objects.

Table 12-7 Primary Commands for Creating Collections

To create this collection	Use this command
Cells or instances	<code>get_cells</code>
Clock tree cells	<code>get_clock_tree_cells</code>
Get clock tree objects	<code>get_clock_tree_objects</code>
Designs	<code>get_designs</code>
Libraries	<code>get_libs</code>
Library cells	<code>get_lib_cells</code>
Library cell pins	<code>get_lib_pins</code>
Nets	<code>get_nets</code>
Pins	<code>get_pins</code>
Ports	<code>get_ports</code>
Get registers	<code>get_registers</code>
Get scan chain cells	<code>get_scan_chain_cells</code>

When used at the command prompt, any command that creates a collection will implicitly query the collection. You can also view the contents of a collection by using the `query_objects` command.

Primary Collection Command Example

The `get_cells` command creates a collection of cells in the design.

The syntax is

```
get_cells [-hierarchical] [-quiet] [-regexp] [-nocase] [-exact]
[-filter expression] patterns | -of_objects objects
```

For more information, see the `get_cells` man page.

For example, enter the following command to query the cells that begin with `o` and reference an FD2 library cell:

```
pp_shell> get_cells "o*" -filter {ref_name == FD2}
{"o_reg1", "o_reg2", "o_reg3", "o_reg4"}
```

Although the output looks like a list, it is not. The output is only a display of the `get_cells` result. The collection was not saved in a variable, nor passed to another command, so it was deleted as soon as it was displayed.

Given a collection of pins, use the following commands to query the cells connected to those pins:

```
pp_shell> set pins [get_pins o*/CP]
{"o_reg1/CP", "o_reg2/CP"}
pp_shell> get_cells -of_objects $pins
{"o_reg1", "o_reg2"}
```

Enter the following command to remove the wire load model from cells `i1` and `i2`:

```
pp_shell> remove_wire_load_model [get_cells {i1 i2}]
Removing wire load model from cell 'i1'.
Removing wire load model from cell 'i2'.
1
```

The primary collection commands (`get_cells`, `get_pins`, and so on) can take collections of the same type as arguments. For example, you can use syntax such as:

```
get_cells [get_cells u1]
```

This feature is often useful for writing procedures that can take a pattern or a collection as an argument.

Other Commands That Create Collections

[Table 12-8](#) lists other commands that create collections of objects.

Table 12-8 Other Commands for Creating Collections

To create this collection	Use this command
Cells or instances	<code>all_instances</code>
Fanin of pins, ports, or nets	<code>all_fanin</code>
Fanout of pins, ports, or nets	<code>all_fanout</code>
Objects connected to a net, pin, or port	<code>all_connected</code>
Ports	<code>all_inputs</code> , <code>all_outputs</code>
Register cells or pins	<code>all_registers</code>

Saving Collections

You can save collections by assigning the result of a command to a variable.

To save a collection for later use, set a variable equal to the collection, as in this command sequence:

```
pp_shell> set data_ports [get_ports {data[*]}]  
{"data[2]", "data[1]", "data[0]}"  
  
pp_shell> set_input_transition 2.0 $data_ports  
1
```

The result of the first command is a collection. PrimePower automatically displays the contents of the collection. Then the collection is passed into the `set_input_transition` command. You can deallocate the collection by unsetting the variable to which the collection was assigned (`data_ports`).

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

```
pp_shell> set myports [get_ports *]
```

then either of the following two commands deletes the collection referenced by the `myports` variable:

```
pp_shell> unset myports  
  
pp_shell> set myports "newvalue"
```

Collections can be implicitly deleted when they go out of scope, in other words, when the parent of the objects within the collection is deleted. For example, if your collection of ports is owned by a design, the collection is implicitly deleted when the design that owns the ports is deleted. When a collection is implicitly deleted, the variable

that referenced the collection still holds a string representation of the collection. However, since the collection is gone, this string value is not useful.

Querying Objects

The `query_objects` command searches for and displays objects in the PrimePower database.

The output of the `query_objects` command is similar to the `dc_shell find` command. The `query_objects` command does not have a meaningful return value; it simply displays the objects found and returns the empty string.

The syntax is

```
query_objects [-verbose] [-class class_name]  
              [-truncate elem_count] object_spec
```

For more information, see the `query_objects` man page.

Implicit Query

When commands that create collections are issued from the command prompt, they implicitly query the collection. You can control how many objects are displayed from the collection by using the variable `collection_result_display_limit`.

Use the following command to see the ports that match the pattern `in*`:

```
pp_shell> get_ports in*  
{"in0", "in1", "in2"}
```

You can also use the following commands to see the same ports.

```
pp_shell> query_objects [get_ports in*]
{"in0", "in1", "in2"}
pp_shell> query_objects -class port in*
{"in0", "in1", "in2"}
```

Using Wildcard Characters

Most commands that create collections allow a list of patterns, which contain wildcard characters. PrimePower uses some UNIX glob-style matching operators, including

- `*` – Matches 0 to n characters
- `?` – Matches one character

Example

Given ports `i0`, `i1`, `in0`, and `in1`, notice the queries in this sequence of commands:

```
pp_shell> get_ports i*
{"i0", "i1", "in0", "in1"}

pp_shell> query_objects [get_ports i?]
{"i0", "i1"}
```

Commands that create explicit collections, such as `get_cells`, allow you to search in the current instance or hierarchically by using a `-hierarchical` option. The rules for different kinds of searches are as follows:

- Using a wildcard pattern alone matches leaf names in the current instance. For example,

```
get_cells i1*
```

- Using a wildcard pattern with the `-hierarchical` option matches leaf names at each level of the hierarchy. For example, to find all cells in the hierarchy with the leaf name containing `n1`,

```
get_cells *n1* -hierarchical
```

- When you search for a wildcard pattern that contains the hierarchy separator, PrimePower breaks up the pattern around the hierarchy separator and matches each piece at progressively deeper levels of the hierarchy. For example, to find the cells in `i1` that begin with `i2`, then return a collection of cells in each `i1/i2*` that has a leaf name of `n1`, enter

```
get_cells i1/i2*/n1
```

If an object you specify has a name that contains a backslash character (`\`) or any wildcard characters (`*` or `?`), it is better to use the `-exact` option to find the object. Using `-exact` makes the primary collection creation command (such as `get_cells` or `get_ports`) consider the *patterns* argument to be a list of exact strings rather than a list of patterns with wildcards. For example:

```
get_cells -exact [list {*cell*1}] ;# finds cell actually  
named  
*cell*1  
get_cells -exact [list {a\b1}] ;# finds cell actually  
named a\b1
```

Wildcard matching uses the same logic as the Tcl “string match” command. Without `-exact`, each wildcard match character needs to be backslash-escaped in order to be considered exactly (not a wildcard). For example:

```
get_cells [list {\*cell\*1}]  
get_cells [list {a\\b1}]
```

Remember that the *patterns* argument to the `collection creation` command is a list. If you do not supply a well-formed list, additional backslashes will be necessary to communicate these special characters.

Filtering Collections

You can filter collections by using the `-filter` option with the primary commands that create collections (see commands listed in [Table 12-7](#)), or you can use the `filter_collection` command.

Using -filter Options

Many commands that create collections accept a `-filter` option that specifies a filter expression. A filter expression is a string composed of a series of logical expressions describing a set of constraints you want to place on a collection.

Each subexpression of a filter expression is a relation contrasting an attribute name (such as `area` or `direction`) with a value (such as `43` or `input`), by means of an operator (such as `==` or `!=`).

The following command gets the cells in `U1` that have an area no greater than 12 or reference a design (or library cell) named `AN2P`, `AO2P`, and so on. The command then assigns the collection to the `Cells` variable.

```
pp_shell> set Cells [get_cells "U1/*" \  
                  -filter {area <= 12 || ref_name =~ "A*P"}]
```

Using Filter Operators

The filter language supports the following logical operators:

- `AND` or `&&` – Logical AND (not case-sensitive)
- `OR` or `||` – Logical OR (not case-sensitive)

You can group logical expressions with parentheses to enforce order; otherwise, PrimePower evaluates the expressions from left to right.

The filter language supports the following relational operators:

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

In the following filter expression,

```
{area <= 12 || ref_name =~ "A*P"}
```

`area` is an attribute (or identifier), `<=` is a relational operator, `12` is a value, and `||` is the logical OR operator.

The filter language also supports the following existence operators:

- `defined`
- `undefined`

An existence operator determines whether an attribute is defined for an object. For example:

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

The right side of a relation can consist of a string or a numeric literal. You do not need to enclose Strings in quotation marks. This method is useful because a filter expression is usually the value for an argument, and the entire expression is enclosed in quotation marks.

As shown in the following command, you do not need to enclose the word *in* with quotation marks.

```
pp_shell> set iports [get_ports * -filter {direction == in}]
```

However, if an expression contains characters that are part of the filter language syntax, you must use curly braces to enclose the expression and double quotation marks to enclose string operands. As shown in the following example, parentheses are part of the filter language, so they are double quoted; and the complete expression is grouped in curly braces:

```
pp_shell > set x [filter_collection $ports \  
                 {full_name =~ "D(*)"}]
```

Relation Rules

Parsing a filter expression can fail because of

- Syntax problems
- An invalid attribute name
- A type mismatch between an attribute and the value

Here are the basic relational rules:

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with patterns.
- Boolean attributes can be compared only with `==` and `!=`.
The value can be only true or false.

Using Pattern Match Filter Operators

The pattern match filter operators, `=~` and `!~`, behave differently in different circumstances. They do one of two types of pattern matching:

- Simple wildcard matching (the default)
- Full regular expression matching

Commands that provide a `-filter` option also provide a `-regexp` option to enable you to change the filtering to full regular expressions. The `filter_collection` command also provides a `-regexp` option to enable full regular expressions.

Using the `filter_collection` Command

The `filter_collection` command is useful for filtering an existing collection. However, it is usually less efficient than using the `-filter` option.

The `filter_collection` command takes a collection and a filter expression as arguments. The result of `filter_collection` is a new collection, or an empty string if no objects match the criteria.

These two commands are equivalent:

```
get_cells * -filter "ref_name =~ A*P"

filter_collection [get_cells *] "ref_name =~ A*P"
```

The `filter_collection` command is useful for translating Design Compiler scripts that use the `filter` command. In addition, if you derive several collections from one larger collection, using the `filter_collection` command might be more efficient than using the `-filter` option, as shown in this series of commands:

```
pp_shell> set acells [get_cells ""]
{"u1", "u2", "u3", "u4"}

pp_shell> set ands [filter_collection $acells \
                  "ref_name =~ AN*"]
{"u1", "u2"}

pp_shell> set ors [filter_collection $acells \
                  "ref_name =~ OR*"]
{"u3", "u4"}
```

Optionally, `filter_collection` accepts the `-regexp` option, which changes `=~` and `!~` to perform real regular expression matching.

Using Implicit Collections as Arguments

Many PrimePower commands have arguments that accept a list of patterns. In such an argument, each pattern is either a collection handle or a string that matches one of a command-specific series of object classes. For example, the `set_input_delay` command accepts ports and pins, in that order. Therefore, the following command is a valid construct:

```
pp_shell> set_input_transition 5.0 [list out4 \
                                   [get_nets U1/*] n*1]
```

In the previous command, `set_load` first tries to find the `out4` port, and if it is not found, tries to find the `out4` net. The net collection of `U1/*` is explicit. The `n*1` pattern is matched against ports. If there is no match, it is matched against net.

The patterns `out4` and `n*1` are the implicit elements of the collection. For convenience, you can use the name of the individual object rather than an explicit collection such as `[get_ports out4]`, especially for singular objects.

Iterating Over the Elements of a Collection

The `foreach_in_collection` command iterates over a collection.

The syntax is

```
foreach_in_collection itr_var collections body
```

For more information, see the `foreach_in_collection` man page.

You can nest the `foreach_in_collection` command within other control structures, including another `foreach_in_collection` command.

During each iteration, the `itr_var` variable is set to a collection of exactly one object. Any command that accepts one of the collections also accepts the `itr_var` variable because they are of the same type (collection handle).

To generate a separate report for each cell, use these commands:

```
pp_shell> foreach_in_collection itr [get_cells o*] {  
    redirect [format "%s%d" $fbase $ndx]  
    report_cell  
}
```

Note:

You can use the `foreach` command to iterate over a list, but not a collection. Attempting to do so will cause the collection to be deleted.

Removing From and Adding to a Collection

PrimePower allows you to remove objects from or add objects to a collection by using the `remove_from_collection` and `add_to_collection` commands.

You can remove objects with the `filter_collection` command or by filtering objects when you initially create the collection. However, there are some constructs, such as removing the elements in one collection from another collection, that you cannot accomplish with filtering. Use the `remove_from_collection` and `add_to_collection` commands for this and other purposes.

The arguments to both commands are a collection and a specification of the objects that you want to add or remove. The specification can be a list of names, wildcard strings, or other collections.

The result of the commands is a new collection, or an empty string if the operation results in zero elements (in the case of the `remove_from_collection` command). The first argument (the base collection) is a read-only argument.

The syntax is

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

The base collection. The base collection is copied to the result collection and objects matching `object_spec` are added to or removed from the new collection. The base collection can be the empty collection.

object_spec

Lists named objects or collections to add or remove. The object class of each element in this list must be the same as the base collection. If the name matches an existing collection, the collection is used. Otherwise, PrimePower searches for the objects in the database using the object class of the base collection.

`-unique`

The `-unique` option for `add_to_collection` eliminates duplicate objects from the resulting collection.

When the *collection* argument to `add_to_collection` is the empty collection, some special rules apply to the *object_spec* argument. If *object_spec* is not empty, at least one homogeneous collection must exist in the *object_spec* list. The first homogeneous collection in the *object_spec* list becomes the base collection and sets the object class for the function.

Examples

This command sets the ports variable for collection of all ports except for `CLOCK`:

```
pp_shell> set ports [remove_from_collection \  
                  [get_ports "*"] CLOCK]
```

This command results in a collection of all cells in the design except those in the top level of hierarchy:

```
pp_shell> set lcells [remove_from_collection \  
                  [get_cells * -hier] [get_cells *]]
```

You can add objects to a collection with the `add_to_collection` command:

```
pp_shell> set isos [add_to_collection [get_cells i*]\  
                  [get_cells o*]]
```

Most collections are homogeneous since commands like `get_ports`, `get_cells`, and so on, create homogeneous collections. Some commands, like `all_connected`, create heterogeneous collections. You can create heterogeneous collections using the `add_to_collection` command. For example,

```
pp_shell> set a [get_ports PH*]  
{"PH1", "PH2"}
```

```
pp_shell> set b [get_pins -regexp {reg(0|1)/CP}]
{"reg0/CP", "reg1/CP"}
pp_shell> query_objects -verbose [add_to_collection $a $b]
{"port:PH1", "port:PH2", "pin:reg0/CP", "pin:reg1/CP"}
```

Note:

In some ways, removing from and adding to a collection in PrimePower is significantly different from doing so in the dcs command mode in Design Compiler.

Sorting Collections

You can sort a collection using the `sort_collection` command. PrimePower sorts according to the list of attributes you specify.

Sorts are ascending by default. You can reverse the order using the `-descending` option. In an ascending sort, PrimePower first sorts Boolean attributes with the objects that have the attribute set to false, then sorts the objects that have the attribute set to true. In the case of a sparse attribute, objects that do not have the attribute follow the objects that have the attribute. Not all collections can be sorted, such as timing paths, timing arcs, or library timing arcs.

The syntax is

```
sort_collection [-descending] collection criteria
```

For example, enter the following command to sort the ports by direction, then by full name.

