PrimePower
User Guide

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

©2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Preface

This preface includes the following sections:

About This Guide

This manual describes the Synopsys PrimePower tool, its methodology, and its use. PrimePower is a static and 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 contains 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 design or verification engineers who need to perform signoff power and multivoltage design analyses.

Related Publications

For additional information about the PrimePower tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

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

- IC Validator
- Design Compiler®
- PrimeTime®
- Power Compiler™
- Library Compiler™

PrimePower Tutorials

There are several tutorials available to help you learn about averaged, vector-based, and time-based power analysis using PrimePower.

To access the tutorials, enter the following path:

```
$INSTALL_DIR/doc/primepower/
```

Release Notes

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

To see the *PrimePower Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select PrimePower, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl 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

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

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

https://solvnet.synopsys.com

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

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

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 support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- 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 https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at https://www.synopsys.com/support/global-support-centers.html

1

Introduction to PrimePower

The Synopsys PrimePower tool is a power analysis solution that accurately analyzes full-chip power dissipation of cell-based designs throughout the design implementation process, from early estimation to signoff.

PrimePower provides accurate power analysis reports for SoC designers to perform timely design optimizations to achieve power targets; the supported power types include peak power, average power, clock network power, leakage power, and multivoltage power.

By closely integrating with the PrimeTime tool, the golden industry standard for timing and signal integrity signoff, PrimePower expands the PrimeTime timing and signal integrity analysis solution to deliver accurate dynamic and leakage power analysis.

This chapter describes PrimePower and its features. It contains the following topics:

PrimePower Capabilities

PrimePower performs vector-free and vector-based peak power and average power analysis. The input waveform vectors used by PrimePower are either RTL-level or gate-level simulation results in the Value Change Dump (VCD) format, Fast Signal Database (FSDB) format, or Switching Activity Interchange Format (SAIF).

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 database format (.db) library. PrimePower also supports nonlinear power model (NLPM) libraries. It calculates the power for a circuit at the cell level and reports the power consumption at the chip, block, and cell levels.

When power analysis is complete, you can view design data and analysis results in the graphical user interface (GUI), including power maps and waveforms, for visual debugging.

PrimePower provides the following features:

- Averaged Power Analysis

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

- Time-Based Power Analysis

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

  PrimePower uses an event-driven algorithm to calculate power for each event. For time-based power analysis, the tool generates detailed average and peak power waveforms, as well as power reports.

- Multivoltage Power Analysis

  PrimePower supports power calculation for cells with multiple power supplies using either UPF, non-UPF, power domains, or power rails to specify power intent. Scaling the power consumption per power net by a specified voltage is also supported.

- Clock Network Power Analysis

  PrimePower supports activity propagation through the clock network to calculate power consumption of the clock network in the averaged power mode.

- Distributed Peak-Power Analysis

  PrimePower supports distributed analysis of peak power using the distributed multi-scenario analysis (DMSA) infrastructure and PrimeTime distributed timing analysis to help reduce memory resource usage and excessive runtime when analyzing a large chip design.

- Native Distributed Analysis in Time-Based Mode

  PrimePower supports a multi-process distributed processing technique to help reduce memory resource usage and excessive runtime when analyzing a full-chip design with multiple gate-level FSDB files.

- Cycle-Accurate Peak Power Analysis

  PrimePower uses the RTL switching activity data to calculate power in the cycle-accurate peak power analysis mode. In this mode, the tool calculates the power per event accurately, and generates power waveforms at the clock-cycle resolution. Use the results to determine the cycle during which maximum power consumption occurs in a design.

- Multirail Power Analysis

  PrimePower supports multirail analysis to check the contribution of each rail in a single run.

- Cell Electromigration Analysis

  PrimePower supports cell electromigration analysis to identify cells whose toggle rates exceed the maximum allowable toggle rate based on the current types (either average, rms or peak) listed in the electromigration table. This allows you to identify and resolve possible electromigration issues early in the flow.

- PST-Based Power Analysis

  A power state table (PST) is a UPF construct that defines the legal combinations of voltage values and states of the power and ground supplies for all supply sets in the design. PrimePower supports UPF PST-based power analysis in the averaged mode.

- Using PrimePower With PrimeTime

  You can invoke PrimePower for power calculation within the PrimeTime tool. The integration of timing, signal integrity, and power eliminates the redundant set-up and calculation steps required when using separate standalone tools.

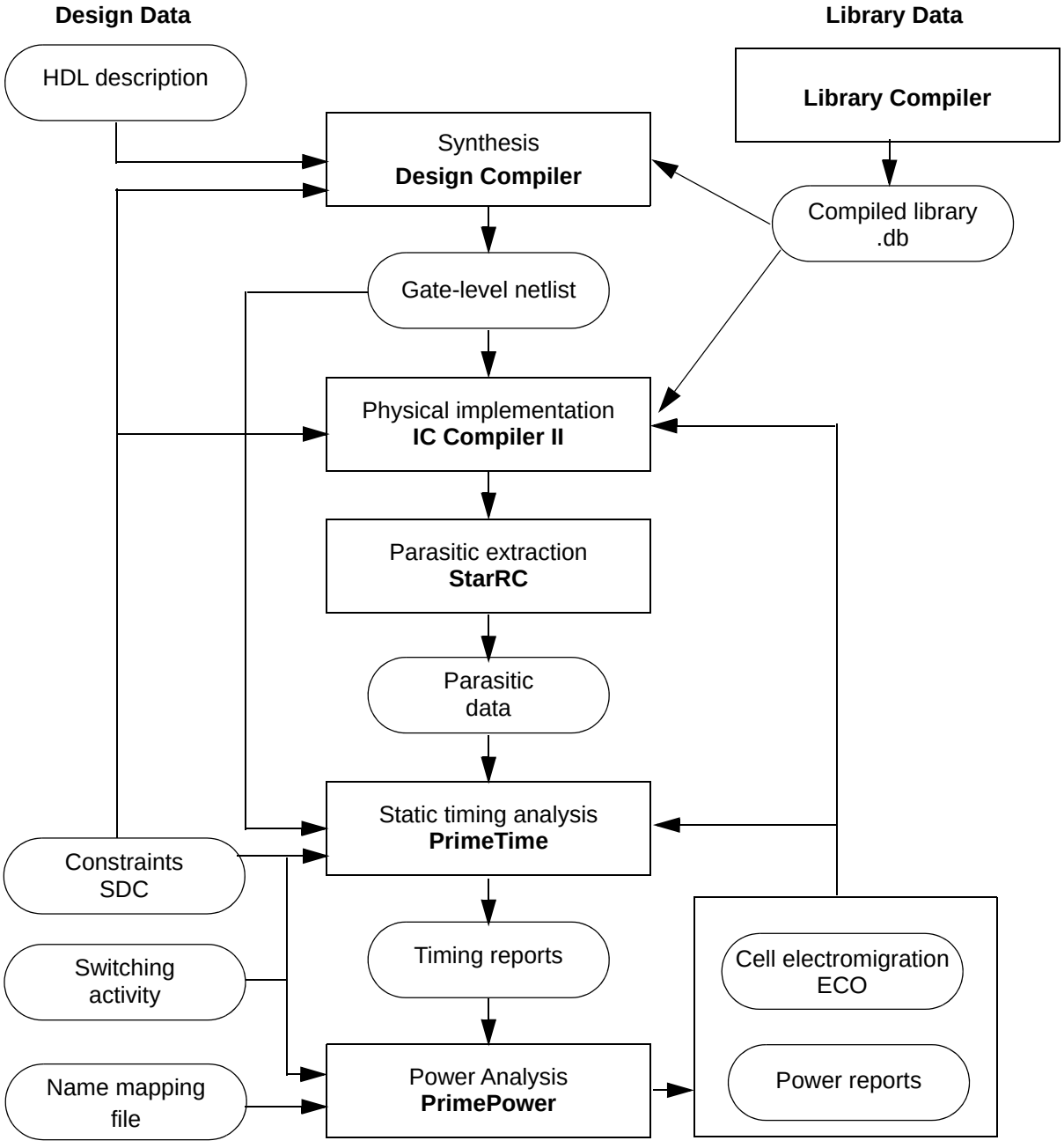
- RedHawk Analysis Fusion in IC Compiler II

  The RedHawk power integrity solution is integrated with the IC Compiler II implementation flow through the RedHawk Analysis Fusion interface. PrimePower supports the generation of the power consumption and static timing (STA) data to perform voltage drop analysis on the power and ground network.

Using PrimePower in the Implementation Flow

PrimePower is part of the Synopsys Galaxy Design Platform, and shares many of the same libraries, databases, and commands with other Galaxy tools, such as Design Compiler and IC Compiler II, as shown in Figure 1-1.

Figure 1-1 Using PrimeTime Suite in the Implementation Flow



Using PrimePower With Other Galaxy Tools

The PrimePower power analysis tool is designed to work with other Galaxy implementation tools, such as Design Compiler, PrimeTime, and IC Compiler II. They are compatible in the following ways:

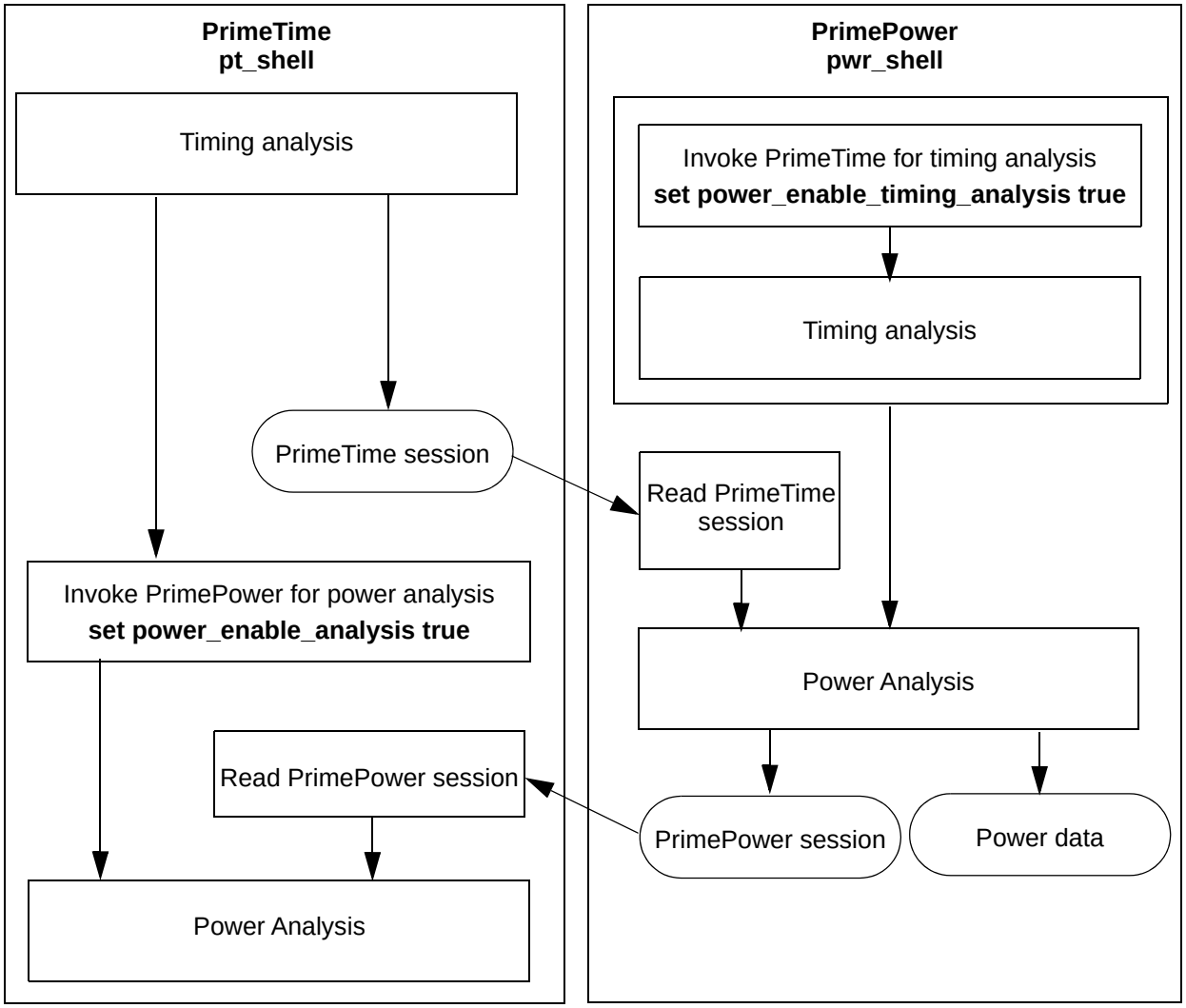
- They use the same logic libraries and read the same design data files in .db and .ddc formats.
- They share many commands (such as `link_design` or `create_clock`) that are identical or similar in operation.
- They share the same delay calculation algorithms and generally produce identical delay results.
- PrimePower provides timing and instance power data for the RedHawk Analysis Fusion feature in the IC Compiler II tool.
- PrimePower provides power and cell electromigration reports that can be used as electromigration engineering change order (ECO) guidance for current violation fixing and power optimization.

Using PrimePower With PrimeTime

PrimePower works closely with the PrimeTime tool. Both tools share many of the same libraries, databases, and commands, and support power and timing analysis when the required licenses are available.

Figure 1-2 explains how this integrated timing and power analysis functionality interface works between PrimePower and PrimeTime. You can invoke the PrimeTime timer for timing analysis without switching to pt_shell when a valid PrimeTime license is available. PrimePower power analysis functionality is also available in PrimeTime through this integrated power analysis flow interface

Figure 1-2 The Integrated Analysis Flow Between PrimePower and PrimeTime



You can enable PrimeTime timing analysis within pwr_shell by setting the `power_enable_timing_analysis` variable to `true`. Similarly, You can enable PrimePower analysis within pt_shell by setting the `power_enable_analysis` variable to `true`. Then you can perform both power analysis and timing analysis in the same session.

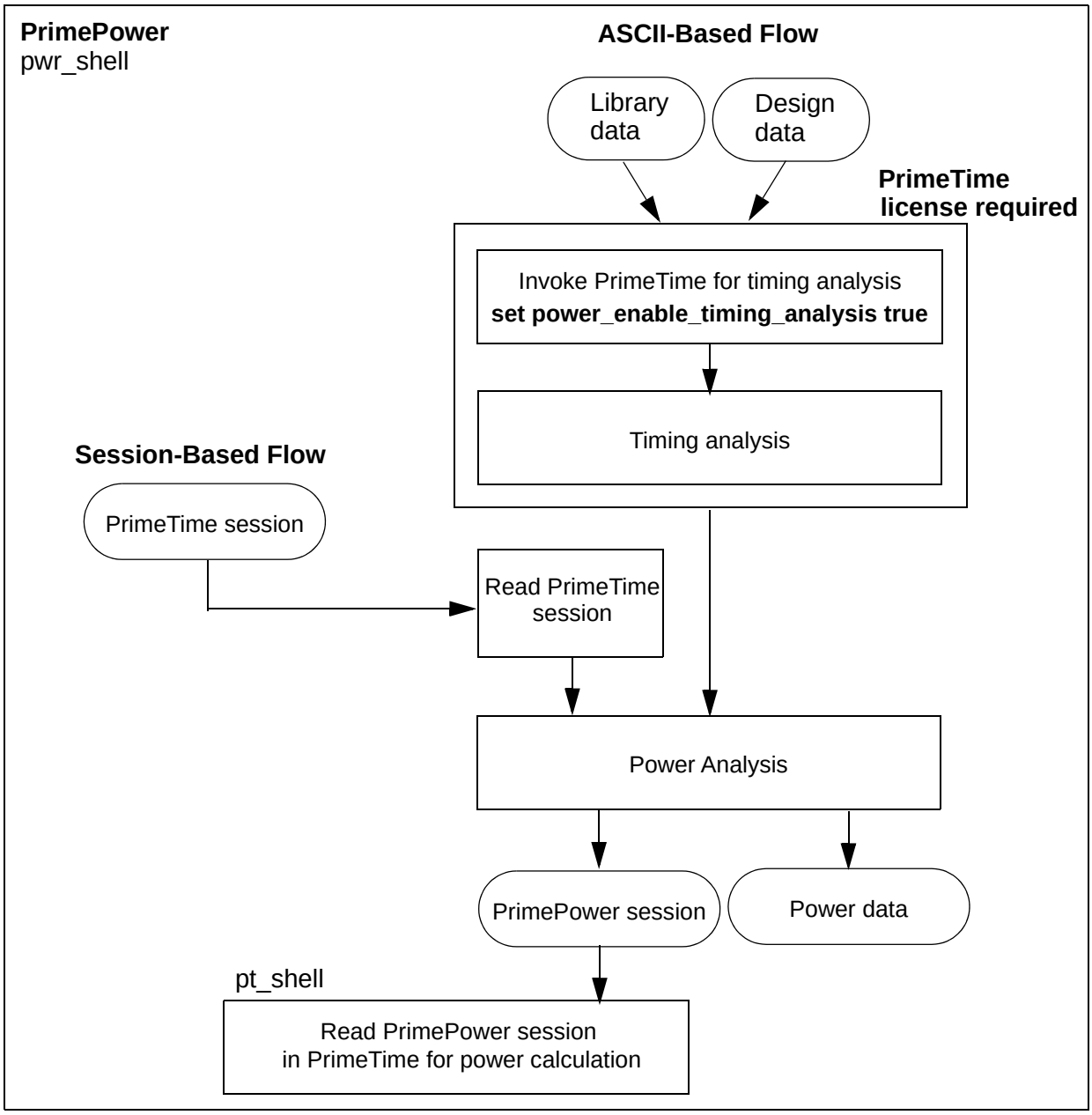
The integrated power and timing analysis flow provides flexibility in using PrimePower and PrimeTime licenses. In either tool, performing timing analysis requires a PrimeTime license and performing power analysis requires a PrimePower license.

PrimePower Session-Based and ASCII-Input Flows

Figure 1-3 illustrates the analysis flows that are available in PrimePower to calculate power dissipation of the design.

- In the session-based flow, you can restore the PrimeTime saved session data by running the `restore_session` command. The saved session contains PrimeTime commands, configuration settings, design data, timing data, and so on. This allows you to use the PrimeTime timing data for power analysis without rerunning the timing analysis in pwr_shell.
- In the ASCII-input-based flow, you start power analysis with the required design and library data, including the .db file, the switching activity file, and the SDC file. You can invoke PrimeTime to generate the necessary timing data for power analysis when a valid PrimeTime license is available.

Figure 1-3 PrimePower Analysis Flows



Power Modeling Overview

This section describes power modeling in the following topics:

- Leakage Power
- Dynamic Power, which includes internal power and switching power
- Generating Power Models

For power analysis, your Synopsys library must contain power models for all of the cells. The NLPM power models contain tables that PrimePower uses to calculate leakage power and internal power.

The tool calculates switching power based on the voltage, netlist capacitance, and switching of the nets. The leakage and dynamic power calculation results are used for peak power analysis and average power analysis.

Leakage Power

Leakage power is the power dissipated by a cell when it is not switching—that is, when it is inactive or static.

Leakage power is dissipated in several ways. Most of the leakage power dissipation results from source-to-drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Leakage power is also dissipated when current leaks between the diffusion layers and the substrate. The leakage power is state- and voltage-dependent. These types of leakage power are referred to as intrinsic leakage.

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:

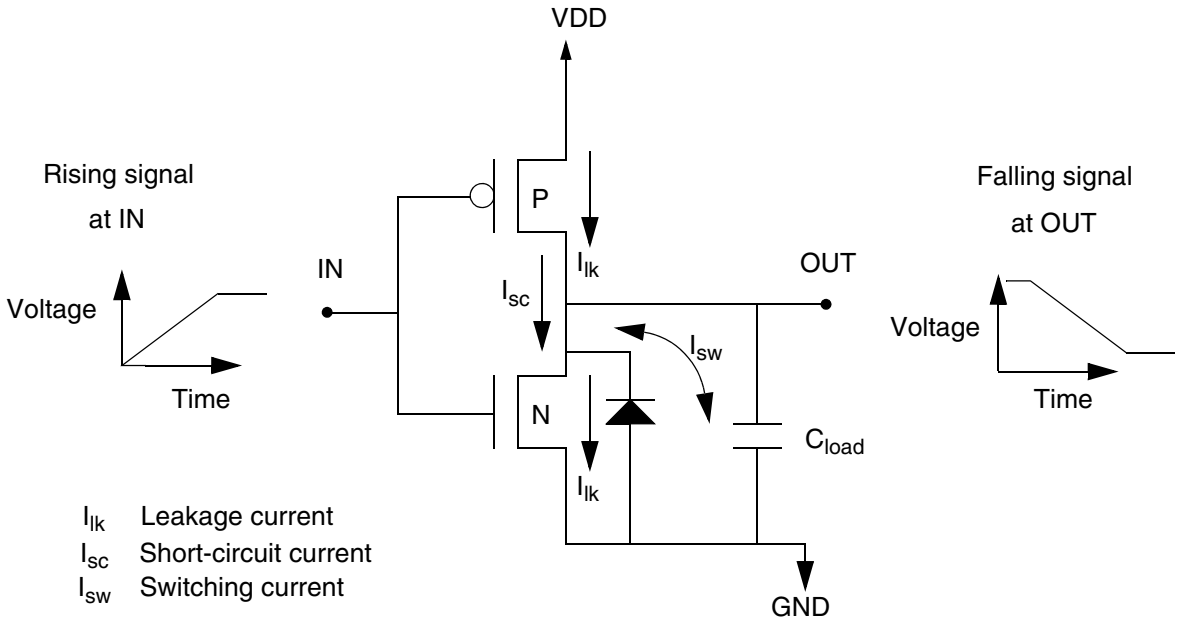
- Internal Power
- Switching Power

Internal Power

Internal power is the dynamic power dissipated within the boundary of a cell. It includes the power dissipation due to charging or discharging of capacitances internal to the cell during switching; and the power dissipation due to the momentary short circuit between the P and N transistors of a gate while both are turned on.

Figure 1-4 considers a simple gate to show the cause of short-circuit power. 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 VDD to GND, causing the dissipation of short-circuit power (P_{sc}).

Figure 1-4 Components of Power Dissipation



For circuits with fast transition times, short-circuit power can be low. However, for circuits with slow transition times, short-circuit power can account for more than 50 percent of the total power dissipated by the gate. Short-circuit power is affected by the dimensions of the transistors and the load capacitance at the gate's output.

For simple library cells, internal power consumption is primarily due to short-circuit power. For complex library cells, however, the dominant source of internal power might be due to the charging and discharging of internal capacitance.

Library developers can model internal power by using the internal power library group.

For more information about modeling internal power, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

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 is 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.

Generating Power Models

PrimePower provides an automated mechanism to store power data in a model. The power model generated by the tool can be instantiated for a faster chip-level power analysis and is based on the Extracted Timing Model (ETM) feature of the PrimeTime tool. The power model is generated by incorporating power information into the ETM. Perform timing analysis and power analysis on your design before generating the power model. Based on the result of the analysis, use the `extract_model -power` command to generate the power models that contain both timing and power data.

When generating the power model from the gate-level design, PrimePower associates the dynamic power of the gate-level design with the clock pins of the generated power model. The dynamic power of the gate-level design is mapped into the internal power of the generated power model within the tool. The internal power of the model can change based on the clock frequency at which the power model is instantiated. To get similar results in terms of power from the generated power model and the gate-level design, the power model must be instantiated at the same clock frequency as the clock that powers the power model.

The tool annotates the switching activity of internal clock pins of ETMs during power analysis of hierarchical designs. This feature is useful for analyzing the power consumption of complex macro cells, which is a function of internal clock frequencies. This is supported in both the averaged and time-based power analysis modes. For more information, see Reporting Switching Activity.

Limitations in Generating Power Models

The limitations when generating the power models are:

- Multiple modes of operation are not supported in the generated power model.
- Block scope checking for power is not supported.
- The `extract_model -power` command writes out the power model only in the .lib format. To generate the .db equivalent of the model, use the Library Compiler or Design Compiler tool.

2

Getting Started

PrimePower supports two types of power analysis modes—averaged and time-based. In each of the modes, the tool supports various options to suit different types of designs and applications. To perform power analysis for a design, select the required power analysis mode and the specific options supported of the mode.

The user interface supports variables, commands, and command options for performing power analysis. Provide the switching activity information in the SAIF, VCD, or FSDB file format. Irrespective of the mode you select and the options you choose, power analysis is performed when you run the `update_power` command.

This chapter describes the basic information for running power analysis. It contains the following topics:

- Saving and Restoring PrimePower Sessions

Configuring the PrimePower Work Environment

When you start a PrimePower session, the tool executes the commands in the PrimePower setup files. In these files, you can initialize parameters, set variables, specify the design environment, and configure your preferred working options.

The setup files have the same name, .synopsys_pwr.setup, but reside in different directories. PrimePower reads the files from three directories in the following order:

1. The Synopsys root directory ($SYNOPSYS/admin/setup)

   This system-wide setup file contains

   - System variables defined by Synopsys
   - General PrimePower setup information for all users at your site

   Only the system administrator can modify this file.

2. Your home directory

   In this user-defined setup file, you can specify your preferences for the PrimePower working environment. The variable settings in this file override the corresponding variable settings in the system-wide setup file.

3. The current working directory from which you started PrimePower

   In this design-specific setup file, you can specify the project or design environment. To use this file, you must invoke PrimePower from this directory. The variable settings in this file override the corresponding variable settings in the user-defined and system-wide setup files.

To suppress the execution of all .synopsys_pt.setup files, start PrimePower by using the pwr_shell command with the `-no_init` option.

The following is an example of the .synopsys_pwr.setup file:

```
%> vi .synopsys_pr.setup
      echo "hi-global pr setup file from the LOCALDIR"
      set_mtcmos_switch_lib_cell HEAD2X16_HVT \
           -input_pg_pin VDDG -output_pg_pin VDD \
           -pin_state { SLEEP 0 } -pin_state_name FULL_ON \
           -on_state_resistance 1
      report_mtcmos_switch_lib_cell HEAD2X16_HVT
      echo "END: hi-global pr setup file from the LOCALDIR"
```

For more information, see the *Using Tcl With Synopsys Tools* manual, available on SolvNet.