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

For example, to get a collection of hierarchical cells decreasing alphabetically, followed by leaf cells decreasing alphabetically, sort the collection of cells using the *is_hierarchical* and *full_name* attributes. In the following example, *i1* and *i2* are hierarchical, and *u1* and *u2* are leaf cells.

```
pp_shell> set c [get_cells {u2 i2 u1 i1}]
{"u2", "i2", "u1", "i1"}
pp_shell> set cs [sort_collection -descending $c \
                        {is_hierarchical full_name}]
{"i2", "i1", "u2", "u1"}
```

The *is_hierarchical* attribute is Boolean, so in a descending sort, cells that have the attribute set to true (*i1* and *i2*) are first. To resolve the equality for the *is_hierarchical* attribute on cells *i1* and *i2*, use the *full_name* attribute.

Using Collection Utility Commands

PrimePower provides these additional commands for manipulating collections:

- `sizeof_collection`
- `compare_collections`
- `copy_collection`
- `index_collection`

Note:

Some collections, such as timing paths, cannot be indexed, copied, or compared.

Determining the Size of a Collection

The `sizeof_collection` command returns the number of elements in a collection.

The syntax is

```
sizeof_collection collection
```

You can use the empty string (the empty collection) as an argument. `sizeof_collection ""` is always 0.

Examples

To gauge the size of a design, enter

```
pp_shell> sizeof_collection [get_cells * -hier]
```

To determine whether a collection command yielded results, enter

```
pp_shell> if {[sizeof_collection [get_cells U2/U2]] != 0}
```

Comparing Collections

The `compare_collections` command compares the contents of two collections, object for object. You can specify that PrimePower compare the objects in order.

The syntax is

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

If the objects in both collections are the same, the result is 0 (like the result of string compare). If the objects are different, the result is nonzero.

Copying Collections

The `copy_collection` command duplicates a collection, resulting in a new collection. The base collection remains unchanged.

The `copy_collection` command is an efficient mechanism for duplicating an existing collection. However, copying a collection and having multiple references to the same collection are significantly different.

The syntax is

```
copy_collection collection
```

Examples

If you create a collection and save a reference to it in variable `c1`, assigning the value of `c1` to another variable `c2` creates a second reference to the same collection:

```
pp_shell> set c1 [get_cells "U1*"]
{"U1", "U10"}
pp_shell> set c2 $c1
{"U1", "U10"}
pp_shell> echo $c1 $c2
_sel3 _sel3
```

The previous commands do not copy the collection; only `copy_collection` creates a new collection that is a duplicate of the original.

This command sequence shows the result of copying a collection:

```
pp_shell> set collection1 [get_cells "U1*"]
{"U1", "U10"}
pp_shell> set collection2 [copy_collection $collection1]
{"U1", "U10"}
pp_shell> compare_collections $collection1 $collection2
0
```

Indexing Collections

The `index_collection` command creates a collection of one object that is the n th object in another collection. Objects in a collection are numbered 0 through $n-1$.

Although collections that result from commands such as `get_cells` are not really ordered, each has a predictable, repeatable order: The same command executed n times (for example, `get_cells *`) creates a collection of cells in the same order.

The syntax is

```
index_collection collectionindex
collection
```

Specifies the collection handle to search.

```
index
```

Specifies the index into the collection.

Example

This example shows how to extract the first object in a collection.

```
pp_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
pp_shell> index_collection $c1 0
{"u1"}
```

Using Lists

Lists are an important part of Tcl; they are used to represent groups of objects. Tcl list elements can consist of strings or other lists.

[Table 12-9](#) shows the Tcl commands you can use with lists.

Table 12-9 Tcl Commands to Use With Lists

Command	Task
concat	Concatenates two lists and returns a new list.
join	Joins elements of a list into a string.
lappend	Creates a new list by appending elements to a list (modifies the original list).
lindex	Returns a specific element from a list; this command returns the indexed element if it is there, or an empty string if it is not there.
linsert	Creates a new list by inserting elements into a list (it does not otherwise modify the list).
list	Returns a list formed from its arguments.
llength	Returns the number of elements in a list.
lrange	Extracts elements from a list.
lreplace	Replaces a specified range of elements in a list.

Table 12-9 Tcl Commands to Use With Lists (Continued)

Command	Task
<code>lsearch</code>	Searches a list for a regular expression.
<code>lsort</code>	Sorts a list.
<code>split</code>	Splits a string into a list.

Most publications about Tcl contain extensive descriptions of lists and the commands that operate on lists. Note these facts about Tcl:

- This is the preferred method of entering a list:

```
[list a b c d]
```

The entries "a b c d", {a b c d}, and [list a b c d] might all work equally well. However, the preferred method is more reliable.

- Special characters pose some challenges for lists. A list of two bused port names might look like this:

```
{{a[0]} {a[0]}}
```

Note:

Do not use commas to separate list items, as you do in the default shell of Design Compiler.

Example

If you are attempting to perform command or variable substitution, the form with braces does not work. For example, given that variable `a` is set to 5, these commands yield different results:

```
pp_shell> set a 5
5
```

```
pp_shell> set b {c d $a [list $a z]}
c d $a [list $a z]
```

```
pp_shell> set b [list c d $a [list $a z]]
c d 5 {5 z}
```

- You can nest lists, as shown in the previous example.
- Any list is also a well-formed Tcl command.
- You can use the `concat` command or other Tcl commands to concatenate lists.

Using Other Utilities

Tcl contains several other commands that handle the following:

- Strings and regular expressions (commands such as `format`, `regexp`, `regsub`, `scan`, and `string`)
- File operations (commands such as `file`, `open`, and `close`)
- Launching system subprocesses (commands such as `exec`)
- The `date` command returns the current date and time. It does so without creating a subprocess. This should be used instead of the `exec date` command.

Quoting Values

You can quote values in these ways:

- Escape individual special characters with the backslash (\) so that they are interpreted literally. Note that the backslash character itself is very difficult to use from the user interface, so its use in netlist objects is discouraged. (The number of times a backslash needs to be escaped depends on whether the option to which it is passed is a string or a list.)
- Enclose a set of words separated by spaces within braces ({ }). This syntax is called rigid quoting because variable, command, and backslash substitutions do not occur.
- Enclose a set of words separated by spaces within double quotation marks (" "). This syntax is called weak quoting because variable, command, and backslash substitutions can still occur.

The PrimePower shell allows you to use `blk[0]` without quoting even though brackets (`[]`) are part of the language. Although `pp_shell` can help in some cases, rigidly quoting these strings, as in `{blk[0]}`, is always better.

Examples

In these examples, the value of variable `a` is 5. Both commands are valid but have different results:

```
pp_shell> set s "temp = data[$a]"  
temp = data[5]
```

```
pp_shell> set s {temp = data[$a]}  
temp = data[$a]
```

Nesting Commands

You can nest commands within other commands (also known as command substitution) by enclosing the nested command within brackets (`[]`). Tcl imposes a depth limit of 1000 for command nesting.

PrimePower makes one exception to the use of command nesting with square brackets so it can recognize netlist objects with bus references. PrimePower accepts a string, such as `data[63]`, as a name rather than as the word *data* followed by the result of command `63`. Without this extension, you would have to rigidly quote `data[63]`, as `{data[63]}`, or escape the square brackets, as in `data\[63\]`.

Examples

These commands are valid examples of how to nest a command:

```
pp_shell> set title "Gone with the Wind"
Gone with the Wind
```

```
pp_shell> set lc_title [string tolower $title]
gone with the wind
```

Evaluating Expressions

Tcl supports expressions. However, arithmetic operators are not included as part of the base Tcl language syntax. Instead, use the `expr` command to evaluate expressions. The `expr` command can also evaluate logical and relational expressions.

Examples

This construct is correct:

```
set a [expr (12*$p)]
```

This construct is not correct:

```
set a (12 * $p);
```

Flow Control

Flow control commands—`if`, `while`, `for`, `foreach`, `break`, `continue`, and `switch`—determine the execution order of other commands. You can use any `pp_shell` command in a control flow command, and you can include other control flow commands.

Using the `if` Command

An `if` command has two or more arguments:

- An expression to evaluate
- A script to conditionally execute based on the result of the expression

You can extend the `if` command to contain an unlimited number of `elseif` clauses and one `else` clause.

- An `elseif` argument to the `if` command requires two additional arguments: an expression and a script.
- An `else` argument requires only a script.

Example

This example shows the correct way to specify `elseif` and `else` clauses. Notice that the `else` and `elseif` clauses appear on the same line as the closing brace (`}`). This syntax is required because a new line indicates a new command. If the `elseif` clause is on a separate line, it is treated as a command, which it is not. The `elseif` clause is an argument to the `if` command.

```
if {$x == 0} {  
    echo "Equal"  
} elseif {$x > 0} {  
    echo "Greater"  
} else {  
    echo "Less"  
}
```

Note:

If you are comparing strings, use `string compare $x $y` rather than `$x==$y`.

Using while Loops

The `while` command is similar to the same construct in the C programming language. The `while` command has two arguments:

- An expression
- A script

Example

This `while` command prints squared values from 0 to 10:

```
set p 0
while {$p <= 10} {
    echo "$p squared is: [expr $p * $p]"; incr p
}
```

Using for Loops

The `for` command is similar to the same construct in the C programming language. The `for` command has four arguments:

- An initialization script
- A loop termination expression
- An iterator script
- An actual working script

Example

This example shows how to rewrite the `while` loop shown earlier as a `for` loop.

```
for {set p 0} {$p <= 10} {incr p} {
    echo "$p squared is: [expr $p * $p]"
}
```

Iterating Over a List: `foreach`

The `foreach` command is similar to the same construct in the C shell. This command iterates over the elements in a list.

The `foreach` command has three arguments:

- An iterator variable
- A list
- A script to execute (the script references the iterator's variable)

You cannot use `foreach` to iterate over a PrimePower collection; attempting to do so will destroy the collection.

Example 1

This statement prints an array:

```
foreach el [lsort [array names a]] {  
    echo "a\($el\) = $a($el)"  
}
```

Example 2

This statement searches in the `search_path` for several files, then reports whether the files are directories:

```
foreach f [which {t1 t2 t3}] {  
    echo -n "File $f is "  
    if { [file isdirectory $f] == 0 } {  
        echo -n "NOT "  
    }  
    echo "a directory"  
}
```

Terminating a Loop

The `break` and `continue` commands terminate a loop before the termination condition is reached, as follows:

- The `break` command causes the innermost loop to terminate.
- The `continue` command causes the current iteration of the innermost loop to terminate.

Using the `switch` Command

The `switch` command is equivalent to an `if` tree that compares a variable to a number of values. One of a number of scripts is executed, based on the value of the variable:

```
switch $x {  
  a {incr t1}  
  b {incr t2}  
  c {incr t3}  
}
```

The `switch` command has other forms and several complex options. For more examples of the `switch` command, see books about the Tcl language.

Using Procedures

To extend the Tcl command language, you can write reusable Tcl procedures. You can write new commands that can use an unlimited number of arguments, and the arguments can contain default values. You can use a varying number of arguments.

This procedure prints the contents of an array:

```
proc array_print {arr} {  
    upvar $arr a  
    foreach el [lsort [array names a]] {  
        echo "$arr\($el\) = $a($el)"  
    }  
}
```

Keep in mind the following points about procedures:

- Procedures can use any supported PrimePower command or other procedure you define.
- Procedures can be recursive.
- Procedures can contain local variables and reference variables outside their scope.
- Arguments to procedures can be passed by value or by reference.
- PrimePower provides extensions to Tcl procedures that allow you to define the syntax and semantics of arguments. With these features, you can write extensions to PrimePower that look like commands, and you can easily parse the arguments to your procedure. For more information, see the section called “Extending Procedures” in the “Tcl Advanced Topics” chapter of the *PrimePower User Guide: Advanced Timing Analysis*.

Using the `proc` Command to Create Procedures

The `proc` command creates a new Tcl procedure.

You cannot create a procedure using the name of an existing built-in or application command. However, if a procedure with the name you specify exists, the new procedure usually replaces the existing procedure.

When you invoke the new command, PrimePower executes the contents of the body.

The `proc` command returns an empty string. When a procedure is invoked, the return value is the value specified in a `return` command. If the procedure does not execute an explicit `return`, the return value is the value of the last command executed in the body of the procedure. If an error occurs while the body of the procedure executes, the procedure returns that error.

When the procedure is invoked, a local variable is created for each formal argument to the procedure. The value of the local variable is the value of the corresponding argument in the invoking command or the default value of the argument. Arguments with default values are not specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that do not have defaults.

Except for one special case, no extra arguments can exist. The special case permits procedures with variable numbers of arguments. If the last formal argument has the name `args`, a call to the procedure can contain more actual arguments than the procedure has formal arguments. In this case, all actual arguments,

starting at the one assigned to `args`, are combined into a list (as if you had used the `list` command). The combined value is assigned to the local variable `args`.

The arguments to Tcl procedures are named positional arguments. You can program positional arguments with default values; you can include optional arguments by using the special argument name `args`.

PrimePower allows you to program extended information for a procedure and its arguments. For more information, see the section called “Extending Procedures” in the “Tcl Advanced Topics” chapter of the *PrimePower User Guide: Advanced Timing Analysis*.

When the body is executed, variable names typically refer to local variables, which are created automatically when referenced and deleted as the procedure returns. One local variable is automatically created for each argument of the procedure. You can only access global variables by invoking the `global` or the `upvar` command.

The syntax of the `proc` command is

```
proc name arguments body
```

name

Names the new procedure.

arguments

Lists formal arguments for the procedure. Each list element specifies one argument. (The arguments list can be empty.) Each argument specifier is also a list with one or two fields.

- If there is one field in the specifier, it is the name of the argument.
- If there are two fields in the specifier, the first field is the argument name and the second field is its default value.

body

Defines the procedure script.

Programming Default Values for Arguments

To set up a default value for an argument, make sure the argument is located in a sublist that contains two elements: the name of the argument and the default value.

This procedure reads a favorite library by default, but reads a specific library if its name is given:

```
proc read_lib {{lname favorite.db}} {  
    read_db $lname  
}
```

Specifying a Varying Number of Arguments

You can specify a varying number of arguments by using the `args` argument. You can specify that at least some arguments must be passed into a procedure; you can then deal with the remaining arguments as necessary.

Example 1

To report the square of at least one number, use

```
proc squares {num args} {  
    set nlist $num  
    append nlist ""  
    append nlist $args  
    foreach n $nlist {  
        echo "Square of $n is [expr $n*$n]"  
    }  
}
```

Example 2

This procedure adds two numbers and returns the sum:

```
pp_shell> proc plus {a b} { return [expr $a + $b]}  
pp_shell> plus 5 6  
11
```

For more information about writing procedures, see books about the Tcl language.

Displaying the Body of a Procedure

The `info` command with the `body` argument displays the body (contents) of a procedure. The `proc_body` command is a synonym for the `info body` command.

If you define the procedure with the `hide_body` attribute, you cannot use the `info body` command to view the contents of the procedure.

The syntax is

```
info body proc_name  
proc_body proc_name
```

The *proc_name* argument is the name of the procedure.

Example

This example shows the output of the `info body` command for a simple procedure named `plus`:

```
pp_shell> proc plus {a b} {return [expr $a + $b]}
pp_shell> info body plus
return [expr $a + $b]
pp_shell>
```

Displaying the Formal Parameters of a Procedure

The `info` command with the `args` argument displays the names of the formal parameters of a procedure.

The syntax is

```
info args proc_name
```

The *proc_name* argument is the name of the procedure.

Example

This example shows the output of the `info args` command for a simple procedure named `plus`:

```
pp_shell> proc plus {a b} { return [expr $a + $b] }
pp_shell> info args plus
a b
pp_shell>
```

Tcl Advanced Features

The PrimePower command interface is based on the Tcl command language. The following sections explain some advanced features of Tcl in the following sections:

- [Tcl Packages and Autoload](#)
- [Regular Expressions](#)
- [Extending Procedures](#)
- [TclPro Toolkit](#)

Tcl Packages and Autoload

PrimePower supports the standard Tcl package and autoload facilities. However, the load command is not supported, so packages that require shared libraries cannot be used. PrimePower is shipped with the standard implementations of the packages that are part of the Tcl distribution. For reference, these packages can be found below the root of the Synopsys installation at:

auxx/tcllib/lib/tcl8.2

You can add a new Tcl package to PrimePower either by installing the package into the Synopsys tree or by adding a new directory to the `auto_path` variable in the application startup script (`.synopsys_pp.setup`).