Power Analysis Inputs

PrimePower requires the following data for performing power analysis:

- Logic library: A cell library containing timing and power characterization information for each cell. The supported formats are `nlpm` and `ccs` for timing data, and `nlpm` for power data.
- Gate-level netlist: A flat or hierarchical gate-level netlist in Verilog, VHDL, or Synopsys database format, containing leaf-level instantiation of the library cells.
- Design constraints: An SDC file containing design constraints to calculate the transition time on the primary inputs and to define the clocks.
- Switching activity: The design switching activity information for averaged power analysis or accurate peak power analysis. The switching activity information can be specified in an event file in the VCD for FSDB format.
- Net parasitics: A parasitics file (SPEF) containing net capacitances for all the nets.

Starting a PrimePower Session

PrimePower operates in a windows environment on a Linux platform. Before starting the tool, make sure the path to the bin directory is included in your $PATH variable.

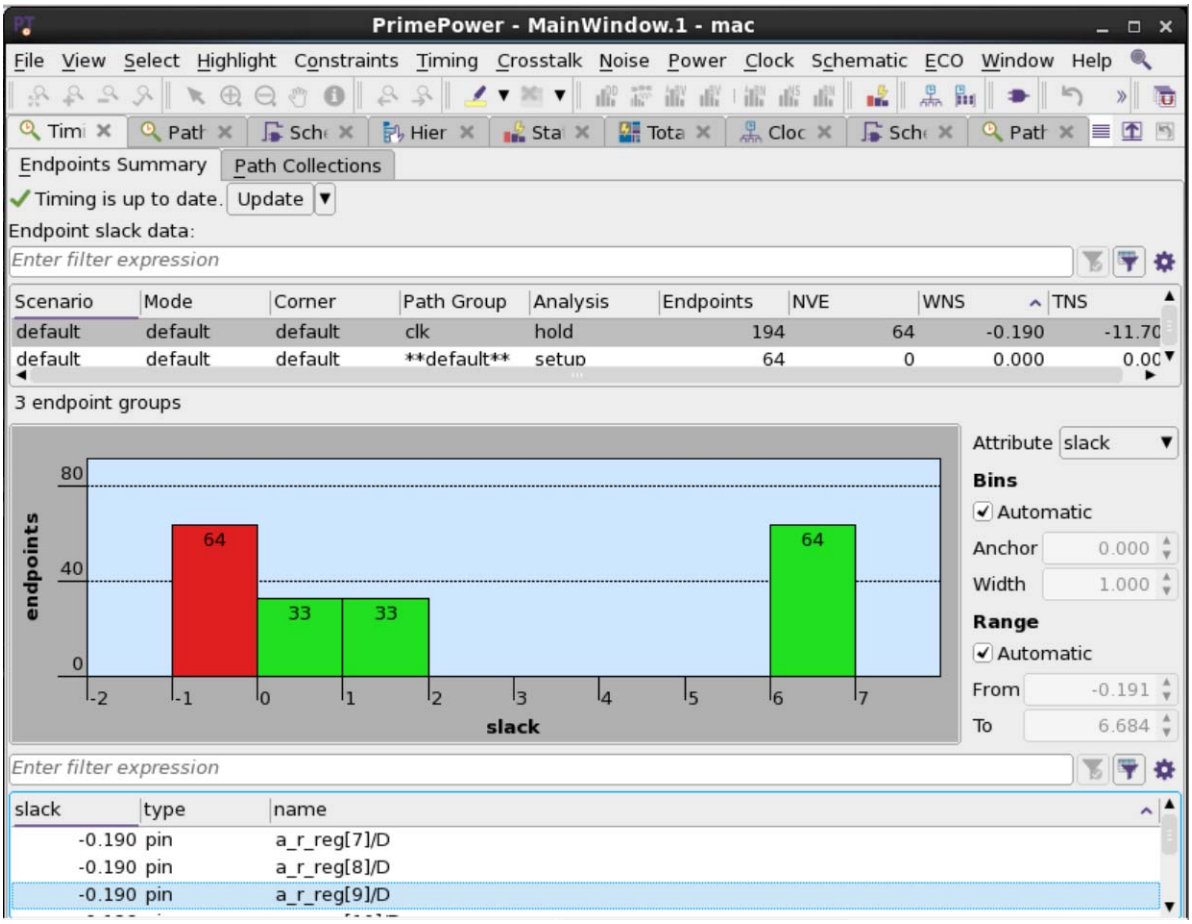
- To start the tool in the pwr_shell command-line interface, enter the `pwr_shell` command at the Linux prompt:

```
% pwr_shell

                              PrimePower

            Version O-2018.06 for linux64 - June 13, 2018
                  Copyright (c) 2018 Synopsys, Inc.
  This software and the associated documentation are proprietary ...

pwr_shell>
```

- To start the PrimePower graphical user interface, enter the `pwr_shell -gui` command at the Linux shell:

```
% pwr_shell -gui
```

Figure 2-1 PrimePower Graphical User Interface



You can include other options on the command line when you start the tool. For example,

- `-f` *`script_file_name`* to execute a script
- `-x` to execute a PrimePower shell command
- `-display` *`terminal_name`* to display the GUI on a different terminal from the one you set with the DISPLAY environment variable
- `-h` to display a list of the available options without starting the tool

At startup, pwr_shell does the following tasks:

1. Creates a command log file. See Command Log Files for more information.
2. Runs the setup files. See Setup Files for more information.

3. Runs any script files or commands specified by the `-f` and `-x` options, respectively, on the command line.
4. Displays the program header and pwr_shell> prompt in the shell in which you started pwr_shell.

For a complete list of startup options, see the command man page. For more information about using Tcl commands, see the *Using Tcl With Synopsys Tools* manual, available on SolvNet.

Command Log Files

The command log file records all pwr_shell commands processed by the tool for the current session, including those in the setup files. By default, the tool writes the command log to a file, named pwr_shell_command.log, in the directory from which you invoked pwr_shell. You can change the name of the command log file by setting the `sh_command_log_file` variable in your setup files.

For more information about the command log file, see the *Using Tcl With Synopsys Tools* manual, available on SolvNet.

Setup Files

PrimePower runs the tool's default setup file, named .synopsys_pwr.setup. You can have up to three setup files saved in the following directories with a precedence order from high to low:

- Settings in the current working directory where you start the tool
- Settings in your home directory
- Settings in the installation (Synopsys root) directory

Entering pwr_shell Commands

Based on the tool command language (Tcl), the PrimePower command language supports variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the pwr_shell> prompt
- Enter individual commands interactively on the console command line in the GUI
- Run Tcl command scripts, which are text files containing pwr_shell commands

When entering a command, an option, or a file name interactively, you can minimize your typing by pressing the Tab key when you have typed enough characters for automatic name completion. If the characters you typed could be used for more than one name, the tool lists the qualifying names, from which you can select by using the arrow keys and the Enter key.

For general information about the Tcl command-line interface, see the *Using Tcl With Synopsys Tools* manual, available on SolvNet.

Getting Help While Running PrimePower

PrimePower provides a variety of user-assistance tools.

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

- Command-Line Help
- PrimePower Man Pages
- Displaying the List of Keyboard Shortcuts

Command-Line Help

The following online information resources are available while you are using pwr_shell:

- To display a brief description of a pwr_shell command,
  - Enter `help` followed by the command name:

    ```
    pwr_shell> help command_name
    ```

- To display a list of command options and arguments with a specified pwr_shell command,
  - Enter the command name followed by the `-help` option:

```
pwr_shell> command_name -help
```

PrimePower Man Pages

To view, search, and print a man page for commands, variables, and error messages in the man page viewer,

1. Choose Help > Man Pages.

   The man page viewer appears and displays a list of links for the different man page categories.

2. Click the category link for the type of man page you want to view: Commands, Variables, or Messages.

   The contents page for the category displays a list of title links for the man pages in that category.

3. Click the title link for the man page you want to view.

To display a man page in pwr_shell and in the console log view in the GUI,

- Enter man followed by the command or variable name in pwr_shell:

  ```
  pwr_shell> man command_or_variable_name
  ```

You can also display man pages in the man page viewer by using the `man` command on the console command line in the GUI.

Displaying the List of Keyboard Shortcuts

You can view a report of the keyboard shortcuts for Tcl commands and commands on menus in the active window. PrimePower displays the keyboard shortcut report in a report view.

To display the report of keyboard shortcuts for the Tcl commands and commands on menus in the active window,

Choose Help > Report Hotkey Bindings.

Working With Licenses

When you start PrimePower, the tool automatically checks for an available PrimePower license. To learn how to manage licenses, see the following topics:

- Getting, Listing, and Releasing the Licenses

Getting, Listing, and Releasing the Licenses

When you invoke PrimePower, the Synopsys Common Licensing software automatically checks out the appropriate license. If tasks later in the session require additional licenses, you can use the `get_license` command to check out those licenses. This ensures that each license is available when you are ready to use it. After a license is checked out, it remains checked out until you release it or exit PrimePower.

To view the licenses that you currently have checked out, use the `list_licenses` command. For example,

```
pwr_shell> list_licenses

Licenses in use:
         PrimePower (1)
```

To release a license that is checked out to you, use the `remove_license` command. For example,

```
pwr_shell> remove_license PrimePower
```

Enabling License Queuing

PrimePower has a license queuing functionality that allows your application to wait for licenses to become available if all licenses are in use. To enable this functionality, set the `SNPSLMD_QUEUE` variable to `true`. The following message is displayed:

```
Information: License queuing is enabled.
```

When you have enabled the license queuing functionality, you might run into a situation where two processes are waiting indefinitely for a license that the other process owns. To prevent this situation, set the `SNPS_MAX_WAITTIME` environment variable and the `SNPS_MAX_QUEUETIME` environment variable. You can use these variables only if the `SNPSLMD_QUEUE` environment variable is set to `true`.

The `SNPS_MAX_WAITTIME` variable specifies the maximum wait time to acquire a license to start an application, such as the license for starting pwr_shell. Consider the following scenario:

The queuing functionality places this last job in the queue for the specified wait time. The default wait time is 259,200 seconds (or 72 hours). If the license is not available after the predefined time, you see the following message:

```
Information: Timeout while waiting for feature 'PrimePower'
```

The `SNPS_MAX_QUEUETIME` variable specifies the maximum wait time for checking out subsequent licenses within the same pwr_shell process. You use this variable after you have successfully checked out the first license to start pwr_shell. Consider the following scenario:

You have already started the tool and are running a command that requires a PrimePower license. The queuing functionality attempts to check out the license within the specified wait time. The default is 28,800 seconds (or eight hours). If the license is still not available after the predefined time, you might see a message similar to the following:

```
Information: Timeout while waiting for feature 'PrimePower'.
```

As you take your design, the queuing functionality might display other status messages, as shown below:

```
Information: Successfully checked out feature 'PrimePower'.
Information: Started queuing for feature 'PrimePower'.
Information: Still waiting for feature 'PrimePower'.
```

See Also

- Working With Licenses

Enabling Multicore Processing

Several PrimePower functions support multicore processing. Multicore processing improves turnaround time by performing multiple tasks in parallel.

Use the `set_host_options -max_cores` command to configure multicore processing in PrimePower. PrimePower supports up to 4 cores per license.

For example, if your machine has two CPUs and each CPU has three cores, specify four as the maximum number of threads:

```
pwr_shell> set_host_options -max_cores 4
```

By default, the number of cores specified by the `-max_cores` option applies to all commands that support multithreading.

Note:

A single machine has one or more CPUs and each CPU has one or more cores. The total number of cores available for processing on a machine is the number of CPUs multiplied by the number of cores in each CPU.

PrimePower Analysis Flow

Table 2-1 shows the steps for performing power analysis, as described in the following sections:

Table 2-1 PrimePower Analysis Flow

Steps	Tasks	Commands/Variables	Related Topics
1	Perform vector analysis	`write_activity_waveforms`	Performing Vector Analysis

2	Read in the design data, which includes a gate-level netlist and associated logic libraries	`set search_path`
		`set link_path`
		`read_verilog`
		`link_design`
			Reading the Design Data and Logic Library

3	Specify variables to control the power analysis results	`power_limit_extrapolation_range`, `set_operating_conditions`, `power_use_ccsp_pin_capacitance`	Specifying Variables
4	Get timing analysis data by invoking PrimeTime timer or restoring a PrimeTime session	`update_timing`, `restore_session`	Timing Analysis
5	Check the quality of the power data	`check_power`	Checking for Potential Errors That Might Affect Power Accuracy
6	Specify the power analysis mode to be averaged or time_based	`power_analysis_mode` `averaged | time_based`	Selecting the Power Analysis Mode
7	Check the quality of the activity data	`check_activity`	Checking Activity Data
8	Annotate switching activity data	`read_vcd`, `read_saif`, `read_fsdb`, `set_switching_activity`, `set_case_analysis`	Annotating Switching Activity
9	Specify options for power analysis	`set_power_analysis_options`	Specifying Options for Power Analysis

Table 2-1 PrimePower Analysis Flow (Continued)

Steps	Tasks	Commands/Variables	Related Topics
10	Calculate power consumption of the design	`update_power`	Performing Power Analysis
11	Generate power reports	`report_power` `report_power_consumption`	Generating Power Reports
12	Save the PrimePower session	`save_session`	Saving and Restoring PrimePower Sessions

Performing Vector Analysis

If an event-based activity file (either in VCD or FSDB format) is available, you can qualify the activity data before reading the design data using the vector analysis feature. During vector analysis, the tool plots the average toggle rate per interval of the signals in the activity file so that you can identify the time windows of high activity likely to produce peak power. Use the `write_activity_waveforms` command to generate activity waveforms from the activity file.

For more information, see Performing Vector Analysis.

Reading the Design Data and Logic Library

The tool supports netlists in Verilog, VHDL, .db, .ddc, and Milkyway formats. The logic library must be in the .db (Synopsys database) format. Design data includes parasitic information, such as port and net loads, wire load models, back-annotated parasitics, and transition time information. For averaged power analysis, the design data includes the design constraints compiled with the PrimeTime tool in a Synopsys Design Constraints (SDC) file.

For more information, see the PrimeTime documentation.

NLPM Data

PrimePower supports NLPM data. For more information about NLPM power libraries and how to generate them, see the Library Compiler documentation.

Setting Operating Modes for Power Models

You can define operating modes, such as read and write modes, for a power model by specifying the mode definition in the library. The following example uses the `mode_name` and `mode_value` attributes (in bold) to define conditional data modeling in the `mode_definition` group.

```
library (mylib3) {
  delay_model : table_lookup;
  cell(cell_3) {
    pg_pin(VDD) {
      voltage_name : VDD;
      pg_type : primary_power;
    }
    pg_pin(VSS) {
      voltage_name : GND;
      pg_type : primary_ground;
    }
    mode_definition(rw) {
      mode_value(read) {
      when : "A1";
      sdf_cond : "A1 == 1";
    }
      mode_value(write) {
      when : "!A1";
      sdf_cond : "A1 == 0";
    }
  }
  pin(A1) {
    direction : input;
    related_power_pin : VDD;
    related_ground_pin : VSS;
  }
  leakage_current() {
    mode(rw, read);
    pg_current(VDD) {
      value : 1.0;
    }
    pg_current(VSS) {
      value : -2.0;
    }
    gate_leakage(A1) {
      input_high_value : 0.5;
      input_low_value : -0.5;
  }}
```

For more information, see the Library Compiler documentation.

Specifying Variables

PrimePower provides the following variables for performing power analysis.

Table 2-2 Variables for Power Analysis

Variables	Description
`power_limit_extrapolation_range`	By default, PrimePower extrapolates indefinitely if the data point for internal power lookup is out of range. When set to `true`, the tool limits the extrapolation. The NLPM power table stops extrapolation at one additional index grid. The NLPM power table stops extrapolation at the specified range. The default is `false`. If there are many high-fanout nets, such as clock or global reset nets, you might want to limit the extrapolation for more accurate power values.
`set_operating_conditions`	Specifies the operating conditions for power analysis. When specified, PrimePower uses the appropriate set of parameter values from the logic library. These parameter values generally include fabrication process, operating temperature, and power supply voltage as characterized by the ASIC vendor.
`power_use_ccsp_pin_capacitance`	Considers the Miller Effect when deriving the capacitive load for power analysis using deep sub-micron and low-voltage technologies. When set to `true`, the capacitive load used for power analysis is more pessimistic because it includes the gate-to-source and gate-to-drain capacitance contributions due to the Miller Effect. When set to `false` (the default), the capacitive load is calculated by averaging the C1 and C2 values derived for timing analysis.

Timing Analysis

PrimePower uses the timing data, such as transition time, to calculate accurate power and time windows for averaged power and current waveforms.

In pwr_shell, you can run timing update by enabling PrimeTime timing analysis with the `power_enable_timing_analysis` variable, or read in a PrimeTime saved session. Timing analysis includes the signal integrity effects if you enable the signal integrity feature.

Note:

Perform timing analysis using the `update_timing` command before specifying the switching activity for power. This maximizes performance by preventing any additional timing updates triggered by the activity annotation commands.

A PrimeTime license is required for running PrimeTime timing analysis in PrimePower.

Checking for Potential Errors That Might Affect Power Accuracy

Before calculating power or reporting power, use the `check_power` command to identify potential power calculation problems that might affect your results. This command checks the design structure for potential power violations.

By default, the `check_power` command performs the checks defined by the `power_check_defaults` variable if no option is specified. To update your list of default checks, update this variable or use the `-override_defaults` option.

The `-override_defaults`, `-include`, and `-exclude` checklist options define the power checks to be used. When you specify the `check_power` command, the tool performs `out_of_table_range`, `missing_table`, `no_arrival_window`, `no_base_clock`, and `missing_function` checks. By default, PrimePower performs the `out_of_table_range` and `missing_table` checks.

Selecting the Power Analysis Mode

Use the `power_analysis_mode` variable to select the power analysis mode. Set this variable before using any of the power commands, as follows:

```
set_app_var power_analysis_mode averaged | time_based
```

If you do not set this variable, by default, PrimePower performs averaged power analysis.

The tool loses the switching activity and power information if you modify the value of the `power_analysis_mode` variable after power analysis. To perform power analysis, you must reread the switching activity data.

Power Analysis Techniques

In CMOS circuits, the statistical average power analysis is performed early in the design flow for total power consumption. Accurate event-based peak power analysis for each event is performed closer to the signoff for voltage (IR) drop analysis. Both analysis techniques require design switching activity on every node. Activity is defined as how often a net switches with respect to a specified clock period.

PrimePower performs both averaged power analysis and peak power analysis. The tool calculates static and dynamic power for both power analysis types.

Specifying Switching Activity Data

Specify the switching activity to annotate the activity information. The switching activity source can be a gate-level VCD file, a RTL VCD file, a SAIF file, or an FSDB file. The activity can also be defined by the user-defined commands for annotation, such as the `set_switching_activity` and `set_case_analysis` commands.

If the switching activity is obtained from an RTL simulation, specify a name mapping file to match the RTL activity file names to the corresponding gate-level objects. The default name mapping performed in the Design Compiler tool is correctly matched using the built-in name mapping capability in PrimePower.

When the switching activity information is available, you must annotate this information appropriately on the design objects, before you can use the switching activity information for power analysis.

For more information about specifying net switching activity information for averaged power analysis and time-based power analysis, see Annotating Switching Activity.

For more information about name mapping, see Name Mapping.

Specifying Options for Power Analysis

After selecting the power analysis mode, use the `set_power_analysis_options` command with the options to control the behavior of the power analysis mode. While some options of this command can be used for all the modes, other options are specific to a certain mode. If you use an option that is not supported for the selected mode, the tool generates an error message. When you specify multiple settings by using the command multiple times, the latest setting overwrites the previous settings.

Table 2-3 lists the commonly used options for averaged and time-based modes. For a complete option list, see the man pages.

Table 2-3 Commonly Used Options for Averaged and Time-Based Power Modes

Option	Averaged Power Mode	Time-Based Power Mode
`-cells`	Yes	Yes
`-exclude_arc_scaling_libcells`	Yes	Yes

Table 2-3 Commonly Used Options for Averaged and Time-Based Power Modes (Continued)

Option	Averaged Power Mode	Time-Based Power Mode
`-include`	No	Yes
`-include_groups`	No	Yes
`-multi_rail_waveform`	No	Yes
`-sdpd_tracking`	Yes	Yes
`-sdpd_tracking_cells`	Yes	Yes
`-separate_dyn_and_leak_power_waveform`	No	Yes
`-static_leakage_only`	Yes	No
`-through_mode`	Yes	No
`-waveform_interval`	No	Yes
`-waveform_format`	No	Yes
`-waveform_output`	No	Yes

Performing Power Analysis

To perform power analysis, run the `update_power` command. Based on the settings that you specify along with the `set_power_analysis_options` command, PrimePower performs power analysis.

When you run the `update_power` command, power analysis is triggered consistently for various power analysis modes. After the analysis, the `update_power` command generates the power data.

In the averaged mode, the tool uses the default settings when no activity data is available. However, in the time-based mode, you must specify the activity data in the event-based activity file format before using the `update_power` command.

For more information, see Averaged Power Analysis and Time-Based Power Analysis.

Generating Power Reports

Use the `report_power` command to generate an averaged or time-based power report containing the power consumption for the design.

Example 2-1 shows the power report in the averaged power analysis mode:

Example 2-1 Power Report Generated During the Averaged Power Analysis Mode

```
****************************************
Report : Averaged Power
****************************************
 Attributes
 ----------
 i  -  Including register clock pin internal power
 u  -  User-defined power group
                    Internal  Switching Leakage   Total
Power Group         Power     Power     Power     Power     (%)    Attrs
--------------------------------------------------------------------------
io_pad              0.0000    0.0000    0.0000    0.0000    (0.00%)
memory              0.0000    0.0000    0.0000    0.0000    (0.00%)
black_box           0.0000    0.0000    0.0000    0.0000    (0.00%)
clock_network       8.041e-04 0.0000    0.0000    8.041e-04 (21.28%) i
register            4.705e-04 1.095e-04 6.660e-08 5.801e-04 (15.35%)
combinational       9.012e-04 1.219e-03 1.836e-07 2.121e-03 (56.14%)
sequential          5.301e-05 2.200e-04 9.163e-09 2.730e-04 (7.23%)

 Net Switching Power  = 1.549e-03   (41.00%)
 Cell Internal Power  = 2.229e-03   (59.00%)
 Cell Leakage Power   = 2.594e-07   ( 0.01%)

Total Power           = 3.778e-03  (100.00%)
```

Example 2-2 shows the power report with peak power for time-based power analysis. You can also use the nWave waveform viewer to view the power waveforms.

Example 2-2 Peak Power Report Generated During the Time-Based Power Analysis Mode

```
**************************************
Report : Time-Based Power
...
**************************************
 Attributes
 ----------
 i  -  Including register clock pin internal power
 u  -  User-defined power group

                    Internal   Switching  Leakage  Total
Power Group         Power      Power      Power    Power      (%)      Attrs
-------------------------------------------------------------------------------
io_pad              0.0000     0.0000     0.0000    0.0000    (0.00%)
memory              0.0000     0.0000     0.0000    0.0000    (0.00%)
black_box           0.0000     0.0000     0.0000    0.0000    (0.00%)
clock_network       7.133e-04  0.0000     0.0000    7.133e-04 (19.44%) i
register            4.701e-04  1.095e-04  6.660e-08 5.797e-04 (15.79%)
combinational       8.967e-04  1.207e-03  1.836e-07 2.104e-03 (57.33%)
sequential          5.297e-05  2.200e-04  9.163e-09 2.730e-04 ( 7.44%)

 Net Switching Power  = 1.537e-03   (41.87%)
 Cell Internal Power  = 2.133e-03   (58.12%)
 Cell Leakage Power   = 2.594e-07   ( 0.01%)
 -----------------------------------------------
Total Power           = 3.670e-03   (100.00%)
-----------------------------------------------
X Transition Power    = 2.171e-07
Glitching Power       = 3.018e-05

Peak Power            = 0.0968
Peak Time             = 1956.00
```

You can also generate your own reports using the Tcl constructs and accessing the power attributes. To save and restore your power parameters, set the `power_enable_analysis` variable to `true`.

For more information, see RedHawk Analysis Fusion in IC Compiler II and Viewing Power Waveforms From Time-Based Analysis.

Saving and Restoring PrimePower Sessions

Use the `save_session` and `restore_session` commands to save the current state of a PrimePower tool session and restore the same session. To examine the results of an earlier session of power analysis, use the save and restore feature to avoid repeating the execution of the time-consuming `update_power` command. The `restore_session` command begins from the same analysis point where you last saved your session, and uses only a small fraction of the original runtime.

The tool version used to save and restore a PrimePower tool session must be the same. You can locate the version used to save the session from the README file located in the session directory. You can also restore any PrimeTime or PrimeTime signal integrity timing analysis sessions and continue the power analysis.

For more information about saving and restoring a session, see the *PrimeTime User Guide* and the corresponding man pages.

3

Name Mapping

This chapter describes the name mapping mechanisms used in PrimePower to match the RTL activity to the gate-level netlist, when annotating the RTL simulation activity files. It contains the following topics:

Overview

During synthesis, the name manipulations might cause the RTL synthesis invariants (register outputs, primary inputs, tristates, and black boxes) to be renamed. To match the RTL activity to the gate-level components, PrimePower supports the following mapping mechanisms in a precedence order from high to low:

1. The name mapping `set_rtl_to_gate_name` command
2. Exact name mapping
3. Built-in name mapping

When annotating the RTL activity to the gate-level objects, PrimePower uses the `set_rtl_to_gate_name` command to match RTL to gate-level objects. During synthesis, using the Design Compiler tool, you can track the name changes and generate a map file that contains the `set_rtl_to_gate_name` command.