PrimePower provides two default locations for packages to be loaded into the application:

- For application-specific Tcl packages,
auxx/tcllib/primepower
- For packages that work with all Tcl-based Synopsys tools,
auxx/tcllib/snps_tcl

For example, if you have a Tcl package called mycompanyPP that contains PrimePower reporting facilities used by mycompany, you would create a directory called mycompanyPP under auxx/tcllib/primepower and then put the pkgIndex.tcl file and Tcl source files for my package into that directory. You can use the package in your PrimePower scripts by using the following command:

```
package require mycompanyPP
```

For detailed information about Tcl packages, see the `package` command man page, Tcl reference books, or visit the following Web site:

```
http://tcl.activestate.com/doc/
```

Regular Expressions

Many commands that create explicit collections allow you to choose between simple wildcard patterns (using the `*` and `?` wildcard characters) and complete regular expressions. Such commands provide the `-regexp` option, which you can use to tell the collection engine to use the pattern matching you want. For example,

```
pp_shell> set gc [get_cells -regexp {i(1|2)_.*}]
```

Using `-regexp` causes PrimePower to view the patterns argument as a regular expression. In addition, the pattern matching operators in the filter expression (`=~` and `!~`) also use regular expressions. The following two commands, which illustrate basic regular expressions, are equivalent:

```
get_cells blk* -filter "ref_name =~ AN*"
```

```
get_cells -regexp {blk.*} -filter "ref_name =~ AN.*"
```

To perform a case-insensitive search in PrimePower, the `-regexp` and `-nocase` options must be combined.

PrimePower uses the same regular expression engine as Tcl. For details about supported regular expression language and the `regexp` command, see the man page for `regexp` or `re_syntax`, or consult books on the subject in the engineering section of your local bookstore or library.

Using Regular Expressions With Implicit Collections

Commands that generate implicit collections do not directly support regular expressions. However, because an element of an implicit collection can be an explicit collection, you can use regular expressions indirectly.

Using Regular Expressions With Hierarchy

Mixing `-regexp` and `-hierarchical` options is different from mixing a wildcard pattern with `-hierarchical`. Simply stated, hierarchical searches with regular expressions are always compared to the full name of the objects in question. The rules for hierarchical searches with regular expressions are as follows:

- Using `-regexp` alone matches leaf names in the current instance. For example,


```
get_cells -regexp i1.*
```
- Using `-regexp` with `-hierarchical` matches full names, relative to the current instance, for each object found at or below the current instance. This is independent of the existence of hierarchy separators in the pattern. For example, to create a collection of `i1/i2/n1`, `i1/i21/n1`, `i1/i2/i3/n1`, and so forth, enter


```
get_cells -regexp i1/i2.*/n1
```
- Using `-regexp` does not provide a direct method to match leaf names at each level of the hierarchy. However, you can emulate a method of matching leaf names by using filters. For example, to

use `-regexp` to do the same as the command `get_cells n1 -hierarchical`, use the `base_name` attribute, which is the leaf name of the cell. For example, enter

```
pp_shell> get_cells -regexp -hierarchical ".*" \
            -filter {base_name == n1}
```

Anchoring Regular Expressions

By default, PrimePower automatically anchors regular expression patterns. For example, the pattern `blk.*` is considered the same as `^blk.*$`, which is usually the intended behavior.

If you want a less restricted matching style, like the Tcl `regexp` command, prefix and suffix the pattern with `.*` to unanchor the pattern (that is, not do an exact match). For example, to get any cells that contain `U1`:

```
pp_shell> get_cells -regexp {.*U1.*}
```

This matches `U1`, `U11`, `U1A`, `U1_23`, plus `ZU1`, `ZZU1`, `hello_U1`, and so forth.

Using Regular Expressions With Buses

When using a regular expression to match buses, you must be very careful because the bus characters (`[ ]`) are part of the command language. In addition, the usage varies slightly depending on whether the command argument is a string or a list.

For a command argument which is a string, the following example shows the correct form. The expression argument to `filter_collection` is a string.

```
pp_shell> filter_collection -regexp [get_ports *] \  
    {full_name =~ a\[[0-1]\]}
```

This regular expression matches ports a[0] and a[1]. A single backslash (\) must precede the real square brackets.

For a command argument which is a list, the syntax depends on how you specify the list. Consider the following examples, which use the `get_ports` command. The “patterns” argument to `get_ports` is a list.

```
pp_shell> get_ports -regexp [list {a\[[0-1]\]}]  
pp_shell> get_ports -regexp {{a\[[0-1]\]}}
```

These two commands are essentially equivalent. Proper list forms require single backslash quoting (\), just like string arguments. It is recommended that you use a properly formatted list for a list argument, especially in this situation. However, when you pass a single string into the “patterns” argument, double backslash quoting (\\) is required, for example:

```
pp_shell> get_ports -regexp {a\\[[0-1]\\]}
```

The double backslash is required because the promotion of the string to a list consumes one of the backslashes.

Extending Procedures

Tcl enables you to write reusable procedures. Basic Tcl procedures have positional arguments, allow default values for arguments, and allow a varying number of arguments. PrimePower provides a set of

extensions, which allows you or a process developer to define help text and semantic information for the arguments of a procedure. This is called defining procedure attributes.

The procedure attributes that you can define include

- A one-line informational text string that is shown with simple help
- The command group in which the procedure exists
- Whether the procedure can be modified
- Whether the body of the procedure can be viewed
- Help text and semantic rules for each argument of the procedure
- Whether the procedure can be abbreviated

The `define_proc_attributes` command defines attributes for a procedure. Use this command to modify the attributes of an existing procedure. Once procedure attributes have been defined, you can parse the procedure arguments to your procedure with the `parse_proc_arguments` command.

Default Procedure Attributes

When you create a procedure by using the `proc` command,

- The procedure is placed in the Procedures command group.
- The procedure has no help text for its arguments.
- On invocation, the procedure name can be abbreviated, subject to the value of the `sh_command_abbrev_mode` variable.
- The body of the procedure can be viewed with `info body`.
- The procedure and its attributes can be modified.

Changing the Aspects of a Procedure

You can use the `define_proc_attributes` command to change the aspects of a procedure listed previously.

The `define_proc_attributes` command enables you to do the following:

- Define the help text and semantic rules for each argument.
- Make the help for a procedure look like the help for an application command.

The syntax is

```
define_proc_attributes proc_name [-info info_text]  
                        [-define_args arg_defs] [-command_group group_name]  
                        [-hide_body] [-hidden] [-permanent] [-dont_abbrev]  
proc_name
```

Specifies the name of the procedure to extend.

`-info info_text`

Displays the help text when you use the `help` command on the procedure.

`-define_args arg_defs`

Specifies a list of lists that sets the help text and defines the arguments to the procedure. See the next section, “Format for the `-define_args` Option.”

`-command_group group_name`

Changes the command group of the procedure.

`-hide_body`

Prevents the `info body` command from displaying the contents of the procedure. You can always use the `info args` command to view procedure arguments. This cannot be undone in the current session.

`-hidden`

Prevents the procedure from showing when using the `help` or `info` command.

`-permanent`

Prevents further modifications to the procedure. This cannot be undone in the current session.

`-dont_abbrev`

Prevents the procedure from ever being abbreviated, regardless of the value of the `sh_command_abbrev_mode` variable. This cannot be undone.

Format for the `-define_args` Option

The value for the `-define_args` option is a list of lists. Each element has this format:

```
arg_name option_help value_help [data_type] [attributes]
```

`arg_name`

The name of the argument.

`option_help`

A short description of the argument.

`value_help`

Typically, the argument name for a positional argument or a one-word description for the value of a command-line option. The value has no meaning for a Boolean option.

`data_type`

Optional; any of `string` (the default), `list`, `boolean`, `int`, `float`, or `one_of_string`. Option validation can use the data type.

`attributes`

Optional; a list of attributes that can be any of the following.

<code>required</code>	The default. If you use <code>required</code> , you cannot use <code>optional</code> .
<code>optional</code>	If you use <code>optional</code> , you cannot use <code>required</code> .
<code>value_help</code>	When argument help is shown for the procedure, this attribute also shows the valid values for a <code>one_of_string</code> argument. For data types other than <code>one_of_string</code> , this is ignored.
<code>values</code>	List of allowable values for a <code>one_of_string</code> argument. This is required if the argument type is <code>one_of_string</code> . If you use values with other data types, an error appears.

<code>merge_duplicates</code>	When an option appears more than once in a command, the values are concatenated into a list of values. The <code>parse_proc_arguments</code> receives a single entry, which is a list. When <code>merge_duplicates</code> is not specified, the right most value for the option is used.
<code>script</code>	Identifies the argument as a Tcl script. This is used in the TclPro flow, so that tools like <code>snps_checker</code> will recurse into these arguments.

There are two reasons for using the `define_parse_attributes` command: consistent formatted help, and syntax and semantic checking. To complete syntax and semantic checking, you must use the `parse_proc_arguments` command.

If you do not use `parse_proc_arguments`, the procedure cannot respond to `-help`. However, you can always use the following command:

```
help procedure_name -verbose
```

Example

The following procedure adds two numbers and returns the sum. For demonstration purposes, some unused arguments are defined.

```
pp_shell> proc plus {a b} { return [expr $a + $b]}
pp_shell> define_proc_attributes plus -info "Add two numbers" \
    -define_args { \
        {a "first addend" a string required} \
        {b "second addend" b string required} \
        {"-verbose" "issue a message" "" boolean optional}}
pp_shell> help -verbose plus
Usage: plus      # Add two numbers
    [-verbose]    (issue a message)
    a             (first addend)
    b             (second addend)
pp_shell> plus 5 6
11
```

Parsing Arguments Passed in to a Tcl Procedure

The `parse_proc_arguments` command parses the arguments passed to a Tcl procedure that was defined with the `define_proc_attributes` command. The `parse_proc_arguments` command works only if you use one procedure argument: the Tcl keyword `args`, meaning any number of arguments. Then, you pass the arguments to the `parse_proc_arguments`, which does the semantic checking for you.

Use `parse_proc_arguments` within Tcl procedures to enable support for argument validation and support of the `-help` option. Typically, `parse_proc_arguments` is the first command called within a procedure. You cannot use `parse_proc_arguments` outside a procedure.

The syntax is

```
parse_proc_arguments -args arg_list result_array
```

```
-args arg_list
```

Specifies the list of arguments passed into the Tcl procedure. This should be `$args`.

```
result_array
```

Specifies the name of the array variable in which the parsed arguments are to be stored.

When a procedure that uses `parse_proc_arguments` is invoked with the `-help` option, `parse_proc_arguments` prints help information (in the same style as using `help -verbose`) and then

causes the calling procedure to return. Similarly, if any type of error exists with the arguments (missing required arguments, invalid value, and so forth), `parse_proc_arguments` returns a Tcl error, and the calling procedure terminates and returns.

If you do not specify `-help` and the specified arguments are valid, the array variable `result_array` will contain each of the argument values that you entered, subscripted with the argument name. The argument names are not the names of the arguments in the procedure definition; the argument names are the names of the arguments as defined using the `define_proc_attributes` command.

Example

The following procedure shows how `parse_proc_arguments` typically is used. The procedure `argHandler` accepts an optional argument of each type supported by `define_proc_attributes`, then prints the options and values received.

```
proc argHandler {args} {
    parse_proc_arguments -args $args results
    foreach argname [array names results] {
        echo "    $argname = $results($argname)"
    }
}

define_proc_attributes argHandler \
    -info "Arguments processor" \
    -define_args {
        {{-Oos "oos help" AnOos one_of_string
            {required value_help {values {a b}}}}}
        {-Int "int help" AnInt int optional}
        {-Float "float help" AFloat float optional}
        {-Bool "bool help" "" boolean optional}
        {-String "string help" AString string optional}
        {-List "list help" AList list optional}}
        {-IDup "int dup" AIDup int {optional merge_duplicates}}}
```

Invoking `argHandler` with `-help` generates the following output:

```
pp_shell> argHandler -help
Usage: argHandler      # argument processor
      -Oos AnOos      (oos help:
                       Values: a, b)
      [-Int AnInt]    (int help)
      [-Float AFloat] (float help)
      [-Bool]         (bool help)
      [-String AString] (string help)
      [-List AList]   (list help)
      [-IDup AIDup]   (Int dup)
```

Invoking `argHandler` with an invalid option generates the following output and causes a Tcl error:

```
pp_shell> argHandler -Int z
Error: value 'z' for option '-Int' not of type 'integer'
(CMD-009)
Error: Required argument '-Oos' was not found (CMD-007)
```

Invoking `argHandler` with valid arguments generates the following output:

```
pp_shell> argHandler -Int 6 -Oos a -IDup 2 -IDup 3
      -Int = 6
      -Oos = a
      -IDup = 2 3
```

Note:

Only the arguments entered are shown in the output of `argHandler`. To test to see if a specific argument was entered, you would do the following in your procedure:

```
if {[info exists results(-Bool)]}{
    ...
}
```

TclPro Toolkit

TclPro is an open-source toolkit for Tcl programming. These are some of the tools that come with TclPro:

- TclPro Checker (procheck)
Static script syntax checker
- TclPro Compiler (procomp)
Stand-alone bytecode compiler
- TclPro Debugger (prodebug)
Tcl debugger

The Synopsys Syntax Checker (snps_checker), based on TclPro Checker, is included with the PrimePower distribution. It is not necessary to download the TclPro tools in order to get syntax checking.

This section discusses how you can leverage these tools in the Synopsys environment. Visit <http://tcl.sourceforge.net> for more information about the TclPro tools.

Installation Requirements

To use any TclPro tools other than the syntax checker, you must install TclPro on your system. After you have installed TclPro on your system, you must define the environment variable `SNPS_TCLPRO_HOME`, where the TclPro installation exists. For example, if you installed TclPro version 1.5 at `/u/tclpro1.5`, you must set the `SNPS_TCLPRO_HOME` variable to point at that directory. PrimePower uses this variable as a base for launching some of the TclPro tools. In addition, other Synopsys applications use this variable to link to the TclPro tools.

Checking the Syntax and Semantics of Your Scripts

The Synopsys Syntax Checker, based on the TclPro Checker, helps you find syntax and semantic errors in your Tcl scripts. Everything that you need for syntax and semantic checking is shipped with PrimePower. In this environment, the following items are checked:

- Unknown options
- Ambiguous option abbreviation
- Use of exclusive options
- Missing required options
- Validation of literal option values for numerical range (range, <=, >=)
- Validation of one-of-string (keyword) options
- Recursion into constructs that have script arguments, such as the `redirect` and `foreach_in_collection` commands
- Warning of duplicate options overriding previous values

Limitations Using the Synopsys Syntax Checker

The following limitations apply to the Synopsys Syntax Checker:

- Command abbreviation is not checked. Use of abbreviated commands will show up as undefined procedures.
- Aliases created with the `alias` command are not expanded and will show up as undefined procedures.

- A few checks done when the application is running might not be checked using the `snps_checker` environment. For example, some cases where one option requires another option are not checked.
- Script size is an issue with `snps_checker` and TclPro 1.3 and 1.5. Scripts up to a few thousand lines can be reasonably checked, but beyond that, CPU time becomes a factor. Do not try to check extremely large scripts using `snps_checker`.
- PrimePower allows you to specify Verilog-style bus names on the command line without rigid quoting, for instance, `A[0]`. This format with indices from 0 to 255 is checked. Wildcards `*` and `%` are also checked. Other forms, including ranges as `A[15:0]` would show as undefined procedures, unless represented as `{A[15:0]}`.
- User-defined procedures enhanced with the `define_proc_attribute` command are not checked. Because such procedures are declared with the `args` argument, no semantic errors are reported.

Running the Synopsys Syntax Checker

There are two ways to run `snps_checker` in the Synopsys environment. You can launch `snps_checker` from PrimePower, or you can run it stand-alone.