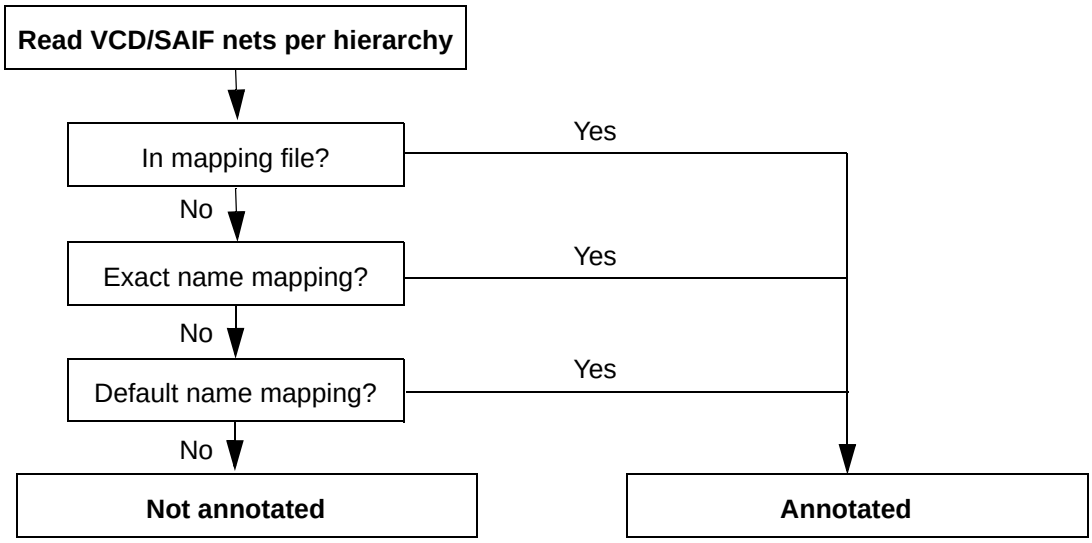
If the map file is unavailable, define the mapping using the `set_rtl_to_gate_name` command in a user-defined file. If the user-defined file is not available, the tool checks for the gate-level names if they exactly match the RTL simulation names. If not, the tool uses a built-in name mapping mechanism that matches the default name-manipulation supported by the Design Compiler tool, or you can use the `-substitute` option of the `set_rtl_to_gate_name` command to make global substitutions.

If the tool does not find a corresponding gate-level object, it does not annotate the object in the RTL activity file.

You must source the map file before the `read_vcd` or `read_saif` command for creating a name-mapping database for name mapping. If you modify the map file, be sure to re-source the map file and then rerun the `read_vcd` or `read_saif` command to reannotate the activity.

Figure 3-1 indicates the name mapping flow.

Figure 3-1 Name Mapping Flow



Name Mapping Using set_rtl_to_gate_name

PrimePower uses the `set_rtl_to_gate_name` command to perform the name mapping of RTL-level to gate-level objects. Using this command, you can specify the unique name mappings and global substitutions. When you specify this command, the tool creates a name mapping database. The tool evaluates the name mapping database when reading the SAIF or VCD simulation activity files. Specify the `set_rtl_to_gate_name` command before the `read_saif` or `read_vcd -rtl` command.

The following example shows the mapping between an RTL activity file object, a, and the gate-level netlist object, a_reg.

```
pwr_shell> set_rtl_to_gate_name -rtl {a} -gate {a_reg}
```

When mapping RTL registers to gate-level sequential cells, the tool applies the annotation to the Q pin, and the inverse to the QB pin.

Generating the Name Mapping File

When performing synthesis using the Design Compiler tool, use the `saif_map` command to generate a name mapping file. The command tracks the name changes throughout synthesis, and generates a mapping file for PrimePower. The generated map file helps maximize the RTL annotation in PrimePower, and thereby the accuracy and performance of power analysis.

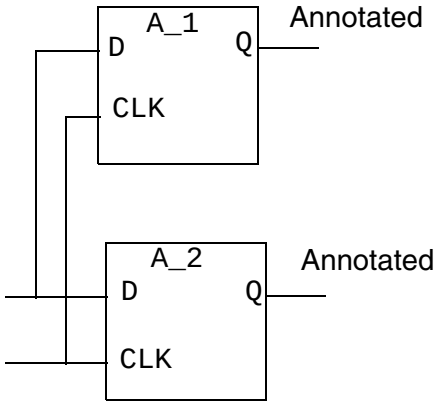
For more information about the name mapping file, see *Power Compiler User Guide*.

Mapping RTL to Multiple Gate-Level Objects

Use the `set_rtl_to_gate_name` command to map an RTL object to multiple gate-level objects. This helps maximize the switching activity annotation where the activity of the same RTL object is applied to multiple ports, nets, and cells for accurate power analysis.

Figure 3-2 shows how an RTL object is mapped to multiple gate-level objects.

Figure 3-2 Mapping RTL-to-Multiple Gate-Level Objects



All the map file entries for the RTL object are applied due to the one-to-many mapping support. Mapping RTL-to-multiple gate-level objects improves the switching activity annotation for the following cases:

- IC Compiler II clock tree synthesis buffer insertion

  When you annotate the buffers inserted by IC Compiler II clock tree synthesis, the multiple mapping capability enables the annotation of the RTL clock to all the inserted buffers.

- Design Compiler manual replication

  When you annotate the replicated instances generated by Design Compiler, the multiple mapping capability supports annotation of all the replicated instances.

- Clock tree synthesis port punching

  When you annotate the additional clock ports of the same RTL clock, the multiple mapping capability supports annotation of all the clock ports.

- Hierarchical flow

  When you annotate the block-level activity and map multiple instances of a hierarchical block, the multiple mapping capability annotates all the instantiated blocks without modifying the block-level map file.

If you repeatedly map one instance of an RTL-to-gate object or map multiple RTL objects to one gate-level object, PrimePower honors only the first mapping definition and generates warning messages for the subsequent instances of the mapping.

You can map an RTL object to some gate-level objects with switching activity inversion and to the other gate-level objects without switching activity inversion.

```
set_rtl_to_gate_name -rtl u0 -gate u0_1   # 1st mapping of RTL object u0
set_rtl_to_gate_name -rtl u0 -gate u0_1   # Duplicate entry for u0->u0_1
                                          # generates warning
set_rtl_to_gate_name -rtl u1 -gate u1_1
set_rtl_to_gate_name -rtl u2 -gate u2_1
set_rtl_to_gate_name -rtl u0 -gate u0_2 -inverted  # 2nd mapping of RTL
                                                   # object u0 with
                                                   # activity inversion
set_rtl_to_gate_name -rtl u1 -gate u1_2
set_rtl_to_gate_name -rtl u2 -gate u2_2
set_rtl_to_gate_name -rtl u2 -gate u0_2            # Conflicting mapping
                                                   # of gate object u0_2
                                                   # with RTL u0 and u2.

                                                   # Generate a warning.
                                                   # Mapping to RTL u0 is
                                                   # honored.
```

You can also specify annotation for block-level RTL activity to multiple instances of the block in a design as shown in Example 3-1.

Example 3-1 Mapping One RTL Object to Two Gate-Level Objects at the Block-Level

```
set_rtl_to_gate_name -rtl u0 -gate u0_1           # 1st mapping of RTL object u0
set_rtl_to_gate_name -rtl u0 -gate u0_1           # Duplicate entry for u0->u0_1
                                                  # generates warning
set_rtl_to_gate_name -rtl u1 -gate u1_1
set_rtl_to_gate_name -rtl u2 -gate u2_1
set_rtl_to_gate_name -rtl u0 -gate u0_2 -inverted  # 2nd mapping of RTL object u0 with
                                                  # activity inversion

set_rtl_to_gate_name -rtl u1 -gate u1_2
set_rtl_to_gate_name -rtl u2 -gate u2_2
set_rtl_to_gate_name -rtl u2 -gate u0_2           # Conflicting mapping of gate
                                                  # object u0_2 with RTL u0 and u2.
                                                  # Generate a warning. Mapping
                                                  # to RTL u0 is honored.
```

As shown in the Example 3-1,

- You cannot remap the RTL object u0 with the gate u0_1.
- You can also map the RTL object u0 with the switching activity inverted u0_2 gate.
- You cannot map the same gate-level object u0_2 with multiple RTL objects u2 and u0. In this case, only the first mapping is honored.

Exact Name Mapping

To annotate the RTL objects that are not specifically mapped using the `set_rtl_to_gate_name` command, PrimePower searches for gate-level objects with the same name in the gate-level netlist. For example, with the exact name mapping, the RTL net A is mapped to the gate-level net A.

Case Sensitivity

By default, the exact name matching is case insensitive; for example, the RTL net A is mapped to the gate-level net a. The case sensitivity is controlled by the `power_read_activity_ignore_case` variable. The default is `true`. When reading the activity files generated from VHDL simulation, set the variable to `false`, because VHDL is case-sensitive.

Disabling Exact Name Mapping of RTL Activity to Gate-Level Nets

To disable the exact name mapping of the RTL activity to gate-level nets, use the `power_disable_exact_name_match_to_net` variable.

Synthesis might introduce combinational nets with inverted logic while retaining the original name. If these nets are not listed in the name mapping file, by default PrimePower performs exact name mapping of the RTL activity to the gate-level netlist. This causes the non-inverted values to be annotated. To avoid inadvertent annotation of the RTL activity that is no longer functionally equivalent to the gate-level net with the same name, set the `power_disable_exact_name_match_to_net` variable to `true`. By default, the setting is `false`. The exact name mapping is applicable to the cells and pins, but is disabled for nets.

The tool uses the exact name mapping method to annotate the nets connected to the hierarchical pins, because they have the same name as the pin, and exist at the same hierarchical level. To disable the exact name mapping to the hierarchical pins, set the `power_disable_exact_name_match_to_hier_pin` variable to `true`.

Built-In Name Mapping

If no exact match is found, the tool performs default name mapping to match the RTL to gate-level names. The built-in mapping technique considers the netlist flattening and the default name modifications performed during synthesis.

PrimePower performs default name mapping, when

- RTL hierarchy is mapped to the flattened gate-level objects by replacing the hierarchical separator “/” with “_” and removing the “\” in front of the flattened net names. For example, the hierarchical RTL net a/b/c is mapped to the flattened gate-level net, a_b_c.
- RTL registers are mapped to the gate-level objects appended with _reg. By default, the Design Compiler tool appends _reg to the RTL register names which are synthesized into sequential cells. For example, the RTL register, a, is mapped to the gate-level cell a_reg. As with mapping performed with the `set_rtl_to_gate_name` command, the default name mapping technique applies the annotation to the Q pin of the sequential, and its inverse to the QB pin.
- Bus dimension Verilog separators [ ] are mapped to “_”. For example, the RTL register reg a[7] is mapped to the gate-level object a_reg_7_.

Reporting User-Defined Mapping Rules

To generate a report that shows the RTL-to-gate mapping specified by the `set_rtl_to_gate_name` command, use the `report_name_mapping` command. If the gate-level objects specified by the `set_rtl_to_gate_name -gate` command do not exist in the netlist, the mapping is not stored in the database and cannot be used.

In the following example, the `report_name_mapping` command reports the value of the inverted flag for each name mapping entry.

```
pwr_shell> report_name_mapping
****************************************************************************
Report : report_name_mapping
****************************************************************************
Attributes
----------
     v - Inverted
----------------------------------------------------------------------------
RTL Name     Gate-Level Name    Type     Attributes
----------------------------------------------------------------------------
u0           u0_1               cell               # 1st mapping for u0
u1           u1_1               cell
u2           u2_1               cell
u0           u0_2               cell      v        # 2nd mapping for
                                                   # u0 with inversion
u1           u1_2               cell
u2           u2_2               cell
----------------------------------------------------------------------------
```

Reporting Mapping Information Using report_activity_file_check

Run the `report_activity_file_check` command to report the name mapping mechanism used for annotation without actual annotation. This allows you to quickly find out if the hierarchy is correctly interpreted or if any missing signals are in the map file. For a detailed description about reporting mapping information, see Debugging RTL Activity Annotation.

Resetting the Name Mapping Database

To avoid any inadvertent annotation of the RTL to gate-level objects due to the exact name mapping, you can disable the exact name mapping of RTL objects to the nets and hierarchical pins. If any annotation is missing, modify the name mapping database created by the `set_rtl_to_gate_name` command. Be sure to clear the database using the `reset_rtl_to_gate_name` command before you reload a modified name mapping file and reread the RTL activity file.

4

Annotating Switching Activity

This chapter describes how to annotate switching activity for power calculation in the averaged and time-averaged power analysis modes.

In the averaged power analysis mode, the tool calculates averaged power based on the toggle rates. The sources of the annotated activity can be default toggle rates, user-defined switching activity, SAIF files, or VCD files generated from simulation.

In the time-based mode, PrimePower reads either VCD or FSDB activity files that are generated from the gate-level or RTL simulation.

To calculate the power consumption for the complete design, the tool uses zero-delay simulation to propagate the switching activity to non-annotated nets.

Propagating activity through clock networks is also supported. For more information, see Propagating Activities Through Clock Networks.

This chapter contains the following topics:

RTL- and Gate-Level Annotation

PrimePower annotates switching activity from both RTL and gate-level simulation.

Early in the design cycle, use RTL simulation to explore your design and identify

- Which RTL architecture consumes the least power?
- Which module consumes the most power?
- Where is power being consumed within a specified block?

Later in the design cycle, use gate-level simulation instead of RTL simulation to annotate specific nets of your design or all the elements of your design for better accuracy.

Table 4-1 summarizes the various methods of annotating switching activity:

Table 4-1 Comparing Methods of Capturing Switching Activity

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

Annotating Switching Activity

The accuracy of the power analysis is proportional to the accuracy of the switching activity. You must specify the switching activity generated from the RTL or gate-level simulation for performing power analysis. If activity files are not available, use the user-defined switching activity commands to provide realistic activity for accurate analysis results.