To run `snps_checker` stand-alone, use the wrapper scripts provided with the PrimePower installation. Running `snps_checker` directly will not work. For each platform there is a script in the appropriate bin directory. For PrimePower you can find the script in `sparcOS5/syn/bin/ptprocheck`, `linux/syn/bin/ppprocheck`, and so on.

To launch `snps_checker` from PrimePower, you need to load a package provided with the installation, which is loaded as follows:

```
package require snpsTclPro
```

This makes the `check_script` command available. Pass the name of the script you want to check to the `check_script` command.

Bytecode-Compiled Files

Bytecode-compiled files are created using the TclPro Compiler, `procomp`.

Bytecode-compiled files have the following advantages over ASCII scripts:

- The files are efficient to load.
- The files are secure; the contents are not readable.
- The body of compiled Tcl procedures is hidden.

PrimePower can load bytecode-compiled scripts. To load a bytecode-compiled file, you simply source it like you do other scripts. You do not need any files other than the application in order to load bytecode-compiled files.

Debugging Scripts

The TclPro debugger is the most complex tool in the package. The debugger is similar to most source code debuggers. You can step over lines in the script, step into procedures, set breakpoints, and so on. It is not stand-alone—it requires the application to be running while you debug the script.

Launching Prodebug From PrimePower

Running the TclPro debugger is dependent on the `SNPS_TCLPRO_HOME` environment variable. Once this variable is set, you can debug scripts using the `debug_script` command.

To launch prodebug from PrimePower, you need to load a package provided with the installation. The package is as follows:

```
package require snpsTclPro
```

This makes the `debug_script` command available. Pass the name of the script you want to debug to the `debug_script` command.

The `debug_script` command launches prodebug, and connects PrimePower to it. The script is instrumented, and then you are ready for debugging. You can make subsequent calls to instrument the script by sourcing the file with the `source` command or with the `debug_script` command.

For more information, see the TclPro documentation shipped with the debugger, or the `debug_script` man page.

TclPro Limitations

TclPro is not supported on the IBM RS/6000 platform. However, bytecode-compiled files can be loaded into this application on all platforms, including the IBM RS/6000; and snps_checker is supported on all UNIX platforms.

Object Attributes

An attribute is a string or value associated with an object in the design that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can write programs in Tcl to get attribute information from the design database and generate custom reports on the design.

The following sections describe the attributes and how to use them:

- [Using Attributes](#)
- [Attribute Names and Usage](#)

Using Attributes

PrimePower provides a set of commands for setting, reporting, listing, and creating attributes, as summarized in [Table 12-10](#).

Table 12-10 Attribute Commands

Attributes	Description
<code>list_attributes</code>	Shows the attributes defined for each object class or a specified object class; optionally shows application attributes.
<code>get_attribute</code>	Retrieves the value of any attribute from a single object.
<code>report_attribute</code>	Displays the value of all attributes on one or more objects; optionally shows application attributes.
<code>define_user_attribute</code>	Creates a new attribute for one or more classes.
<code>set_user_attribute</code>	Sets a user-defined attribute on one or more objects.
<code>remove_user_attribute</code>	Removes a user-defined attribute from one or more objects.

The `get_attribute`, `report_attribute`, `set_user_attribute`, and `remove_user_attribute` commands accept an object specification, which can be a collection or a list of collections. The object specification for the `get_attribute` command is limited to one collection containing one object.

Defining User Attributes

The `define_user_attribute` command defines a new attribute.

PrimePower has a set of attributes that it considers application attributes. You can view these attributes by using `list_attributes -application`. You must define any other attribute before you can use it in PrimePower. One such use is importing an attribute from the `.db` file.

You can apply attributes to most object classes in PrimePower. These can mark interesting cells or nets, store values you have computed, and so on. PrimePower cannot use these attributes, but you can use them in scripts, procedures, and so on. You can list the attributes you define using the `list_attributes` command.

When you define an attribute, decide on the appropriate data type.

The syntax is

```
define_user_attribute -type data_type -classes class_list
                        [-range_min min] [-range_max max]
                        [-one_of values] [-import] attr_name
```

For more information, see the `define_user_attribute` man page.

Use the following commands to define attributes of various types. You can specify more than one class. Enter

```
pp_shell> define_user_attribute attr_s \
           -class {cell net} -type string
pp_shell> define_user_attribute attr_i \
           -class cell -type int
```


The following command sequence defines attribute `attr_ir1` as ≥ 0 and ≤ 100 , attribute `attr_ir2` as ≥ 0 with no maximum, and attribute `attr_ir3` as ≤ 100 with no minimum.

```
pp_shell> define_user_attribute attr_ir1 \  
          -class cell -type int -range_min 0 -range_max 100  
pp_shell> define_user_attribute attr_ir2 \  
          -class cell -type int -range_min 0  
pp_shell> define_user_attribute attr_ir3 \  
          -class cell -type int -range_max 100
```

The following command defines attribute `attr_oo` for cells. The attribute is a string, but you can set it only to A, B, C, or D. Enter

```
pp_shell> define_user_attribute attr_oo \  
          -class cell -type string -one_of {A B C D}
```

Importing User-Defined Attributes

To load user-defined attributes (attributes that are not PrimePower application attributes) from a `.db` file into PrimePower, you must define the attributes in PrimePower before you read any designs.

Assume you created a Boolean attribute in Design Compiler called `MarkedPin` and placed this attribute on several pins. In PrimePower, before you read any designs you must define the `MarkedPin` attribute. For example, enter

```
pp_shell> define_user_attribute -class pin -type boolean \  
          -import MarkedPin
```

After you define the attribute in this way, PrimePower extracts it from the `.db` file as it would extract any other attribute. For example, enter

```
pp_shell> list_attributes -class pin
```

PrimePower displays text similar to this:

```
*****
Report : List of Attribute Definitions
Design :
Version: 2004.06
Date   : Tues May 12 13:25:04 2004
*****
```

```
Properties:
  A - Application-defined
  U - User-defined
  I - Importable from db (for user-defined)
```

Attribute Name	Object	Type	Properties	Constraints

--				
MarkedPin	pin	boolean	U,I	

Attributes appear on design objects only after the design is linked.

User-defined attributes can be imported from both top-level designs and lower-level designs. To import user-defined attributes at the top level, there are no special considerations. You can simply use `define_user_attribute -import`. However, note that a top-level design attribute is attached to a design, but an inherited (lower-level) design attribute is attached to an instance of that design, which is a cell. Therefore, to import and inherit a lower-level design attribute, the attribute must be defined for both designs and cells. The `-import` option is only needed at the design level.

Similarly, a top-level port attribute is attached to a port, but an inherited port attribute is attached to an instance of that port, which is a pin. Therefore, to import and inherit a port attribute, the attribute must be defined for both ports and pins. The `-import` option is only needed at the port level.

To import and inherit a design attribute, the attribute must be defined for both designs and cells, although it only needs to be imported for the design. To import and inherit a port attribute, the attribute must be defined for both ports and pins, although it only needs to be imported for the port. A required attribute is defined automatically if necessary to import a related attribute.

After the design is linked, you can see that user-defined attributes have been applied. For example, enter

```
pp_shell> report_attribute [get_pins */CP]
```

PrimePower displays text similar to this:

```
*****
Report : Attribute
Design : M
Version: X-2005.06
Date   : February 22, 2009 10:48 am
*****
```

Design	Object	Type	Attribute Name	Value
M	o_reg1/CP	boolean	MarkedPin	true
M	o_reg2/CP	boolean	MarkedPin	true
M	o_reg3/CP	boolean	MarkedPin	true
M	o_reg4/CP	boolean	MarkedPin	true

You can create a collection of all pins with this attribute. Enter

```
pp_shell> set ipins [get_pins * -hier -filter \
                "MarkedPin == true"]
```

Attribute Names and Usage

Find the attributes by class in these tables:

- `cell` object class, [Table 12-11](#)
- `clock` object class, [Table 12-12](#)
- `design` object class, [Table 12-13](#)
- `lib` object class, [Table 12-14](#)
- `lib_cell` object type, [Table 12-15](#)
- `lib_pin` object type, [Table 12-16](#)
- `net` object type, [Table 12-17](#)
- `pin` object class, [Table 12-18](#)
- `port` object class, [Table 12-19](#)

The PrimePower `get_attribute` command supports the attributes listed in the tables in this appendix.

PrimePower attributes are read-only (you cannot set them) unless otherwise specified.

Table 12-11 *Attributes of the cell Object Class*

Attribute name	Type	Description
area	float	The area of a cell. If the cell is hierarchical, this includes net area.
base_name	string	The leaf name of a cell. For example, the base_name of cell U1/U2/U3 is U3.
disable_timing	boolean	This attribute is true if the timing for the cell has been marked as disabled in set_disable_timing. You can set and unset the disable_timing attribute.
dont_touch	boolean	Identifies cells to be excluded from optimization in Design Compiler. Cells with the dont_touch attribute set to true are not modified or replaced during compilation in Design Compiler. Setting dont_touch on a hierarchical cell sets the attribute on all cells below it. Set with set_dont_touch, and used by characterize_context and create_timing_context. You can set and unset the dont_touch attribute.
full_name	string	The complete name of a cell. For example, the full name cell U3 within cell U2 within cell U1 is U1/U2/U3. The full_name attribute is not affected by current_instance.
is_clock_gating_check	boolean	This attribute is true when a cell is a clock-gating check cell. It is valid only after execution of report_clock_network or get_clock_network_objects.
is_clock_network_cell	boolean	This attribute is true when the cell is in the clock network. It is valid only after execution of report_clock_network or get_clock_network_objects.

Table 12-11 *Attributes of the cell Object Class (Continued)*

Attribute name	Type	Description
<code>is_combinational</code>	boolean	A cell is combinational if it is nonsequential or non-three-state and all of its outputs compute a combinational logic function. The <code>report_lib</code> command reports such a cell as not being a black box.
<code>is_generated</code>	boolean	This attribute is found to be true for each generated clock in the design. Typical use is after specifying the <code>all_clocks</code> command. You can filter generated clocks based on this attribute.
<code>is_mux</code>	boolean	This attribute is true if a cell is a multiplexer.
<code>is_three_state</code>	boolean	This attribute is true if a cell is a three-state device.
<code>is_rise_edge_triggered</code>	boolean	This attribute is true if a cell is used in the design as a rising-edge-triggered flip-flop.
<code>is_fall_edge_triggered</code>	boolean	This attribute is true if a cell is used in the design as a falling-edge-triggered flip-flop.
<code>is_positive_level_sensitive</code>	boolean	This attribute is true if a cell is used in the design as a positive level-sensitive latch.
<code>is_negative_level_sensitive</code>	boolean	This attribute is true if a cell is used in the design as a negative level-sensitive latch.
<code>is_hierarchical</code>	boolean	This attribute is true for any cells that are instantiations of another design, and false for cells that are instantiations of a library cell (also known as leaf cells).
<code>is_sequential</code>	boolean	A cell is sequential if it is not combinational.

Table 12-11 *Attributes of the cell Object Class (Continued)*

Attribute name	Type	Description
number_of_pins	integer	Number of pins on the cell. The number of pins can be different before and after linking. For example, if some pins were unconnected in a Verilog instance, after linking to the lower-level design, additional pins can be created on the cell.
object_class	string	The class of the object. This is a constant equal to <code>cell</code> .
pp_dynamic_power	double	Dynamic power consumption of the cell in watts.
pp_glitch_power	double	Glitch power of the cell in watts.
pp_internal_power	double	Internal power of the cell in watts.
pp_leakage_power	double	Leakage power of the cell in watts.
pp_number_of_states	double	Number of state values for the cell. This includes transition on all the input and output pins, including transitions to and from the X-state.
pp_peak_end_time	double	End time of peak interval for the cell in library time units (generally ns).
pp_peak_power	double	Peak power of the cell in watts.
pp_peak_start_time	double	Start time for the peak interval in library time units (generally ns).
pp_switching_power	double	Switching power for the cell in watts.
pp_total_power	double	Total power for the cell in watts.
pp_xtran_power	double	Power consumed during transitions to and from X-states for the cell.

Table 12-11 *Attributes of the cell Object Class (Continued)*

Attribute name	Type	Description
ref_name	string	The name of the design or library cell of which the cell is (or will be) an instantiation. Also known as the reference name. The linker looks for a design or library cell by this name to resolve the reference.
timing_model_type	string	Returns the timing model type of a cell. Valid values: ITS (interface timing specification) Stamp QTM (quick timing model) extracted DCL (Delay Calculation Language) none (normal library model) You can set the <code>timing_model_type</code> attribute.
wire_load_model_max	string	The name of the wire load model effective on a hierarchical cell for the maximum operating condition. You can set the <code>wire_load_model_max</code> attribute.
wire_load_model_min	string	The name of the wire load model effective on a hierarchical cell for the minimum operating condition (valid in <code>bc_wc</code> or <code>on_chip_variation</code> analysis types). You can set the <code>wire_load_model_min</code> attribute.
wire_load_selection_group_max	string	The name of the wire load selection group on a hierarchical cell for the maximum operating condition. You can set the <code>wire_load_selection_group_max</code> attribute.
wire_load_selection_group_min	string	The name of the wire load selection group on a hierarchical cell for the minimum operating condition (valid in <code>bc_wc</code> or <code>on_chip_variation</code> analysis types). You can set the <code>wire_load_selection_group_min</code> attribute.

Table 12-12 *Attributes of the clock Object Class*

Attribute name	Type	Description
clock_latency_fall_max	float	The maximum fall latency (insertion delay) for a clock. Set with <code>set_clock_latency</code> .
clock_latency_fall_min	float	The minimum fall latency (insertion delay) for a clock. Set with <code>set_clock_latency</code> .
clock_latency_rise_max	float	The maximum rise latency (insertion delay) for a clock. Set with <code>set_clock_latency</code> .
clock_latency_rise_min	float	The minimum rise latency (insertion delay) for a clock. Set with <code>set_clock_latency</code> .
clock_source_latency_early_fall_max	float	The maximum early falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_fall_min	float	The minimum early falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_rise_max	float	The maximum early rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_rise_min	float	The minimum early rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_fall_max	float	The maximum late falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_fall_min	float	The minimum late falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_rise_max	float	The maximum late rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_rise_min	float	The minimum late rising source latency. Set with <code>set_clock_latency</code> .
full_name	string	The name of the clock. This is set with <code>create_clock</code> . It is either the name given with the <code>-name</code> option, or the name of the first object to which the clock is attached. Once set, this attribute is read only.

Table 12-12 Attributes of the clock Object Class (Continued)

Attribute name	Type	Description
<code>hold_uncertainty</code>	<code>float</code>	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
<code>is_active</code>	<code>boolean</code>	This attribute is true if the clock is active (the default state). Clocks can be made active or inactive with <code>set_active_clocks</code> .
<code>max_fall_delay</code>	<code>float</code>	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .
<code>max_rise_delay</code>	<code>float</code>	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .
<code>max_time_borrow</code>	<code>float</code>	A floating-point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with <code>set_max_time_borrow</code> .
<code>min_fall_delay</code>	<code>float</code>	A floating-point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>min_rise_delay</code>	<code>float</code>	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>object_class</code>	<code>string</code>	The class of the object. This is a constant, equal to "clock". You cannot set the <code>object_class</code> attribute.

Table 12-12 *Attributes of the clock Object Class (Continued)*

Attribute name	Type	Description
period	float	The clock period (or cycle time) is the shortest time during which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. Set with <code>create_clock -period</code> .
propagated_clock	boolean	Specifies that clock latency (insertion delay) be determined by propagating delays from the clock source to destination register clock pins. If this attribute is not present, ideal clocking is assumed. Set with <code>set_propagated_clock</code> .
setup_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
sources	string	This is a collection of the source pins or ports of the clock. The sources are defined with the <code>create_clock</code> command.
waveform	string	This is a <code>string</code> representation of the clock waveform. For example, a clock rising at 2.5 and falling at 5.0 has a waveform attribute value of <code>{2.5 5}</code> . The waveform is defined with the <code>create_clock</code> command.