PrimePower supports the following methods for annotating switching activity in averaged and time-based power modes:

Annotation Methods	Commands	Related Topics
Annotation with activity files	`read_vcd` and `read_fsdb`	Annotating Switching Activity Using Event Files
Annotation with SAIF files	`read_saif or merge_saif`	Annotating Switching Activity Using SAIF Files
Annotation on design objects	`set_switching_activity`	Annotating Activity Using the set_switching_activity Command
Annotation of constant values on register set and reset pins	`infer_switching_activity -apply`	Inferred Switching Activity
Annotation on clock nets	`create_clock`	Annotating Clocks
Annotation of case constants	`set_case_analysis` or `set_switching_activity -force`	Annotating Case Constants

The tool supports annotations and bidirectional (that is, forward and backward) implications sequentially for each activity type (except case constants) in the precedence order from high to low. Activity with lower precedence cannot override the implied activity derived from an activity type with higher precedence. See Table 5-1 and Table 6-1 for the lists of the activity annotation types supported in averaged and time-based modes, respectively.

Annotating Switching Activity Using Event Files

A VCD or an FSDB file stores logic value changes in a design. Use the `read_vcd` and `read_fsdb` commands to specify an RTL or a gate-level activity file in either VCD or FSDB format, respectively.

For more information about types of simulation activity file format and how to generate the VCD, FSDB or SAIF activity files, see Simulation Activity Formats.

In the averaged power analysis mode, when you specify an event-based activity file, the tool by default extracts the toggle rates and static probabilities from the file and applies them to the corresponding design objects. Use the `-strip_path` option to specify the path to the current design instantiated in the simulator environment.

The following example reads a VCD file, my.vcd, which instantiates the current design as TB/DUT:

```
pwr_shell> read_vcd -strip_path TB/DUT my.vcd
```

The `read_vcd` command supports the following features during averaged power analysis:

- Supports name mapping when annotating VCD activity for both gate-level and RTL event-based activity files, either with or without the `-rtl` option.
- Supports Conditional Power Analysis with the `-when` option. When you specify this option, the tool extracts the switching activity from the VCD file. The annotated toggle rates reflect only the activity for those time windows.

When you use the `read_vcd` command, PrimePower checks the percentage of the annotated nets and extracts the toggle information for the nets available in the VCD file.

When state-dependent and path-dependent tracking is enabled, you can specify only one VCD file. When tracking is disabled, which is the default, you can apply activity from multiple VCD files. For more information, see Extracting State-Dependent and Path-Dependent Activity.

After a successful execution, the `read_vcd` command generates a summary that includes the number of nets in the design, the number of leaf cells in the design, and the number of leaf cells with switching activity annotation on all input pins. The following example shows a summary generated by the `read_vcd` command:

```
pwr_shell> read_vcd rtlvcd.dump -strip_path tb/mypath
checked out license 'PrimePower'

============================================================================
Summary:
Total number of nets = 1625
Number of annotated nets = 437 (26.89%)
Total number of leaf cells = 1339
Number of fully annotated leaf cells = 114 (8.51%)
============================================================================
```

As shown in the example, there are 1625 nets in the design. If a net appears at different levels of the design hierarchy, the net is counted only once.

The number of annotated nets is the number of nets for which switching activity is specified in the VCD file. The total number of leaf cells is the number of leaf cells in the design. The number of fully annotated leaf cells is the number of leaf cells in the design for which switching activity is annotated for all the leaf cell pins.

Virtual FSDB Files

By default, running the `read_fsdb` command invokes the native FSDB reader through Verdi. To virtually load multiple FSDB files through the Verdi platform, set the `power_enable_advanced_fsdb_reader` variable to `true` before the `read_fsdb` command.

Using the virtual FSDB file allows you to seamlessly debug across multiple FSDB files without keeping track of the file content. You can merge multiple virtual FSDB files into one full, gate-level FSDB file.

Script Example

```
set power_enable_advanced_fsdb_reader true
set search_path .
set link_library "* core_typ.db"
read_verilog test.vg
current_design test
link
read_fsdb -strip_path tb/MUT VirtualFile.vf
report_switching_activity
report_power
```

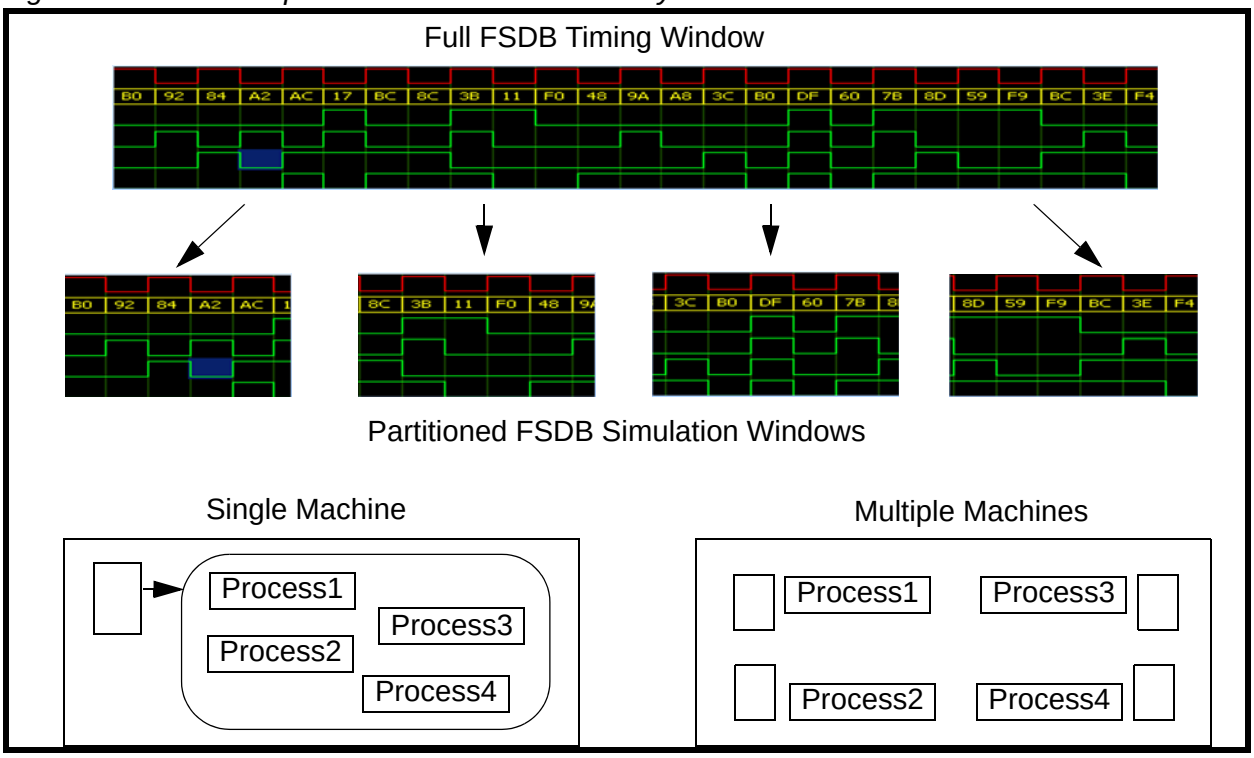
Solving Activity Conflicts

When running power analysis on a full-chip design, you might merge multiple block-level virtual FSDB files into one single FSDB file by using Verdi fsdbjoin or fsdbmerge utilities.

By default, PrimePower uses all the switching data in the merged FSDB file for power calculation. However, the merged FSDB file might contain conflict activity data between parent and child blocks when the signal on a net has different transitions on the same time stamp, thus resulting in inaccurate power numbers.

If top-level annotation is missing, then only internal and leakage power will be propagated. If child-level annotation is missing, then only switching power will be propagated.

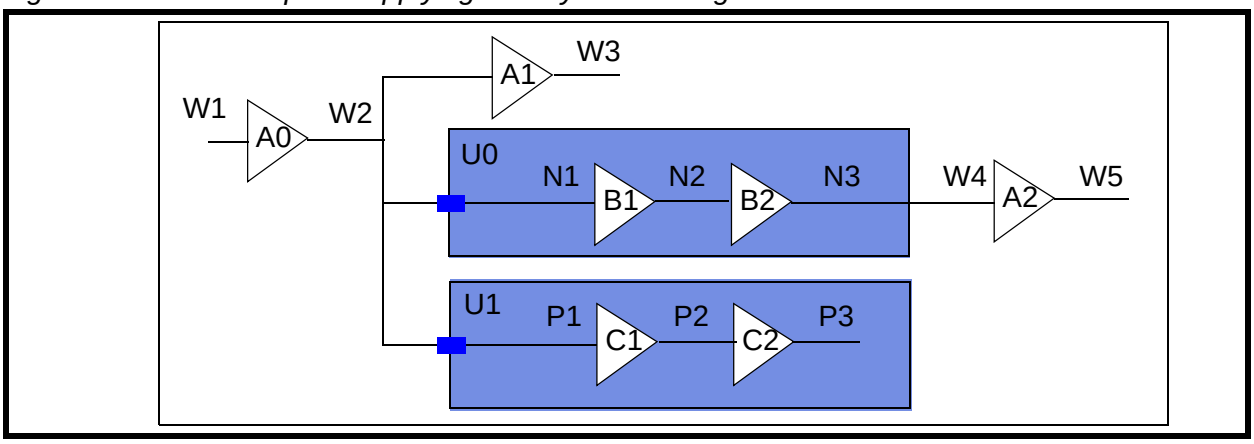
Figure 4-1 An Example of Concurrent Event Analysis in the Time-Based Mode



Example

Taking the design in Figure 4-2 as an example, net segments W2, N1 and P1 are on the same net, and each net segment has activity valid only within its block. The design has a merged FSDB file, which contains all activity data in the hierarchy. By default, PrimePower uses all the activity data from the same net.

Figure 4-2 An Example of Applying Activity in the Merged FSDB Flow



To resolve the conflict activity,

1. Set the `power_enable_merged_fsdb` variable to `true` to enable the merged FSDB flow.

```
pwr_shell> set power_enable_analysis true
pwr_shell> set power_analysis_mode time_based
pwr_shell> set power_enable_merged_fsdb true
```

2. Run the `set_power_analysis_options -conflict_activity_cells` command to specify block instances U0 and U1 as those which might have conflict activity.

```
pwr_shell> set_power_analysis_options -conflict_activity_cells { u0 }
```

3. Run the `read_fsdb` command to apply the activity data from the merged FSDB file by hierarchy. Block-level activity is applied only to the child-level blocks, for calculating block-level power values, such as the power from N1 to B1.

```
pwr_shell> read_fsdb -strip_path top/inst gate.fsdb
```

4. Run the `update_power` command to calculate power for the design. When finished, run the `report_power -cell_power` command to generate the power report.

Figure 4-2 shows the activity applied to each block instance for power calculation. As shown, the tool assumes each net segment has activity that are only valid within its block:

- Activity on W2 net segment is valid for cells A0 and A1.
- Activity on N1 net segment is valid for cell B1.
- Activity on P1 net segment is valid for cell C1.

Table 4-2 Net Activity Used During Power Calculation in the Merged FSDB Flow

Instance power	A0	A1	B1	C1	B2	C2
Switching power	W2	W3	N2	P2	N3	P3
Internal power	W1 W2	W2 W3	N1 N2	P1 P2	N2 N3	P2 P3
Leakage power	W1 W2	W2 W3	N1 N2	P1 P2	N2 N3	P2 P3

Annotating Switching Activity Using SAIF Files

PrimePower uses the following ways to read and write the SAIF files:

- Reading SAIF Files Using the read_saif Command

- Reading SAIF Files Using the merge_saif Command
- Writing SAIF Files Using the write_saif Command

Reading SAIF Files Using the read_saif Command

Use the `read_saif` command to apply the switching activity from both the RTL and gate-level SAIF files.

If any existing activity is overwritten by the newly annotated values, the tool issues a PSW-211 warning message and lists the nets whose annotated activity values are overwritten. To generate a report with detailed debugging information, use the `-report_inconsistent_annotation` option of the `read_saif` command.

Here is the syntax of the report:

```
Net1: <full_net_name>
saif_object_name_1 object_type toggle_rate static_prob
saif_object_name_2 object_type toggle_rate static_prob
…
Net2 : <full_net_name>
saif_object_name_1 object_type toggle_rate static_prob
saif_object_name_2 object_type toggle_rate static_prob
```

Use the `-strip_path` option to specify the path to the current design instantiated in the simulator environment. For example, the following example reads in a SAIF file, my.saif, which instantiates the current design as TB/DUT:

```
pwr_shell> read_saif -strip_path TB/DUT my.saif
```

Reading SAIF Files Using the merge_saif Command

Use the `merge_saif` command to read switching activity information from multiple SAIF files. When you specify a weight for each input SAIF file, the tool annotates a weighted sum of the switching activities. Use the command for flows where different SAIF files are generated for different modes of the same design. The switching activity from all the different modes can be used for power calculations. For example, if your design has the standby, slow, and fast modes, the standby.saif, slow.saif, and fast.saif SAIF files are generated for these modes, respectively.

Based on the expected usage of the design, specify the following weight to the SAIF files:

- standby.saif, 80%
- slow.saif, 5%
- fast.saif, 15%

Use the `merge_saif` command to read and weight the SAIF files. Use the `merge_saif -output` command to generate a SAIF file containing the weighted sum of the switching activities.