Table 12-13 *Attributes of the design Object Class*

Attribute name	Type	Description
<code>analysis_type</code>	string	The analysis type: single, bc_wc, or on_chip_variation.
<code>area</code>	float	The total area of the design. This is the sum of the areas of all leaf cells and nets.
<code>designWare</code>	boolean	This attribute specifies if a cell came from a designware design.
<code>dont_touch</code>	boolean	Identifies designs to be excluded from optimization in Design Compiler. Designs with the <code>dont_touch</code> attribute set to true are not modified or replaced during compile in Design Compiler. Setting <code>dont_touch</code> on a design has an effect only when the design is instantiated within another design as a level of hierarchy; setting <code>dont_touch</code> on the top-level design has no effect. Set with <code>set_dont_touch</code> , and used by <code>characterize_context</code> and <code>create_timing_context</code> .
<code>extended_name</code>	string	The complete, unambiguous name of a design. The <code>extended_name</code> of the design is the <code>source_file_name</code> attribute followed by a colon (:) followed by the <code>full_name</code> attribute. For example, the <code>extended_name</code> of design TOP read in from /u/user/simple.db is /u/user/simple.db:TOP.
<code>full_name</code>	string	The name of a design. For example, the <code>full_name</code> of design TOP read in from /u/user/simple.db is TOP. This name can be ambiguous because several designs of the same name can be read in from different files.
<code>is_current</code>	boolean	This attribute is true for the current design. This attribute changes for all designs when you use the <code>current_design</code> command.

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
max_area	float	A floating-point number that represents the target area of the design. The units must be consistent with the units used from the technology library during optimization. The max_area value is set in Design Compiler using <code>set_max_area</code> .
max_capacitance	float	A floating-point number that sets the default maximum capacitance design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_max_capacitance</code> .
max_fanout	float	A floating-point number that specifies the default maximum fanout design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_max_fanout</code> .
max_transition	float	A floating-point number that specifies the default maximum transition design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_max_transition</code> .
min_capacitance	float	A floating-point number that sets the default minimum capacitance design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_min_capacitance</code> .
min_fanout	float	A floating-point number that specifies the default minimum fanout design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_min_fanout</code> .

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
<code>min_transition</code>	float	A floating-point number that specifies the default minimum transition design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with <code>set_min_transition</code> .
<code>object_class</code>	string	The class of the object. This is a constant equal to <code>design</code> .
<code>operating_condition_max</code>	string	The name of the maximum or single operating condition for the design. Set with <code>set_operating_conditions</code> .
<code>operating_condition_min</code>	string	The name of the minimum operating condition for the design. This attribute is not valid in single operating condition analysis.
<code>process_max</code>	float	The process value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command.
<code>process_min</code>	float	The process value of the minimum operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. Operating conditions are associated with a design using the <code>set_operating_conditions</code> command.

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
rc_slew_lower_ threshold_pct_rise	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimePower uses to calculate delays and transition times. These attributes from a design object return the values obtained from the main library (the first library in the link path).
rc_slew_upper_ threshold_pct_rise		
rc_slew_lower_ threshold_pct_fall		
rc_slew_upper_ threshold_pct_fall		
rc_input_threshold_ pct_rise		
rc_input_threshold_ pct_fall		
rc_output_threshold_ pct_rise		
rc_output_threshold_ pct_fall		
rc_slew_derate_ from_library		
source_file_name	string	The name of the file from which the design was read. For example, the source_file_name of design TOP read in from /u/user/simple.db is /u/user/simple.db.
temperature_max	float	The temperature value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the ambient temperature value for the operating condition.

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
temperature_min	float	The temperature value of the minimum operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command. This attribute represents the ambient temperature value for the operating condition.
tree_type_max	string	The <code>tree_type</code> value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command. The <code>tree_type</code> value is used in prelayout interconnect delay estimation, and can have a value of <code>best_case</code> , <code>balanced_case</code> , <code>balanced_resistance</code> (cmos2 only), or <code>worst_case</code> .
tree_type_min	string	The <code>tree_type</code> value of the minimum operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command. The <code>tree_type</code> value is used in prelayout interconnect delay estimation, and can have a value of <code>best_case</code> , <code>balanced_case</code> , <code>balanced_resistance</code> (cmos2 only), or <code>worst_case</code> .
voltage_max	float	The voltage value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command. This attribute represents the supply voltage value for the operating condition.

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
<code>voltage_min</code>	float	The voltage value of the minimum operating condition for the design. The operating condition is defined in a library or by the <code>create_operating_conditions</code> command. You associate operating conditions with a design using the <code>set_operating_conditions</code> command. This attribute represents the supply voltage value for the operating condition
<code>wire_load_min_block_size</code>	float	Specifies the smallest hierarchical cell that has automatic wire load selection by area applied. If an automatic wire load selection group is specified as the default in the main library, or through the <code>set_wire_load_selection_group</code> command, it is applied to all hierarchical cells larger than the specified minimum block size.
<code>wire_load_mode</code>	string	Determines which wire load model to use to compute wire capacitance, resistance, and area for nets in a hierarchical design that has different wire load models at different hierarchical levels. Allowed values are <code>top</code> , which tells PrimePower to use the wire load model at the top hierarchical level; <code>enclosed</code> , which tells PrimePower to use the wire load model on the smallest design that encloses a net completely; and <code>segmented</code> , which indicates to break the net into segments, one within each hierarchical level. In the segmented mode, each net segment is estimated using the wire load model on the design that encloses that segment. The segmented mode is not supported for wire load models on clusters. If no value is specified for this attribute, PrimePower searches for a default in the first library in the link path. If no default is found, <code>top</code> is the default. Set with <code>set_wire_load_mode</code> .
<code>wire_load_model_max</code>	string	The name of the design's wire load model for maximum conditions. Set with <code>set_wire_load_model</code> .

Table 12-13 *Attributes of the design Object Class (Continued)*

Attribute name	Type	Description
wire_load_model_min	string	The name of the design's wire load model for minimum conditions. This attribute is not valid for single operating condition analysis. Set with <code>set_wire_load_model</code> .
wire_load_selection_group_max	string	The name of the design's wire load selection group for maximum conditions. Set with <code>set_wire_load_selection_group</code> .
wire_load_selection_group_min	string	The name of the design's wire load selection group for minimum conditions. Set with <code>set_wire_load_selection_group</code> .

Table 12-14 *Attributes of the lib Object Class*

Attribute name	Type	Description
default_max_capacitance	float	The library default maximum capacitance design rule limit.
default_max_fanout	float	The library default maximum fanout design rule limit.
default_max_transition	float	The library default maximum transition design rule limit.
default_min_capacitance	float	The library default minimum capacitance design rule limit.
default_min_fanout	float	The library default minimum fanout design rule limit.
default_min_transition	float	The library default minimum transition design rule limit.

Table 12-14 *Attributes of the lib Object Class (Continued)*

Attribute name	Type	Description
extended_name	string	The complete, unambiguous name of a library. The extended_name of the library is the source_file_name attribute followed by a colon (:) followed by the full_name attribute. For example, the extended_name of library tech1 read in from /u/user/lib1.db is /u/user/lib1.db:tech1.
full_name	string	The name of a library. For example, the full_name of library tech1 read in from /u/user/lib1.db is tech1. This name can be ambiguous because several libraries of the same name can be read in from different files.
is_mux	boolean	This attribute is true if a library cell is a multiplexer.
is_three_state	boolean	This attribute is true if a library cell is a three-state device.
is_rise_edge_triggered	boolean	This attribute is true if the library cell is used in the design as a rising-edge-triggered flip-flop.
is_fall_edge_triggered	boolean	This attribute is true if library cell is used in the design as a falling-edge-triggered flip-flop.
is_positive_level_sensitive	boolean	This attribute is true if the library cell is used in the design as a positive level-sensitive latch.
is_negative_level_sensitive	boolean	This attribute is true if the library cell is used in the design as a negative level-sensitive latch.
object_class	string	The class of the object. This is a constant equal to lib.

Table 12-14 *Attributes of the lib Object Class (Continued)*

Attribute name	Type	Description
source_file_name	string	The name of the file from which the library was read. For example, the source_file_name of library tech1 read in from /u/user/lib1.db is /u/user/lib1.db.

Table 12-15 *Attributes of the lib_cell Object Type*

Attribute name	Type	Description
area	float	A floating-point value representing the area of a library cell.
base_name	string	The name of a library cell. For example, the base_name of library cell tech1/AN2 is AN2.
disable_timing	boolean	This attribute is true if the timing for the library cell has been specified to be disabled using the set_disable_timing command.
dont_touch	boolean	Identifies library cells to be excluded from optimization. Values are true (the default) or false. Library cells with the dont_touch attribute set to true are not modified or replaced during compile. Set in Design Compiler with set_dont_touch.
full_name	string	The fully qualified name of a library cell. This is the name of the library followed by the library cell name. For example, the full_name of library cell AN2 in library tech1 is tech1/AN2.
is_combinational	boolean	This attribute is true if the library cell is not sequential.
is_mux		This attribute is true if a library cell is a multiplexer.
is_sequential	boolean	This attribute is true if the library cell is sequential.

Table 12-15 *Attributes of the lib_cell Object Type (Continued)*

Attribute name	Type	Description
is_three_state	boolean	This attribute is true if a library cell is a three-state device.
number_of_pins	integer	Number of pins on the library cell.
object_class	string	The class of the object. This is a constant, equal to lib_cell.
pp_dynamic_power	double	Sum of the dynamic power consumption of all the instances of the library cell, in watts.
pp_glitch_power	double	Sum of the glitch power consumption of all the instances of the library cell, in watts.
pp_internal_power	double	Sum of the internal power consumption of all the instances of the library cell, in watts.
pp_leakage_power	double	Sum of the leakage power of all the instances of the library cell, in watts.
pp_switching_power	double	Sum of the switching power of all the instances of the library cell in watts.
pp_total_power	double	Total power of all the instances of the library cell, in watts.
pp_xtran_power	double	Sum of the power consumed during transitions to and from the X-state, for all the instances of the library cell, in watts.
timing_model_type	string	Returns the timing model type of a library cell. The values are ITS (interface timing specification) Stamp QTM (quick timing model) extracted DCL (Delay Calculation Language) none (normal library model)

Table 12-16 *Attributes of the lib_pin Object Type*

Attribute name	Type	Description
base_name	string	The leaf name of the library cell pin. For example, the base_name of tech1/AN2/Z is Z.
clock	boolean	A Boolean value that is set to true when a clock attribute is attached to a lib_pin in the library definition.
direction	string	The direction of a pin. Value can be in, out, inout, or internal.
disable_timing	boolean	This attribute is true if the library pin has been specified to be disabled using the <code>set_disable_timing</code> command.
drive_resistance_fall	float	A floating-point value representing the linear drive resistance for falling delays of a library pin.
drive_resistance_rise	float	A floating-point value representing the linear drive resistance for rising delays of a library pin.
fanout_load	float	A floating-point value representing the fanout load value of a library pin. This value is used in computing max_fanout design rule cost.
full_name	string	The fully qualified name of a library cell pin. This is the name of the library followed by the library cell name followed by a pin name. For example, the full_name of pin Z on library cell AN2 in library tech1 is tech1/AN2/Z.
is_three_state	boolean	This attribute is true if the library pin is a three-state driver.
is_async_pin	boolean	This attribute is true if a library pin is an asynchronous preset/clear pin.
is_preset_pin	boolean	This attribute is true if a library pin is an asynchronous preset pin.
is_clear_pin	boolean	This attribute is true if a library pin is an asynchronous clear pin.

Table 12-16 *Attributes of the lib_pin Object Type (Continued)*

Attribute name	Type	Description
is_clock_pin	boolean	This attribute is true if a library pin is a clock pin of sequential cell.
is_data_pin	boolean	This attribute is true if a library pin is a data pin of a sequential cell.
is_clock_gating_pin	boolean	This attribute is true if a library pin is a pin of a clock-gating cell.
is_three_state_enable_pin	boolean	This attribute is true if a library pin is an enable pin of a three-state device.
is_three_state_output_pin	boolean	This attribute is true if a library pin could output a three-state signal.
is_mux_select_pin	boolean	This attribute is true if a library pin is a select pin of a multiplexer device.
is_rise_edge_triggered_clock_pin	boolean	This attribute is true if the library pin is used in the design as a rising-edge-triggered flip-flop clock pin.
is_rise_edge_triggered_data_pin	boolean	This attribute is true if the library pin is used in the design as a rising-edge-triggered flip-flop data pin.
is_fall_edge_triggered_clock_pin	boolean	This attribute is true if the library pin is used in the design as a falling-edge-triggered flip-flop clock pin.
is_fall_edge_triggered_data_pin	boolean	This attribute is true if the library pin is used in the design as a falling-edge-triggered flip-flop data pin.
is_positive_level_sensitive_clock_pin	boolean	This attribute is true if the library pin is used in the design as a positive edge-triggered flip-flop clock pin.
is_positive_level_sensitive_data_pin	boolean	This attribute is true if the library pin is used in the design as a positive edge-triggered flip-flop data pin.

Table 12-16 *Attributes of the lib_pin Object Type (Continued)*

Attribute name	Type	Description
is_negative_level_sensitive_clock_pin	boolean	This attribute is true if the library pin is used in the design as a negative edge-triggered flip-flop clock pin.
is_negative_level_sensitive_data_pin	boolean	This attribute is true if the library pin is used in the design as a negative edge-triggered flip-flop data pin.
max_capacitance	float	A floating-point value representing the maximum capacitance design rule limit for a library pin.
max_fanout	float	A floating-point value representing the maximum fanout design rule limit for a library pin.
max_transition	float	A floating-point value representing the maximum transition time design rule limit for a library pin.
min_capacitance	float	A floating-point value representing the minimum capacitance design rule limit for a library pin.
min_fanout	float	A floating-point value representing the minimum fanout design rule limit for a library pin.
min_transition	float	A floating-point value representing the minimum transition time design rule limit for a library pin.
object_class	string	The class of the object. This is a constant, equal to lib_pin.
pin_capacitance	float	A floating-point value representing the capacitance of a library pin.

Table 12-16 *Attributes of the lib_pin Object Type (Continued)*

Attribute name	Type	Description
rc_slew_lower_ threshold_pct_rise	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimePower uses to calculate delays and transition times. These attributes from a library pin return the values obtained from the library to which the pin belongs.
rc_slew_upper_ threshold_pct_rise		
rc_slew_lower_ threshold_pct_fall		
rc_slew_upper_ threshold_pct_fall		
rc_input_threshold_ pct_rise		
rc_input_threshold_ pct_fall		
rc_output_threshold_ pct_rise		
rc_output_threshold_ pct_fall		
rc_slew_derate_ from_library		

Table 12-17 *Attributes of the net Object Type*

Attribute name	Type	Description
area	float	The estimated area of a net. The net area is calculated using a wire load model.
ba_capacitance_max	float	A floating-point value representing the back-annotated capacitance on a net for maximum conditions. Set with <code>set_capacitance 0 read_parasitics</code> .
ba_capacitance_min	float	A floating-point value representing the back-annotated capacitance on a net for minimum conditions. Set with <code>set_capacitance 0 read_parasitics</code> .

Table 12-17 *Attributes of the net Object Type (Continued)*

Attribute name	Type	Description
<code>ba_resistance_max</code>	float	A floating-point value representing the back-annotated resistance on a net for maximum conditions. Set with <code>set_resistance</code> or <code>read_parasitics</code> .
<code>ba_resistance_min</code>	float	A floating-point value representing the back-annotated resistance of a net for minimum conditions. Set with <code>set_resistance</code> or <code>read_parasitics</code> .
<code>base_name</code>	string	The leaf name of a net. For example, the base name of net <code>i1/i1z1</code> is <code>i1z1</code> . You cannot set this attribute.
<code>dont_touch</code>	boolean	Identifies nets to be excluded from optimization in Design Compiler. Values are true (the default) or false. Nets with the <code>dont_touch</code> attribute set to true are not modified or replaced during compile with Design Compiler. Set with <code>set_dont_touch</code> .
<code>full_name</code>	string	The complete name of a net. For example, the <code>full_name</code> of net <code>i1z1</code> within cell <code>i1</code> is <code>i1/i1z1</code> . The <code>full_name</code> attribute is not affected by current instance. The <code>full_name</code> attribute is read-only.
<code>net_resistance_max</code>	float	A floating-point value representing the resistance of a net for maximum conditions. May be computed from wire load models or set using <code>set_resistance</code> or <code>read_parasitics</code> .
<code>net_resistance_min</code>	float	A floating-point value representing the resistance of a net for minimum conditions. May be computed from wire load models or set using <code>set_resistance</code> or <code>read_parasitics</code> .
<code>object_class</code>	string	The class of the object. This is a constant, equal to <code>net</code> .
<code>pin_capacitance_max</code>	float	A floating-point value representing the sum of all pin capacitances of a net for maximum conditions. You cannot set this attribute.

Table 12-17 *Attributes of the net Object Type (Continued)*

Attribute name	Type	Description
<code>pin_capacitance_min</code>	float	A floating-point value representing the sum of all pin capacitances of a net for minimum conditions. You cannot set this attribute.
<code>rc_network</code>	string	A string describing the parasitic data that has been back-annotated on the net, including resistor values (in KOhms) and capacitor values (in pF).
<code>total_capacitance_max</code>	float	A floating-point value representing the sum of all pin capacitances and the wire capacitance of a net for maximum conditions. You cannot set this attribute.
<code>total_capacitance_min</code>	float	A floating-point value representing the sum of all pin capacitances and the wire capacitance of a net for minimum conditions. You cannot set this attribute.
<code>wire_capacitance_max</code>	float	A floating-point value representing the wire capacitance of a net for maximum conditions. The value can be computed from wire load models or set using <code>set_capacitance</code> or <code>read_parasitics</code> .
<code>wire_capacitance_min</code>	float	A floating-point value representing the wire capacitance of a net for minimum conditions. The value can be computed from wire load models or set using <code>set_capacitance</code> or <code>read_parasitics</code> .

Table 12-18 *Attributes of the pin Object Class*

Attribute name	Type	Description
<code>arrival_window</code>	<code>string</code>	The minimum and maximum arrivals for rise and fall transitions. In order to get the <code>arrival_window</code> attribute on pins that are not endpoints, set the variable <code>timing_save_pin_arrival_and_slack</code> to <code>true</code> .
<code>case_value</code>	<code>string</code>	The user-specified logic value of a pin or port propagated from a case analysis or logic constant in the design.
<code>ceff_params_max</code>	<code>string</code>	Returns an access to the parameters internally computed by PrimePower to compute the effective capacitance of a driver pin of a cell, respectively for maximum operating conditions. The returned parameters are <code>rd</code> , <code>t0</code> , <code>delta_t</code> , and <code>Ceff</code> ; they represent how a driver is modeled for computing the effective capacitance.
<code>ceff_params_min</code>	<code>string</code>	Returns an access to the parameters internally computed by PrimePower to compute the effective capacitance of a driver pin of a cell, respectively for minimum operating conditions. The returned parameters are <code>rd</code> , <code>t0</code> , <code>delta_t</code> , and <code>Ceff</code> ; they represent how a driver is modeled for computing the effective capacitance.
<code>clock_latency_fall_max</code>	<code>float</code>	The user-specified maximum fall latency (insertion delay) of a pin in the clock network. Set with <code>set_clock_latency</code> .
<code>clock_latency_fall_min</code>	<code>float</code>	The user-specified minimum fall latency (insertion delay) of a pin in the clock network. Set with <code>set_clock_latency</code> .

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>clock_latency_rise_max</code>	float	The user-specified maximum rise latency (insertion delay) of a pin in the clock network. Set with <code>set_clock_latency</code> .
<code>clock_latency_rise_min</code>	float	The user-specified minimum rise latency (insertion delay) of a pin in the clock network. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_early_fall_max</code>	float	The maximum early falling source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_early_fall_min</code>	float	The minimum early falling source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_early_rise_max</code>	float	The maximum early rising source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_early_rise_min</code>	float	The minimum early rising source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_late_fall_max</code>	float	The maximum late falling source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_late_fall_min</code>	float	The minimum late falling source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_late_rise_max</code>	float	The maximum late rising source latency. Set with <code>set_clock_latency</code> .
<code>clock_source_latency_late_rise_min</code>	float	The minimum late rising source latency. Set with <code>set_clock_latency</code> .
<code>clock_sources</code>	string	This attribute contains a list of clock source pins that propagates through the subject pins. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>direction</code>	<code>string</code>	The direction of a pin. Value can be in, out, inout, or internal. The <code>pin_direction</code> attribute is a synonym for <code>direction</code> . Directions can change as a result of linking a design, as references are resolved.
<code>disable_timing</code>	<code>boolean</code>	Disables timing arcs. This has the same effect on timing as not having the arc in the library. Set with <code>set_disable_timing</code> .
<code>effective_capacitance_max</code>	<code>float</code>	The effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.
<code>effective_capacitance_min</code>	<code>float</code>	The effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks
<code>fanout_load</code>	<code>float</code>	A floating-point value representing the fanout load value of a pin. This value is used in computing <code>max_fanout</code> design rule cost.
<code>full_name</code>	<code>string</code>	The complete name of a pin to the top of the hierarchy. For example, the full name of pin Z on cell U2 within cell U1 is U1/U2/Z. The setting of the current instance has no effect on the full name of a pin. See also the <code>lib_pin_name</code> attribute.
<code>hold_uncertainty</code>	<code>float</code>	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
<code>is_clock_gating_clock</code>	<code>boolean</code>	This attribute is true if a pin is a clock-gating check clock pin. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>is_clock_gating_enable</code>	boolean	This attribute is true if a pin is a clock-gating check enable pin. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .
<code>is_multiple_clock_fanin_point</code>	boolean	This attribute is true if a pin has multiple clock sources merging through the pin. It is valid only after execution of the command <code>report_clock_network</code> or <code>get_clock_network_objects</code> .
<code>is_port</code>	boolean	This attribute is true for a pin or a port. Pins of ports are accessible only from a <code>timing_point</code> .
<code>is_potential_clock_source</code>	boolean	This attribute is true if a pin is identified as a potential clock source pin. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .
<code>is_three_state</code>	boolean	This attribute is true if a pin is a three-state driver.
<code>is_async_pin</code>	boolean	This attribute is true if a pin is an asynchronous preset/clear pin.
<code>is_preset_pin</code>	boolean	This attribute is true if a pin is an asynchronous preset pin.
<code>is_clear_pin</code>	boolean	This attribute is true if a pin is an asynchronous clear pin.
<code>is_clock_pin</code>	boolean	This attribute is true if a pin is a clock pin of a sequential cell.
<code>is_data_pin</code>	boolean	This attribute is true if a pin is a data pin of a sequential cell.
<code>is_clock_gating_pin</code>	boolean	This attribute is true if a pin is a pin of a clock-gating cell.

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>is_three_state_enable_pin</code>	boolean	This attribute is true if a pin is an enable pin of a three-state device.
<code>is_three_state_output_pin</code>	boolean	This attribute is true if a pin could output a three-state signal.
<code>is_mux_select_pin</code>	boolean	This attribute is true if a pin is the select pin of a multiplexer device.
<code>is_rise_edge_triggered_clock_pin</code>	boolean	This attribute is true if a pin is used in the design as a rising-edge-triggered flip-flop clock pin.
<code>is_rise_edge_triggered_data_pin</code>	boolean	This attribute is true if a pin is used in the design as a rising-edge-triggered flip-flop data pin.
<code>is_fall_edge_triggered_clock_pin</code>	boolean	This attribute is true if a pin is used in the design as a falling-edge-triggered flip-flop clock pin.
<code>is_fall_edge_triggered_data_pin</code>	boolean	This attribute is true if a pin is used in the design as a falling-edge-triggered flip-flop data pin.
<code>is_positive_level_sensitive_clock_pin</code>	boolean	This attribute is true if a pin is used in the design as a positive edge-triggered flip-flop clock pin.
<code>is_positive_level_sensitive_data_pin</code>	boolean	This attribute is true if a pin is used in the design as a positive edge-triggered flip-flop data pin.
<code>is_negative_level_sensitive_clock_pin</code>	boolean	This attribute is true if a pin is used in the design as a negative edge-triggered flip-flop clock pin.
<code>is_negative_level_sensitive_data_pin</code>	boolean	This attribute is true if a pin is used in the design as a negative edge-triggered flip-flop data pin.

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>lib_pin_name</code>	<code>string</code>	The leaf pin name. For example, the <code>lib_pin_name</code> of pin U2/U1/Z is Z. This attribute is read-only.
<code>max_capacitance</code>	<code>float</code>	A floating-point value representing the maximum capacitance design rule limit for a pin.
<code>max_fall_arrival</code>	<code>float</code>	A floating-point value that specifies the arrival time for the longest path with a falling transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.
<code>max_fall_delay</code>	<code>float</code>	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .
<code>max_fall_slack</code>	<code>float</code>	A floating-point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.
<code>max_fanout</code>	<code>float</code>	A floating-point value representing the maximum fanout design rule limit for a pin.
<code>max_rise_arrival</code>	<code>float</code>	A floating-point value that specifies the arrival time for the longest path with a rising transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.
<code>max_rise_delay</code>	<code>float</code>	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .

Table 12-18 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
<code>max_rise_slack</code>	<code>float</code>	A floating-point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.
<code>max_time_borrow</code>	<code>float</code>	A floating-point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with <code>set_max_time_borrow</code> .
<code>max_transition</code>	<code>float</code>	A floating-point value representing the maximum transition time design rule limit for a pin.
<code>min_capacitance</code>	<code>float</code>	A floating-point value representing the minimum capacitance design rule limit for a pin.
<code>min_fall_arrival</code>	<code>float</code>	A floating-point value that specifies the arrival time for the shortest path with a falling transition on a pin. In best-case worst-case mode, this value is for the best-case mode.
<code>min_fall_delay</code>	<code>float</code>	A floating-point value that specifies the minimum falling delay on clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>min_fall_slack</code>	<code>float</code>	A floating-point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated.
<code>min_fanout</code>	<code>float</code>	A floating-point value representing the minimum fanout design rule limit for a pin.

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
<code>min_rise_arrival</code>	<code>float</code>	A floating-point value that specifies the worst hold slack of all paths passing through a pin with a rising transition at a pin. In best-case, worst-case operating conditions, this value is for the best-case condition. If all such paths are unconstrained, the value is infinity.
<code>min_rise_delay</code>	<code>float</code>	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>min_rise_slack</code>	<code>float</code>	A floating-point value representing the worst slack at a pin for rising minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.
<code>min_transition</code>	<code>float</code>	A floating-point value representing the minimum transition time design rule limit for a pin.
<code>object_class</code>	<code>string</code>	The class of the object. This is a constant, equal to <code>pin</code> .
<code>pin_capacitance_max</code>	<code>float</code>	A floating-point value representing the capacitance of a pin for maximum conditions.
<code>pin_capacitance_min</code>	<code>float</code>	A floating-point value representing the capacitance of a pin for minimum conditions.
<code>pin_direction</code>	<code>string</code>	The direction of a pin. Value can be <code>in</code> , <code>out</code> , <code>inout</code> , or <code>internal</code> . This attribute exists for backward compatibility with <code>dc_shell</code> . See the <code>direction</code> attribute.

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
propagated_clock	boolean	Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with set_propagated_clock.
rc_slew_lower_ threshold_pct_rise_max	float	The characterization trip points (waveform measurement thresholds) and slewing factor that PrimePower uses to calculate delays and transition times. These attributes on an instance pin return the library threshold values obtained from the maximum library in a min-max analysis.
rc_slew_upper_ threshold_pct_rise_max		
rc_slew_lower_ threshold_pct_fall_max		
rc_slew_upper_ threshold_pct_fall_max		
rc_input_threshold_ pct_rise_max		
rc_input_threshold_ pct_fall_max		
rc_output_threshold_ pct_rise_max		
rc_output_threshold_ pct_fall_max		
rc_slew_derate_ from_library_max		

Table 12-18 *Attributes of the pin Object Class (Continued)*

Attribute name	Type	Description
rc_slew_lower_ threshold_pct_rise_min	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimePower uses to calculate delays and transition times. These attributes on an instance pin return the library threshold values obtained from the minimum library in a min-max analysis.
rc_slew_upper_ threshold_pct_rise_min		
rc_slew_lower_ threshold_pct_fall_min		
rc_slew_upper_ threshold_pct_fall_min		
rc_input_threshold_ pct_rise_min		
rc_input_threshold_ pct_fall_min		
rc_output_threshold_ pct_rise_min		
rc_output_threshold_ pct_fall_min		
rc_slew_derate_ from_library_min		
setup_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
user_case_value	string	The user-specified logic value of a pin or port.

Table 12-19 *Attributes of the port Object Class*

Attribute name	Type	Description
arrival_window	string	The minimum and maximum arrivals for rising and falling transitions.
case_value	string	The user-specified logic value of a pin or port propagated from a case analysis or logic constant in the design.