When you use the `merge_saif` command, the tool reads and propagates the switching activity for each SAIF file separately; and then applies the weights. The output SAIF file contains the resulting state-dependent and path-dependent gate-level activity.

Use the `-simple_merge` option to prevent the separate propagation of each SAIF file when merging gate-level SAIF files which contain the activity for all the nets in the design. When you specify this option, the tool weighs the activity in each SAIF file and merges before propagating activity and output in the SAIF file specified with the `-output` option.

You cannot use the SAIF files with state-dependent and path-dependent information with the `merge_saif` command.

Writing SAIF Files Using the write_saif Command

Use the `write_saif` command to write out SAIF files which contain user-annotated and propagated switching activities for the design.

Writing out switching activities for large designs might generate large SAIF files and take huge amounts of disk space. In this case, use the `-compress gzip` option to generate a compressed SAIF file with the .gz extension. The compressed gzip file is supported by the `read_saif` command.

```
pwr_shell> write_saif file1.saif -compress gzip
```

To uncompress the compressed file, use the gunzip utility.

Annotating Activity Using power_default_toggle_rate

When simulation activity files are not available, you can annotate the activity by setting a toggle rate to all the starting points by using the `power_default_toggle_rate` variable. For accurate power calculation results, you can set the toggle rate for different parameters based on your design types. The value that you specify must generate a pessimistic power value compared to the gate-level VCD results.

The following example annotates a default toggle rate value of 0.2 to all the starting points.

```
pwr_shell> set power_default_toggle_rate 0.2
```

Annotating Activity Using the set_switching_activity Command

If the toggle rate information from simulation is not available, use the `set_switching_activity` command to specify the switching activity on the design objects. Specify the toggle rate and the static probability to the specific objects or object types, and apply the switching activity to the objects within the specified clock domain.

When annotating clock-gating cells with a factor that is a multiple of the related clock's toggle rate, use the `-clock_domains` option along with the `-clock_derate` option to ensure that the clock gate generates toggles at the expected rate.

Note:

The clock derating factor set by the `set_activity_derate` command takes precedence over the one set by the `set_switching_activity -clock_derate` command.

Table 4-3 lists the commonly used options for annotating switching activity on nets, ports, and pins.

Table 4-3 Commonly Used Options for the set_switching_activity Command

Option	Description
`-toggle_rate`	Sets the toggle rate that represents the number of glitch-free transitions either from low to high, or high to low, during the time period specified with the `-period`, the `-base_clock` or the `-clock_domains` option. If neither the `-period` nor the `-base_clock` option is specified, the tool assumes the default library time unit is the same as the period.
`-static_probability`	Specifies the percentage of time when a signal stays at logic state 1.
`-period`	Specifies the time period for applying the toggle rate.
`-base_clock`	Indicates that the toggle rate is relative to the named clock, which can be any design clock. When you specify this option, the tool controls the toggle rate applied to the object and does not modify the value of the `power_base_clock` attribute.

Table 4-3 Commonly Used Options for the set_switching_activity Command (Continued)

Option	Description
`-clock_domains`	Specifies the period to which the toggle rate applies. When you specify this option, the toggle, which is relative to the period of the clock domain, is assigned to the objects within the listed clock domain. The clock domain is determined by the value of the pin or the net attribute `power_base_clock`, representing the related clock. The tool derives the value from the SDC commands, by tracing the clock network and the fanin-to and fanout-from nets where the `power_base_clock` attribute is known. By default, when an object has more than one related clock, the tool assigns the fastest clock to the `power_base_clock` attribute. To manually assign a clock to the `power_base_clock` attribute, use the `set_power_base_clock` command. To remove the manual setting, use the `reset_power_base_clock` command. To specify the default `power_base_clock` assigned to an object with no related clock, set the value of the `power_default_base_clock` variable to the clock.

Examples

In the following example, the tool applies a toggle rate of 0.1 to all the primary inputs of the design within the clk1 clock domain.

```
pwr_shell> set_switching_activity -type inputs -toggle_rate 0.1 \
           -clock_domains {clk1}
```

If the period of the clock clk1 is 25 nanoseconds, the tool applies the toggle rate 0.1 to all the inputs in the clk1 clock domain, which is 0.004 toggles per nanosecond. Internally, the tool converts all toggle rates to an absolute value relative to the default library time unit and stores the value as the `toggle_rate` attribute. Therefore, the toggle rate of the clock clk1 is 2 toggles/25 nanoseconds = 0.08 toggles/nanosecond.

In the following example, the tool assigns a toggle rate of 0.3 to the activity of net n1, relative to the period of the clk2 clock.

```
pwr_shell> set_switching_activity -toggle_rate 0.3 -base_clock clk2 n1
```

If the clk2 clock has a period of 10 nanoseconds, then the toggle rate of n1 is 0.3/10 nanoseconds = 0.03.

Annotating Clocks

PrimePower annotates the switching activity on the clock objects that are defined by the `create_clock` or `create_generated_clock` command.

The following example indicates the clock port `clk1` that toggles twice every 25 nanoseconds:

```
pwr_shell> create_clock -period 25 clk
```

You can also define clocks on the internal clock pins by using the `create_clock` command.

Inferred Switching Activity

When the VCD or SAIF switching activity is not provided, the tool estimates the switching activity at the primary input ports of the design and the output ports of the black boxes. It uses the default switching activity for the non-annotated pins and where the switching activity cannot be propagated.

For high-fanout nets, such as register resets, the default activity cannot be used. To avoid the default annotation on the register set and reset pins, use the `infer_switching_activity` command to apply activity to these pins so that they are disabled. The command also annotates constant values to design nets, and reports the recommended value for disabling the set and reset pins so that switching activity is propagated through the registers.

When you specify the `infer_switching_activity` command, the tool identifies the preset and clear pins on all the registers using the following pin attributes:

- `is_clear_pin true`
- `is_preset_pin false`

Note:
This mechanism might not identify all the preset and clear pins of the registers.

Use the `is_async_pin` attribute to determine if the preset and clear pins are asynchronous or synchronous. During identification, the tool traces the pins back to the ports or pins through the buffers and inverters to determine the active level of the signals. It verifies if the parity of the set and reset pins is compatible to ensure that the disabling value is the same for all the pins connected to the net.

When you specify the `infer_switching_activity` command without any options, PrimePower generates a report showing the current and inferred switching activity values specific to each pin, as shown in Example 4-1. Use the inferred values of the toggle rate and static probability to provide more realistic switching activity to derive more accurate power results.

Example 4-1 Report Generated by the infer_switching_activity Command With Default Settings

```
pwr_shell> infer_switching_activity
-------------------------------------------------------------------------
                      Current         Current     Proposed     Proposed
                      Static          Toggle      Static       Toggle
Objects   Type        Probability     Rate        Probability  Rate
-------------------------------------------------------------------------
rst       Driver      0.50            0.01        1.0          0.0
-------------------------------------------------------------------------
```

When you specify the `-apply` option, PrimePower

- Generates a report showing the current and inferred switching activity values specific to each pin (same as Example 4-1).
- Applies the inferred switching activity values using the `set_switching_activity` commands to the respective pins.

When you specify the `-verbose` option, the report shows the receiver pins and their types.

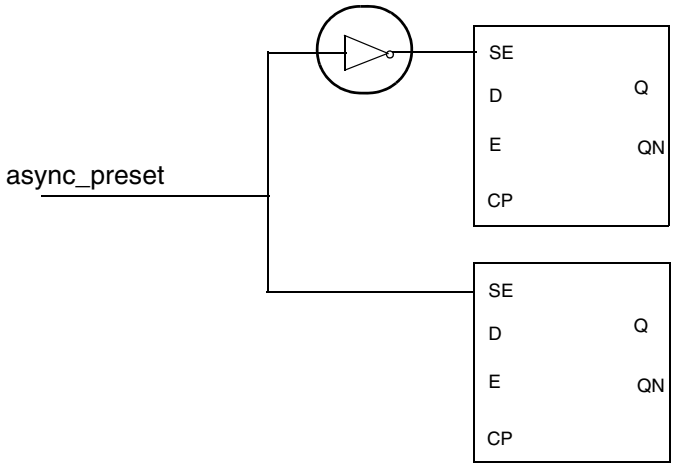
You can save the inferred switching activity values to an ASCII file using the `-output` option of the `infer_switching_activity` command. The ASCII file contains the `set_switching_activity` command issued for each pin, as shown in the following example:

```
pwr_shell> set_switching_activity -static_probability 1.0 \
           -toggle_rate 0.0 rst
```

PrimePower does not infer the switching activity in the following cases:

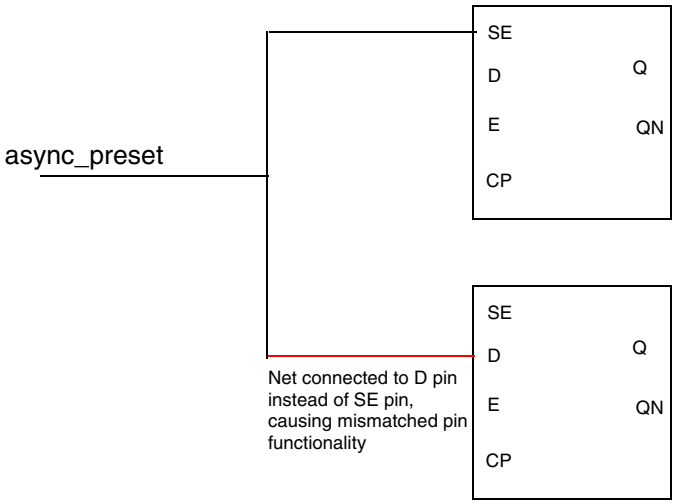
- When it finds a mismatch in the active values of the nets connected to clear and preset pins of multiple registers, as shown in Figure 4-3.

Figure 4-3 The async_preset Pin Has Mismatched Active Values on Two Paths



- When the signals connected to clear and preset pins of the registers are also connected to register pins with different functionality, as shown in Figure 4-4. You cannot use the `infer_switching_activity` command for the scan-enable pins.

Figure 4-4 The async_preset Pin Connected to Mismatched Pin Functions of Two Registers



Annotating Case Constants

By default, PrimePower annotates power or ground nets with a toggle rate of 0 and a static probability of 1 or 0. Pins connected to these power or ground nets are reported as unconnected. PrimePower retains their constant values.

To set a case value (either 0 or 1) on any pin or port in the design, use the `set_case_analysis` or `set_switching_activity -force` command.

When setting a constant logic value on a pin or a port by using the `set_case_analysis` command, the tool calls the PrimeTime timer and propagates this logic value forward through the network if this constant value is a controlling value for the traversed logic. Backward propagation through the design is not supported for case analysis.

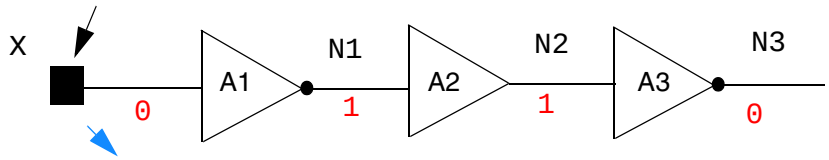
For example, when you set a constant value of 0 on one of the inputs of a NAND gate, PrimePower calls the PrimeTime timer and propagates this constant value to the NAND output, which is now considered at a logic constant of 1. This constant value is then propagated to all cells driven by this signal.

To set a constant value on a pin or a port without a timing update, use the `set_switching_activity -force` command. This constant overrides the constant value generated by the `set_case_analysis` command during activity propagation, but the results of the timing update remain unchanged.

Figure 4-5 shows how the constant value set by the `set_switching_activity -force` command replaces the value set by the `set_case_analysis` command.

Figure 4-5 Constants Set by set_switching_activity -force Override those Specified by set_case_analysis

```
set_case_analysis 1 (get_ports X]
```




```
set_switching_activity -force -static_probability 0 \
 -toggle_rate 0 (get_ports X]
```

Propagating case constants provides the following features and limitations:

- All combinational cells and sequential cells in the design are supported, including clock-gating cells and regular integrated clock gating (ICG) cells (excluding generic ICG cells).
  - All single-output combinational cells (such as AND, OR, XOR, etc.) are considered for constant value propagation. You need to annotate the data pins on combinational cells. If some of the pins are not annotated and the constant value is a controlling value for the traversed logic cell, then the input value is propagated through the cell. Else, the applied constant value will not be propagated to the output of the cell.
  - For propagation through a MUX cell, the select lines need to have a constant value set by the `set_switching_activity -force` or `set_case_analysis` command. Else, the tool assumes the select line is at the logic value X and stops propagation at the MUX cell.
  - For ICG cells, all the special enable pins, like SE or SI, need to be annotated; otherwise, a constant value 0 will be applied. The enable pins need be annotated; otherwise, the input value will be propagated through the pin.
- In the average mode, the tool first propagates forward the constant logic values and then updates the activity of all nodes in the design. The tool then performs implication and probabilistic ZDS propagation with the updated logic and activity values.
- In cycle-accurate peak power and time-based modes, you generate an event for each net which has a constant value and use this event to propagate the value to its fanout logic cone.
- The constant propagation with the `-force` option has the highest precedence over other activity annotation types. See Table 5-1 and Table 6-1 for the precedence order of the supported annotation types in the averaged and time-based modes.
- Only forward propagation is supported; backward propagation is not supported.