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>ceff_params_max</code>	string	The parameters used to find the maximum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (<code>rd</code> = drive resistance, <code>t0</code> = start of voltage ramp, <code>delta_t</code> = duration of voltage ramp, <code>ceff</code> = effective capacitance).
<code>ceff_params_min</code>	string	The parameters used to find the minimum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (<code>rd</code> = drive resistance, <code>t0</code> = start of voltage ramp, <code>delta_t</code> = duration of voltage ramp, <code>ceff</code> = effective capacitance).
<code>clock_gating_related_clock_pin</code>	string	If the subject pin is a clock-gating enable pin, the attribute will contain a list of pins that are related clock-gating clock pins. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .
<code>clock_gating_related_enable_pin</code>	string	If the subject pin is a clock-gating clock pin, the attribute will contain a list of pins that are related clock-gating enable pins. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .
<code>clock_latency_fall_max</code>	float	The user-specified maximum fall latency (insertion delay) for clock networks through a port. Set with <code>set_clock_latency</code> .
<code>clock_latency_fall_min</code>	float	The user-specified minimum fall latency (insertion delay) for clock networks through a port. Set with <code>set_clock_latency</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
clock_latency_rise_max	float	The user-specified maximum rise latency (insertion delay) for clock networks through a port. Set with <code>set_clock_latency</code> .
clock_latency_rise_min	float	The user-specified minimum rise latency (insertion delay) for clock networks through a port. Set with <code>set_clock_latency</code> .
clock_source_latency_early_fall_max	float	The maximum early falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_fall_min	float	The minimum early falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_rise_max	float	The maximum early rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_early_rise_min	float	The minimum early rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_fall_max	float	The maximum late falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_fall_min	float	The minimum late falling source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_rise_max	float	The maximum late rising source latency. Set with <code>set_clock_latency</code> .
clock_source_latency_late_rise_min	float	The minimum late rising source latency. Set with <code>set_clock_latency</code> .
clock_sources	string	This attribute contains a list of clock source pins that propagates through the subject port. It is valid only after execution of <code>report_clock_network</code> or <code>get_clock_network_objects</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>direction</code>	<code>string</code>	The direction of a port. Value can be in, out, inout, or internal. The <code>port_direction</code> attribute is a synonym for <code>direction</code> . You cannot set this attribute.
<code>disable_timing</code>	<code>boolean</code>	This attribute is true if the timing for the port has been marked as disabled with the <code>set_disable_timing</code> command.
<code>drive_resistance_fall_max</code>	<code>float</code>	A floating-point value representing the linear drive resistance for falling delays and maximum conditions, associated with an input or inout port. Set with <code>set_drive_resistance</code> .
<code>drive_resistance_fall_min</code>	<code>float</code>	A floating-point value representing the linear drive resistance for falling delays and minimum conditions, associated with an input or inout port. Set with <code>set_drive_resistance</code> .
<code>drive_resistance_rise_max</code>	<code>float</code>	A floating-point value representing the linear drive resistance for rising delays and maximum conditions, associated with an input or inout port. Set with <code>set_drive_resistance</code> .
<code>drive_resistance_rise_min</code>	<code>float</code>	A floating-point value representing the linear drive resistance for rising delays and minimum conditions, associated with an input or inout port. Set with <code>set_drive_resistance</code> .
<code>driving_cell_dont_scale</code>	<code>boolean</code>	When true, indicates not to scale the transition time on the port using the driving cell. Otherwise the transition time is scaled by operating condition factors. Set with <code>set_driving_cell</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>driving_cell_fall_max</code>	string	A string that names a library cell from which to copy max fall drive capability to be used in fall transition calculation for the port. Set with <code>set_driving_cell</code> .
<code>driving_cell_fall_min</code>	string	A string that names a library cell from which to copy the minimum fall drive capability to be used in fall transition calculation for the port. Set with <code>set_driving_cell</code> .
<code>driving_cell_from_pin_fall_max</code>	string	A string that names the <code>driving_cell_fall_max</code> input pin to be used to find timing arc max fall drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_from_pin_fall_min</code>	string	A string that names the <code>driving_cell_fall_min</code> input pin to be used to find timing arc minimum fall drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_from_pin_rise_max</code>	string	A string that names the <code>driving_cell_rise_max</code> input pin to be used to find timing arc rise drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_from_pin_rise_min</code>	string	A string that names the <code>driving_cell_rise_min</code> input pin to be used to find timing arc rise drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_library_fall_max</code>	string	A string that names the library in which to find the <code>driving_cell_fall_max</code> . Set with <code>set_driving_cell</code> .
<code>driving_cell_library_fall_min</code>	string	A string that names the library in which to find the <code>driving_cell_fall_min</code> . Set with <code>set_driving_cell</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>driving_cell_library_ rise_max</code>	string	A string that names the library in which to find the <code>driving_cell_rise_max</code> . Set with <code>set_driving_cell</code> .
<code>driving_cell_library_ rise_min</code>	string	A string that names the library in which to find the <code>driving_cell_rise_min</code> . Set with <code>set_driving_cell</code> .
<code>driving_cell_max_fall_ itrans_fall</code>	float	The value of the maximum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the falling transition at the <code>from_pin</code> of the driving cell that is used to compute the falling transition value at a pin that drives the port.
<code>driving_cell_max_fall_ itrans_rise</code>	float	The value of the maximum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the falling transition value at the <code>from_pin</code> of the driving cell that is used to compute the rising transition value at the pin that drives the port.
<code>driving_cell_max_rise_ itrans_fall</code>	float	The value of the maximum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the rising transition value at the <code>from_pin</code> of the driving cell that is used to compute falling transition value at the pin that drives the port.

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>driving_cell_max_rise_</code> <code>itrans_rise</code>	float	The value of the maximum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the rising transition value at the <code>from_pin</code> of the driving cell that is used to compute the rising transition value at the pin that drives the port.
<code>driving_cell_min_fall_</code> <code>itrans_fall</code>	float	The value of the minimum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the minimum falling transition value at the <code>from_pin</code> of the driving cell that is used to compute the falling transition value at the pin that drives the port.
<code>driving_cell_min_fall_</code> <code>itrans_rise</code>	float	The value of the minimum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the minimum falling transition value at the <code>from_pin</code> of the driving cell that is used to compute the rising transition value at the pin that drives the port.
<code>driving_cell_min_rise_</code> <code>itrans_fall</code>	float	The value of the minimum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the minimum rise transition value at the <code>from_pin</code> of the driving cell that is used to compute the falling transition value at the pin that drives the port.

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>driving_cell_min_rise_itrans_rise</code>	float	The value of the minimum input transition for the driving cell that was associated with a port by the <code>set_driving_cell</code> command. This attribute represents the minimum rise transition value at the <code>from_pin</code> of the driving cell that is used to compute the rising transition value at the pin that drives the port.
<code>driving_cell_multiply_by</code>	float	A floating-point value that multiplies the transition time of the port marked with this attribute. Set with <code>set_driving_cell</code> .
<code>driving_cell_no_design_rule</code>	boolean	This attribute is true if driving cell information has been set on a port with <code>set_driving_cell -no_design_rule</code> . If true, the driving cell's design rule limits (<code>max_capacitance</code> and so forth) are not used for the port.
<code>driving_cell_pin_fall_max</code>	string	A string that names the <code>driving_cell_fall_max</code> output pin to be used to find timing arc fall drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_pin_fall_min</code>	string	A string that names the <code>driving_cell_fall_min</code> output pin to be used to find timing arc fall drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_pin_rise_max</code>	string	A string that names the <code>driving_cell_rise_max</code> output pin to be used to find timing arc rise drive capability. Set with <code>set_driving_cell</code> .
<code>driving_cell_pin_rise_min</code>	string	A string that names the <code>driving_cell_rise_min</code> output pin to be used to find timing arc rise drive capability. Set with <code>set_driving_cell</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>driving_cell_rise_max</code>	string	A string that names a library cell from which to copy max rise drive capability to be used in rise transition calculation for the port. Set with <code>set_driving_cell</code> .
<code>driving_cell_rise_min</code>	string	A string that names a library cell from which to copy the minimum rise drive capability to be used in rise transition calculation for the port. Set with <code>set_driving_cell</code> .
<code>effective_capacitance_max</code>	float	The effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.
<code>effective_capacitance_min</code>	float	The effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks.
<code>fanout_load</code>	float	Specifies the fanout load on output ports. Set with <code>set_fanout_load</code> .
<code>full_name</code>	string	The name of a port. You cannot set this attribute.
<code>hold_uncertainty</code>	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
<code>input_transition_fall_max</code>	float	A floating-point value representing the fixed transition time for falling delays, maximum conditions associated with an input or inout port. Set with <code>set_input_transition</code> .
<code>input_transition_fall_min</code>	float	A floating-point value representing the fixed transition time for falling delays and minimum conditions associated with an input or inout port. Set with <code>set_input_transition</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>input_transition_rise_max</code>	float	A floating-point value representing the fixed transition time for rising delays and maximum conditions associated with an input or inout port. Set with <code>set_input_transition</code> .
<code>input_transition_rise_min</code>	float	A floating-point value representing the fixed transition time for rising delays and minimum conditions associated with an input or inout port. Set with <code>set_input_transition</code> .
<code>is_potential_clock_source</code>	boolean	This attribute is true if the port is identified as a potential clock source port.
<code>max_capacitance</code>	float	A floating-point number that sets the maximum capacitance value for input, output, or bidirectional ports, and designs. The units must be consistent with those of the technology library used during optimization. Set with <code>set_max_capacitance</code> .
<code>max_fall_delay</code>	float	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .
<code>max_fall_slack</code>	float	A floating-point value representing the worst slack at a port for falling maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated. You cannot set this attribute.
<code>max_fanout</code>	float	Specifies the maximum fanout load for the net connected to this port. PrimePower ensures that the fanout load on this port is less than the specified value. Set with <code>set_max_fanout</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>max_rise_delay</code>	float	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_max_delay</code> .
<code>max_rise_slack</code>	float	A floating-point value representing the worst slack at a port for rising maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
<code>max_transition</code>	float	Specifies the maximum transition time for the net connected to this port. The compile command ensures that value. Set with <code>set_max_transition</code> .
<code>min_capacitance</code>	float	A floating-point number that sets the minimum capacitance value for input and bidirectional ports. The units must be consistent with those of the technology library used. Set with <code>set_min_capacitance</code> .
<code>min_fall_delay</code>	float	A floating-point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>min_fall_slack</code>	float	A floating-point value representing the worst slack at a port for falling minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
<code>min_fanout</code>	float	A floating-point value representing the minimum fanout design rule limit for a port. Set with <code>set_min_fanout</code> .

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>min_rise_delay</code>	float	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with <code>set_min_delay</code> .
<code>min_rise_slack</code>	float	A floating-point value representing the worst slack at a port for rising minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
<code>min_transition</code>	float	A floating-point value representing the minimum transition time design rule limit for a port. Set with <code>set_min_transition</code> .
<code>object_class</code>	string	The class of the object. This is a constant, equal to <code>port</code> . You cannot set this attribute.
<code>pin_capacitance_max</code>	float	A floating-point value representing the pin capacitance of a port for maximum conditions (wire capacitance is not included). Set with <code>set_capacitance</code> .
<code>pin_capacitance_min</code>	float	A floating-point value representing the pin capacitance of a port for minimum conditions (wire capacitance is not included). Set with <code>set_capacitance</code> .
<code>port_direction</code>	string	The direction of a port. Value can be <code>in</code> , <code>out</code> , or <code>inout</code> . This attribute exists for backward compatibility with <code>dc_shell</code> . See the <code>direction</code> attribute. You cannot set this attribute.

Table 12-19 *Attributes of the port Object Class (Continued)*

Attribute name	Type	Description
<code>propagated_clock</code>	boolean	Specifies that the clock edge times be delayed by propagating the values through the clock network. Affects all sequential cells in the transitive fanout of this port. If this attribute is not present, PrimePower assumes ideal clocking. Set with <code>set_propagated_clock</code> command.
<code>setup_uncertainty</code>	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with <code>set_clock_uncertainty</code> .
<code>user_case_value</code>	string	The user-specified logic value of a pin or port.
<code>wire_capacitance_max</code>	float	A floating-point value representing the wire capacitance of a port for maximum conditions (pin capacitance is not included). Set with <code>set_capacitance</code> .
<code>wire_capacitance_min</code>	float	A floating-point value representing the wire capacitance of a port for minimum conditions (pin capacitance is not included). Set with <code>set_capacitance</code> .
<code>wire_load_model_max</code>	string	The name of a wire load model (for maximum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with <code>set_wire_load_model</code> .
<code>wire_load_model_min</code>	string	The name of a wire load model (for minimum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with <code>set_wire_load_model</code> .

A

FAQ, Troubleshooting, and Known Limitations

The following sections contain some questions and answers that a user might encounter.

Frequently Asked Questions (FAQ)

1. Isn't PrimePower the same as NanoSim?

PrimePower is intended for ASIC methodology and cells. It is applied to pure gate-level applications with fully-buffered static logic. Under these conditions, PrimePower provides fast and accurate results while fitting into the existing design flow. NanoSim is a transistor-level simulation tool; it still provides the most broadly applicable power analysis and diagnostics, but at the transistor level.

2. Can PrimePower handle designs that were developed using previous versions of Verilog?

PrimePower relies on Verilog netlists, VCD files, and Synopsys libraries. As long as the netlist is gate-level and the VCD is in the standard format, PrimePower can handle it.

3. How does PrimePower take into account the effect of interconnects on power dissipation?

The effect of interconnect on delays is accounted for by the `sdf_annotate` command (specified in the Verilog testbench). The interconnect loads due to wire come from the wire capacitance file. The gate capacitances are derived from the Synopsys `.db` file. The wire capacitance implies effective capacitance for the specified net.

4. Does PrimePower report static leakage power?

Yes. Provided the library is appropriately characterized, both state-dependent and state-independent leakage power are reported hierarchically by PrimePower. State-dependent power leakage is computed dynamically.

5. Does PrimePower report power for transitions related to high-impedance states?

PrimePower reports transitions through Z states.

6. Why is the accuracy of PrimePower affected by X-transitions?

Simulators such as Verilog-XL and VCS do not provide logic levels. PrimePower's accuracy is therefore impacted proportionally to the number of X-transitions it encounters.

7. Can PrimePower handle .db libraries without state and path dependencies?

Yes. PrimePower will also report a switched capacitance power event if a cell is missing from the library.

8. Does PrimePower handle multiple output transitions?

PrimePower estimates power for multiple transitions based on power-characterized data. This power source is not reported separately; rather it is included in the dynamic power component.

Known Limitations

The following are known limitations in PrimePower:

- PrimePower reports glitch power dissipation data, but its handling of complex glitch phenomena still needs improvement.
- PrimePower is based on an inertial delay model. Cells with non-inertial delay models will be processed, but the accuracy of reported power is not guaranteed.
- X-state transitions are handled. However, since the voltage for X-states are not known, the reported power dissipation has a built-in error depending upon the percentage of X-transitions compared to total transitions.
- PrimePower accepts structural level netlists. RTL blocks without any structure are treated by default as black boxes.
- Internal pins that do not exist in the Verilog model pin list are not accepted.

Index

Symbols

*.fsdb file 10-12

*.out file 10-12

A

abbreviating commands 2-6, 12-3

accuracy

 complex cells 8-29

 correlation 8-29

Activity File Formats 9-2

alias command 12-12

all_clocks command 7-10

all_connected command 12-28

all_fanin command 12-28

all_fanout command 12-28

all_inputs command 12-28

all_instances command 12-28

all_outputs command 12-28

all_registers command 12-28

analysis flow (PrimePower usage overview)
2-24

analysis setup 2-27

Annotating Load Delay 7-12

Annotating Switching Activity 8-14

Annotating Timing 7-12

Annotating Timing Checks 7-13

appending command output 12-9

arrival window

 endpoints 12-112

attributes 12-88

 cell object class, listed 12-89

 clock object class, listed 12-93

 defining (define_user_attribute) 12-84

 design object class, listed 12-96

 importing from .db files 12-85

 lib object class, listed 12-102

 lib_cell object class, listed 12-104

 lib_pin class, listed 12-106

 net object class, listed 12-109

 pin object class, listed 12-112

attributes, library

 internal_power 1-20

Average analysis 1-2

average cycle waveform 9-8

Average Cycle Waveform Data 8-48

average power analysis 1-2

B

Back-annotated Parasitics 6-22

back-annotating, after layout 1-22

back-annotation

 overview 2-32

- batch mode 2-10
- black box 5-20
- built-in commands 12-7
- bus
 - report 5-37

C

- Calculate Power Options 8-47
- calculate_power 8-3, 11-3
- Calculating and Reporting Power 1-12
- Calculating the Capacitance of a Net 6-3
- Calculation of leakage power 1-12
- capacitance, port 6-21
- capacitive load, obtaining 1-22
- capacitive switching power
 - (see switching power)
- case analysis 8-32
 - constant propagation based on cell logic 8-37
 - constant propagation log file 8-42
 - removing 8-41
 - reporting 8-40
 - scan chain disabling 8-34
- case-sensitivity
 - pp_shell 2-6
- cell 1-13
 - report 5-29
- cell leakage power, equation 1-12
- cell object class attributes list 12-89
- cell_leakage_power 1-13
- cells, library
 - complex 8-29
- Checking for Unannotated Nets 6-25
- checking the design 2-27
- Choosing a Wire Load Model 6-6
- C_{Loadi}, defined 1-22
- clock
 - characteristics, specifying 7-6
 - creating 7-7
 - off-chip 7-8
 - removing 7-10
 - report 5-28
 - selecting all clocks 7-10
 - virtual 7-8
- clock object class attributes list 12-93
- Clock Tree Power 10-19
- Clock-Related Power 10-15
- CMD-041 message 12-24
- collections 12-24
 - adding objects to 12-39
 - commands 12-26, 12-28
 - comparing 12-44
 - copying 12-45
 - creating 12-25
 - filtering 12-33
 - implicit 12-38
 - indexing 12-46
 - iterating over 12-38
 - removing object from 12-39
 - returning the number of elements in 12-44
 - sorting 12-42
- command 12-16
 - abbreviating 2-6
 - man pages 2-19
 - nesting 2-11
 - script 12-14
 - substitution 12-18
- command abbreviation 12-3
- command history 12-16
- command nesting 12-7
- commands
 - all_clocks 7-10
 - create_clock 7-7
 - debug_script 12-81
 - define_proc_attributes 12-69
 - define_user_attribute 12-83, 12-84
 - derive_clocks 7-8
 - get_attribute 12-83
 - get_clocks 7-9
 - help 2-15

- info body 12-61
- list_attributes 12-83
- man 2-18
- parse_proc_arguments 12-74
- printvar 2-13
- pwd 12-6
- regexp 12-65
- remove_case_analysis 8-41
- remove_clock 7-10
- remove_user_attribute 12-83
- report_attributes 12-83
- report_case_analysis 8-40
- restore_session 2-36
- save_session 2-36
- save_session command 2-36
- set 2-12
- set_case_analysis 8-38
- set_load 6-21
- set_user_attribute 12-83
- source 2-9
- while 12-53
- complex cells
 - accuracy 8-29
- compression scripts 2-9
- Conditional Delays From SDF 7-18
- console window 4-11
- constant propagation 8-37
 - log file 8-42
- control flow commands 12-52
- Correlation 8-29
- correlation
 - defined 8-29
- create_clock command 7-7
- current_design command 5-14
- current_instance command 5-15
- Custom Report Generation 10-21

D

- date 12-6
- debug_script command 12-81

- Debugging Problems With the VCD File 9-9
- Debugging the Annotation 8-30
- Default Power Report 10-2
- default_operating_condition 7-4
- define_proc_attributes command 12-69
- define_user_attribute command 12-83, 12-84
- Defining Wire Load Models 6-14
- definitions
 - C_{Loadi} 1-22
 - correlation 8-29
 - dynamic power 1-9
 - internal power 1-10
 - path-dependent power 1-18
 - short circuit power 1-10
 - static power 1-9
 - switching power 1-10
- derive_clocks command 7-8
- Deriving SDPD Switching Activity 8-28
- design attribute report 5-26
- design checking 2-27
- design data
 - link path 5-2
 - reading 5-4
 - search path 5-2
 - Verilog 5-6
 - VHDL 5-12
- design object class attributes list 12-96
- design objects 5-22
- Design Parasitics 6-1
- docking windows 4-4
- documentation (summary of PrimePower documentation) 2-15
- Dynamic Analysis 9-2
- dynamic power
 - defined 1-9
- dynamic power, analyzing 10-3

E

- environment 2-31

- equations
 - internal power 1-21
 - leakage power 1-12
 - switching power 1-19, 1-22
- error messages
 - man page 2-20
- Estimating Unannotated Switching Activity 8-25
- evaluating expressions 12-51
- Event-based Power Analysis 1-9
- exec command 12-6
- expressions, evaluating 12-51

F

- files
 - *.fsdb 10-12
 - *.out 10-12
 - bytecode-compiled 12-80
 - log file 2-5
 - pp_shell_command.log 2-5
 - script 12-14
 - startup 2-4
 - .synopsys_pt.setup 2-4
- filter expression 12-33
- filter_collection command 12-37
- filtering collections 12-33
- flow
 - PrimePower usage overview 2-24
- for command 12-54
- for loops 12-54
- foreach command 12-55
- foreach_in_collection command 12-38
- full analysis (command overview) 2-33

G

- Gate-level SAIF 8-17
- get_attribute command 12-83, 12-88
- get_cells command 12-26

- get_clocks command 7-9
- get_designs command 12-26
- get_lib command 12-26
- get_lib_cells command 12-26
- get_lib_pins commands 12-26
- get_message_info command 12-20
- get_nets command 12-26
- get_pins command 12-26
- get_ports command 12-26
- getenv command 12-6
- getting help 2-15, 2-18
- getting started with PrimePower 2-1
- glitch power
 - analyzing 10-3
- graphical user interface (GUI) 4-1
 - closing windows 4-7
 - console window 4-11
 - docked and undocked windows 4-4
 - histogram window 4-14
 - menu commands 4-9
 - minimizing, maximizing windows 4-7
 - starting 2-3
 - starting and exiting 4-2
 - starting from pp_shell 4-2
 - status box 4-13
 - toolbars 4-8
 - top-level window 4-3
 - window types 4-8
- grouping cells 11-3
- GUI (graphical user interface) 4-1

H

- Handling Activity Data From Mixed-language Simulations 9-5
- Handling Large Activity Files 9-4
- Handling VCD Files With Non-module Scopes 9-7
- HDL Compiler (to read Verilog) 5-10
- help command 2-15

- Hierarchical Cells 6-14
- histogram window 4-14
- history command 12-16

I

- if 12-52
- if command 12-52
- implicit collection 12-38
- info body command 12-61
- info command 12-61
- interactive mode 1-7
- Internal Power 1-16
- internal power
 - attribute 1-20
 - defined 1-10
 - equations for summing 1-21
 - modeling 1-10, 1-21
 - rise and fall, separate 1-19
 - short circuit power 1-10
 - state-dependent, defined 1-18
- internal_power 1-16, 1-20
- interrupting commands 12-13
- introduction to PrimePower usage 2-1
- I_{sc} , short circuit current 1-10
- itr_var variable 12-39

L

- Leakage Power 1-12
- leakage power
 - cause of 1-9
 - equation 1-12
 - modeling 1-9
- lib object class attributes list 12-102
- lib_cell object class attributes list 12-104
- lib_pin class attributes list 12-106
- libraries
 - attributes
 - internal_power 1-20

- library
 - complex cells 8-29
- library data
 - reading 5-4
- library_rail 11-6
- licenses 2-3
- limitations A-3
- link_design command 5-19
- link_path variable 5-2
- list_attributes command 12-83
- list_designs command 5-17
- list_libraries command 5-18
- lists 12-47
 - Tcl commands to use with 12-47
- loading design data 5-4
- loading library data 5-4
- log file 2-5
- logical operators 12-34
- look-up tables ??–1-19
- loop, terminating 12-56
- loops, while 12-53

M

- man command 2-18
- manuals (description of PrimePower manuals) 2-21
- Mapping the Testbench Instance to the Design Module 9-4
- menu commands in GUI 4-9
- merge_saif 8-4, 8-20, 8-22
- merged NLDM 11-10
- messages
 - CMD-041 12-24
- messages, controlling the output 12-19
- modeling power
 - internal power 1-21
 - leakage power 1-9
 - switching power 1-22
- Modeling Wire Loads 6-2

- Monitoring the Power Off Signal 11-8
- Multiple NLDM Library 11-10
- Multi-VDD 7-21
- Multi-voltage Cells 11-6
- Multi-voltage Design Power Report 11-3
- Multi-voltage Libraries 11-10

N

- nesting commands 2-11, 12-51
- net object class attributes list 12-109
- Net Power Report 10-11
- net report 5-34
- net switching power equation 1-22
- New Power Report 10-6
- NLDM 11-10

O

- Object Attributes 12-82
- objects 12-24
- objects in designs 5-22
- online help 2-15
- operating conditions 2-31
 - min-max analysis 2-31
- operators
 - existence
 - existence operators 12-35
 - logical 12-34
 - pattern match filter 12-36
 - redirection 12-11
 - relational
 - relational operators 12-34
- Optional Reports 10-8
- overview 2-24
 - reading design data 2-26

P

- Parasitic Reports 6-22

- parse_proc_arguments command 12-74
- path-dependent power, defined 1-18
- path-dependent switching activity
 - path-dependent power 1-18
- pattern match filter operators 12-36
- pausing output from PrimePower 2-12
- Peak Power 9-20
- pin object class attributes list 12-112
- port
 - capacitance 6-21
 - report 5-31
- Power 8-45
- power
 - internal cell 1-20
 - leakage, modeling 1-9
 - switching power, defined 1-22
- power (leakage), analyzing 10-3
- power (normal), analyzing 10-3
- power (total), analyzing 10-3
- Power Analysis 8-1, 8-45, 9-10
- Power Analysis With SAIF 3-7
- Power Analysis With VCD 3-2
- Power Calculation Variables 8-45
- Power Compiler
 - interdependencies of inputs 8-29
- power modeling
 - internal power 1-20
 - leakage power 1-9
 - switching power 1-22
- Power Waveforms 9-20
- Power-gating Support 11-7
- pp_shell
 - backslash character 2-7
 - case-sensitivity 2-6
 - command result 2-10
 - command scripts 2-9
 - interactive command entry 2-6
 - multi -line commands 2-7
 - nesting commands 2-11
 - question mark prompt 2-7

- scripts 2-9
- starting the GUI 4-2
- using 2-6
- pp_vcd 3-4
- PrimePower
 - interactive mode 1-7
 - run options 1-7
- PrimePower installation 3-5
- PrimePower Script File 3-8
- PrimePower Simulation Flow 1-3
- primepower.fsdb file 10-12
- primepower.out file 10-12
- print_suppressed_messages command 12-20
- printenv command 12-6
- printvar command 2-13, 12-21, 12-22
- proc command 12-58
- proc_body command 12-61
- procedure
 - changing aspects 12-70
 - creating Tcl 12-58
 - defining attributes 12-69
 - disabling viewing contents 12-71
 - displaying body of Tcl 12-61
 - displaying contents (body) 12-61
 - modifying existing 12-69
 - preventing modification 12-71
- Processing Multiple Activity Files 9-7
- Propagating Switching Activity 8-28
- propagation of constants (case analysis) 8-37
- P_{sc}, short circuit power 1-10
- pwd command 12-6
- pwr_report_power_format 10-6

Q

- query_objects command 12-30
- question mark prompt (pp_shell) 2-7
- quoting values 12-50

R

- read_db command 5-4
- read_saif 8-4, 8-16, 8-17
- read_vcd 9-7
- reading design data 5-4
 - overview 2-26
 - Verilog 5-6
 - VHDL 5-12
- reading library data 5-4
- Reading SAIF 8-17, 8-20
- Reading the File 7-13
- redirect command 12-10
- redirecting command output 12-9
- redirection operators 12-11
- regexp command 12-65
- regular expressions
 - anchor 12-67
 - buses 12-67
- regular expresssions 12-65
- remove_case_analysis command 8-41
- remove_clock command 7-10
- remove_from_collection command 12-39
- remove_user_attribute command 12-83
- Removing Switching Activity 8-23
- Removing Switching Activity Annotation 8-23
- report commands (summary) 2-27
- report_attribute command 12-83
- report_bus command 5-37
- report_case_analysis command 8-40
- report_cell command 5-29
- report_clock command 5-28
- report_design command 5-26
- report_matched_point 8-16
- report_net command 5-34
- report_port command 5-31
- report_power 9-20, 10-6
- report_power -hier 10-7
- report_saif 8-30

- Reporting 7-22
- reporting net delays 7-16
- Reporting Power Calculation 7-22
- Reporting Power Rail Mapping 11-7
- Reporting Wire Load Models 6-17
- reports
 - changing filenames 10-8, 10-9, 10-12
 - pausing 2-12
- restore_session command 2-36
- Reviewing the Power Report 3-6
- rise and fall power 1-19
- RTL SAIF 8-16
- Running PrimePower 3-5

S

- SAIF File Based Analysis 8-14
- SAIF File Commands 8-17
- save_session command 2-36
- Scaling Waveform Data 10-14
- Scan Chain Power 10-20
- scan chain, disabling 8-34
- script
 - compression (gzip, bytecode) 2-9
 - overview 2-9
- scripts 12-14
 - batch mode 2-10
 - comment lines 12-14
 - continue or stop on error 12-15
 - file search path 12-16
 - logging executed commands 12-16
- scrolling (pausing output from PrimePower) 2-12
- SDC 7-11
- SDPD Switching Activity 8-28
- search_path variable 5-2
- Selecting the Wire Load Model 6-14
- separate rise and fall power, specifying 1-19
- session save and restore 2-36
- set command 2-12, 12-21
- set_case_analysis command 8-38
- set_current_rail 11-3
- set_load command 6-21
- set_message_info command 12-20
- set_operating_conditions 7-3
- set_rail_voltage 7-21
- set_report_opt command 10-8
- set_rtl_to_gate_name 8-16
- set_switching_activity 8-4, 8-10, 8-23
- set_user_attribute command 12-83
- setenv command 12-6
- Setting a Wire Load Minimum Block Size 6-12
- Setting a Wire Load Mode 6-12
- Setting a Wire Load Model 6-10
- Setting a Wire Load Selection Group 6-13
- sh command 12-6
- sh_command_abbrev_mode variable 12-4, 12-71
- sh_enable_page_mode variable 2-12
- sh_new_variable_message variable 12-24
- sh_new_variable_message_in_proc 12-24
- sh_new_variable_message_in_script 12-24
- short circuit current, I_{sc} 1-10
- short circuit power
 - defined 1-10
 - internal power 1-10
 - symbol for 1-10
- shortening commands 2-6, 12-3
- Smallest Block Size 6-15
- SNPS_TCLPRO_HOME 12-81
- SOLD (Synopsys Online Documentation) CD 2-23
- Solv-It answer database 2-22
- source command 2-9, 12-14
- SPDM 11-10
- SPDM Library 11-11
- Specifying Port/Net Loads 6-21
- starting PrimePower 2-1

- startup files 2-4
- state-dependent power
 - defined 1-18
 - leakage power 1-18
- Static Analysis 8-2
- static power 1-9
- Statistics 10-25
- Steps for Analyzing Power 3-4
- suppress_message command 12-20
- switch 12-56
- switch command 12-56
- Switching Activity 8-3
- Switching Activity Commands 8-8
- Switching Power 1-10
- switching power
 - defined 1-10
 - equation for 1-19, 1-22
- Synopsys Online Documentation (SOLD) CD 2-23
- .synopsys_pt.setup file 2-4
- system commands 12-6

T

- Tcl 12-2
 - arguments 12-60
 - autoload 12-63
 - commands to use with lists 12-47
 - create procedure 12-58
 - display procedure body (contents) 12-61
 - extensions in PrimePower 12-8
 - lists 12-47
 - other utilities 12-49
 - packages 12-63
 - parsing arguments 12-74
 - positional arguments 12-59
 - procedures 12-56
 - restrictions in PrimePower 12-8
 - special characters 12-5
 - TclPro 12-77

- TclPro
 - bytecode-compiled files 12-80
 - limitations 12-82
 - procheck 12-77
 - procomp 12-77, 12-80
 - prodebug 12-81
 - prodebug 12-77
- The SAIF File 3-8
- timing_save_pin_arrival_and_slack variable 12-112
- top-level GUI window 4-3
- tutorial 3-1

U

- Unannotated Nets 6-25
- Unmatched-States 9-14
- unset command 12-21
- unsuppress_message command 12-20
- usage overview for PrimePower 2-24
- user interface, graphical (GUI) 4-1
- utilities, Tcl 12-49

V

- values, quoting 12-50
- variable
 - man pages 2-20
 - setting 2-12
 - showing the current setting 2-13
- variables
 - command to list 12-22
 - command to see current value 12-22
 - itr_var 12-39
 - link_path 5-2
 - reference value of 12-21
 - search_path 5-2
 - sh_command_abbrev_mode 12-4, 12-71
 - sh_enable_page_mode 2-12
 - sh_new_variable_message 12-24
 - sh_new_variable_message_in_proc 12-24

- sh_new_variable_message_in_script 12-24
- timing_save_pin_arrival_and_slack 12-112
- user-defined 12-22
- using 12-22
- Vector Free Power Analysis 3-11
- Vector-free/Default Activity 8-5
- Vector-less/Default Activity 8-5
- Verifying the Wire Capacitance 6-24
- Verilog
 - reading 5-6
 - reading with HDL Compiler 5-10
- VHDL
 - reading 5-12
- View Power Waveform 3-6
- Viewing Waveform Data 10-14
- virtual clock 7-8

W

- warning messages

- man page 2-20
- which command 12-6
- while command 12-53
- while loops 12-53
- wildcard characters 12-31
- windows in GUI 4-8
- Wire Load Information 6-23
- wire load model 1-22
 - overview 2-32
 - report 6-22
- Wire Load Model Modes 6-7
- write_saif command 8-25
- Writing SAIF 8-25
- Writing SAIF File 8-25

Z

- Zero-delay VCD Files 9-8