- The propagated logic values will not be reverted at the end of constant propagation.

Example

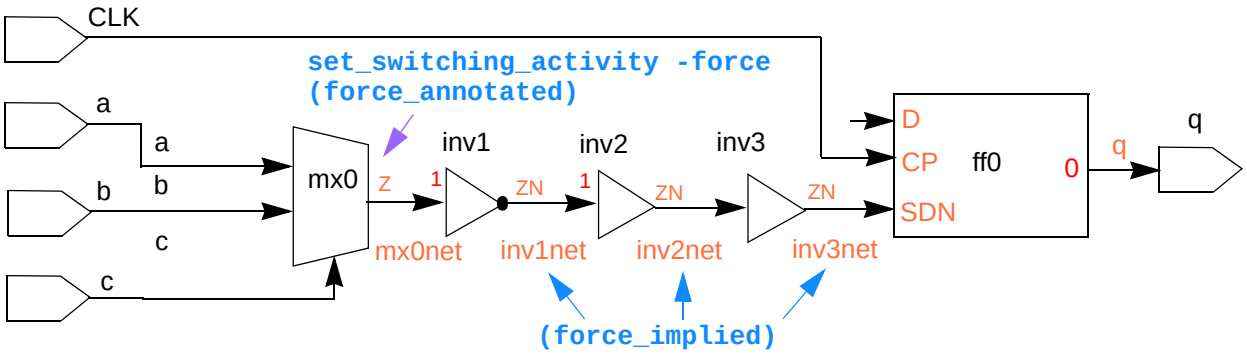
The following example applies a constant value of 0 to the signal reset:

```
pwr_shell> set_case_analysis 0 reset
```

The toggle rate of the reset signal is 0 and the static probability is 0.

In the following example, a constant value of 0 is applied to the output Z of the MUX net mx0. The assigned constant value is propagated to inv1net, inv2net, and inv3net.

```
pwr_shell> set_switching_activity –force –static_probability  0 \
        [get_pins mx0/Z]
pwr_shell> get_switching_activity [get_pins mx0/Z]
{"mx0/Z" 0 force_annotated 0 force_annotated 0 force_annotated}
pwr_shell> get_switching_activity [get_pins inv1/ZN]
{"inv1/ZN" 0 force_implied 0 force_implied 1 force_implied}
pwr_shell> get_switching_activity [get_nets inv2net]
{"inv2net" 0 force_implied 0 force_implied 0 force_implied}
```



Extracting State-Dependent and Path-Dependent Activity

By default, the activity is extracted from an event file as net toggle rates and static probabilities. However, you can use PrimePower to extract state-dependent, path-dependent toggle rates for large macro cells for more accurate power analysis results.

In the state-dependent and path-dependent analysis mode, the tool analyzes the averaged power using the actual state-dependent and path-dependent activity data of the specified cells. For the rest of the cells, the tool uses the estimated state-dependent and path-dependent activity data.

To use directed state-dependent and path-dependent analysis mode, run the `set_power_analysis_options –sdpd_tracking enabled` command before running the `read_vcd` command.

Running a state-dependent and path-dependent power analysis consists of the following steps:

1. Identify large macro cells.

   The tool identifies a list of large macro cells to analyze. By default, the list includes memory and black box power group objects. You can also specify cells using the `set_power_analysis_options –sdpd_tracking_cells` *`list_of_cells`* command.

   You must specify cells that have state-dependent and path-dependent power tables. The total power values of the cells can induce significant differences in the averaged and time-based power analyses. For example, cells that have a large difference in the state-dependent and path-dependent energy values should be included in the list.

   To view the cells that have state-dependent and path-dependent tracking enabled, use the `report_power_analysis_options` command.

   Here is an example:

```
pwr_shell> report_power_analysis_options
Option Name            Value
-----------------------------------
cells                  undefined
static_leakage_only    false
sdpd_tracking          enabled
sdpd_tracking_cells    mem1, mem2
```

2. Annotate large macro cells.

   PrimePower extracts the toggle rates and static probabilities for the nets in the design, and the state-dependent and path-dependent data for the ports of the large macro cells. They are annotated for averaged power analysis.

   You can extract the state-dependent and path-dependent data only when all the ports of the tracked cells are in the VCD file.

3. Analyze the averaged power of the design.

   After annotation, PrimePower estimates the state-dependent and path-dependent activity for the standard cells and computes the averaged power of the design.

Estimating Non-Annotated Switching Activity

During power analysis, PrimePower requires switching activity information on all design nets and state-dependent and path-dependent information on all design cells and pins. The tool, by default, estimates switching activity that is not user-annotated before calculating power.

The tool estimates the switching activity in the following stages:

- The design nets where the switching activity cannot be derived through propagation are assigned with the default activity.
- The switching activity is implied on the non-annotated pins of buffers and inverters or register outputs from the existing annotation. Based on the functionality of the cell, the tool derives the switching activity of non-annotated pins without simulating the annotated pins.
- The annotated activity is propagated using the zero-delay simulation to determine the activity on all other nets in the design.
- The simple switching activity on all the nets (either user-specified annotation, default, implied or propagated) is used to derive the state-dependent and path-dependent switching activity for the power arcs of all the cells.
- When tracking is enabled, the state-dependent and path-dependent activity for the tracked cells is extracted from the VCD file. For the remaining cells, simple switching activity is used to estimate the state-dependent and path-dependent activity.

Annotating Design Nets With Default Switching Activity Values

For design nets where PrimePower cannot accurately derive the switching activity or cannot estimate the activity using the propagation mechanism, the default switching activity is used. These include primary inputs, black box outputs, sequential cells, and tristates. If the tool does not find any user-defined annotation, it assigns the default switching activity to the nets.

The tool provides the following variables to assign default activity on the primary inputs, the black box outputs, and the sequential cells:

- `power_default_toggle_rate` (the default is `0.1`)
- `power_default_static_probability` (the default is `0.5`)
- `power_default_toggle_rate_reference_clock [fastest | related]` (the default is `related`)
- `power_sequential_default_static_probability` (the default is `0.5`)
- `power_sequential_default_toggle_rate` (the default is `-1`)

PrimePower uses a value of 0.1 for the default toggle rate and 0.5 for the default static probability. The toggle rate is relative to the associated clock.

Implied Activity

After annotating the switching activity, PrimePower derives the annotation through the inverters and buffers, and from the output pin of one register to another. This activity is called *implied activity* because it is derived without performing zero-delay simulation of random vectors.

For example, if the Q pin of a register is annotated with a toggle rate of 0.2 and a static probability of 0.8, the activity of the QB pin is implied with the same toggle rate, and a static probability of 0.2.

For multidriven nets, PrimePower implies annotation based on the activity applied to the driving and loading pins.

Similarly for inverters and buffers, the activity on either input or output pin can be implied from the existing annotation on the other pin.

For the following types of design nets, the rules for implied switching activity propagation are:

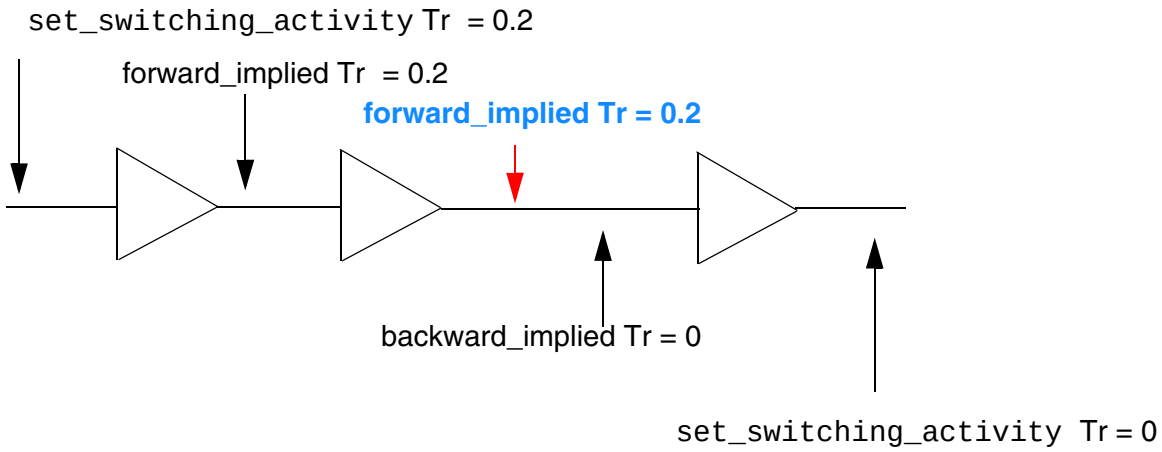
- Nets driving or driven by buffers: If the buffer input or output net is user- or default-annotated, then the non-annotated buffer output or input is default-annotated with the switching activity values on the annotated input or output.
- Nets driving or driven by inverters: If the inverter input or output net is user- or default-annotated, then the non-annotated 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, 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.

For the nets connected to buffers and inverters, the annotation is implied in both forward and backward directions. PrimePower assigns priority to the forward implied activity over the backward implied activity unless the forward implied activity is derived from the default annotation.

For inverter chains, the tool propagates the annotated activity to all the pins of the cells on the chain. When you specify multiple `set_switching_activity` commands, the forward and backward implied activity values might differ because the implied activity is derived in both the forward and backward directions. This causes a conflict, as shown in Figure 4-6.

Depending on the order in which you specified the `set_switching_activity` command, the resulting toggle rate (Tr) can be 0.2 or 0.0.

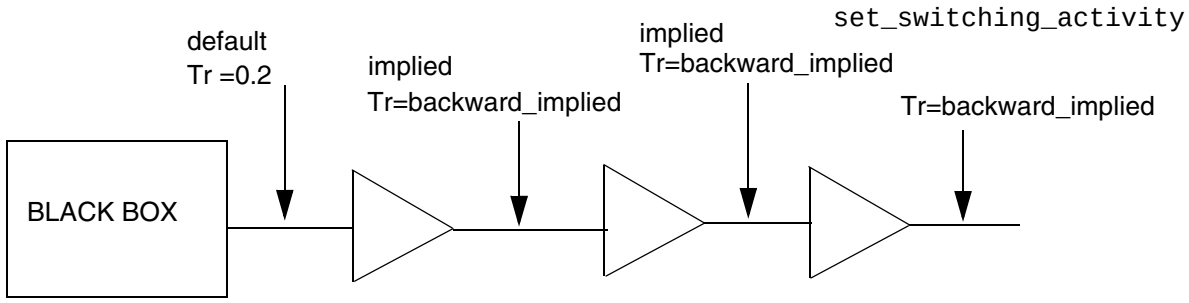
Figure 4-6 Conflict Resolution With the Precedence Rules Applied



To resolve this conflict, PrimePower assigns a higher precedence to the forward propagation over the backward propagation as shown in Figure 4-6. In this case, PrimePower uses the toggle rate of 0.2 because the forward propagation takes precedence over the backward propagation.

If the default activity is propagated forward, the backward implied activity derived from the user-defined annotation takes precedence, as shown in Figure 4-7. In this case, PrimePower chooses the `backward_implied` toggle rate because the user-defined backward propagation takes precedence over the default forward propagation. The annotated values are not overwritten.

Figure 4-7 Precedence Rules When User-Specified Activity is Applied



The `report_switching_activity` command reports both the forward and backward activity as implied in the report.

Deriving Net Switching Activity With Multiple Drivers

During power analysis, PrimePower derives the power characteristics of a net with multiple drivers using the following steps:

1. The toggle rate of the net is the sum of the toggle rates of the drivers driving the net.
2. The static probability of the net is the average static probability of the drivers driving the net.
3. The toggle rate and static probability value of 0 are assigned to the drivers that do not have annotated switching activity.
4. Drivers without annotated switching activity are excluded from the toggle rate and switching activity calculations.
5. Annotating switching activity on a net is equivalent to annotating switching activity on all the loads of the net. Similarly, annotating switching activity on one of the loads of a net is equivalent to annotating switching activity on the net and on all the other loads of the net.
6. When you set the switching activity multiple times on a net and its loads, the last setting overrides the previous ones.
7. When you annotate switching activity on a driver, the switching activity on the net or the load is ignored and the tool issues a warning.
8. When you do not annotate switching activity on any driver and the switching activity is annotated on the load (or net), the toggle rate of the load (or net) is distributed evenly to the drivers. Static probability of the load (or net) is copied to all the drivers.

The tool generates a warning message if it encounters a conflict when deriving the switching activity.

If the derived switching activity value does not match the annotated value, the tool marks the source of the switching activity as implied.

For more information about implied switching activity, see Implied Activity.

Propagating Switching Activity

During switching annotation, PrimePower uses the zero-delay simulation, a propagating mechanism, to derive the switching activity of design nets that are not user-annotated or default annotated. To determine the toggle rates and static probabilities of the non-annotated nets, the tool propagates random vectors by matching the annotated toggle rates and static probabilities with zero-delay simulation. The propagated values do not overwrite the user-annotated switching activity.

Timing Exceptions

When generating and propagating switching activities, PrimePower uses the following exceptions, as shown in Table 4-4.

Table 4-4 Exceptions Considered by PrimePower to Generate Vectors

Timing command	Attribute
`case_analysis`	`Yes`
`multicycle_path`	`No`
`disable_timing`	`No`
`false_path`	`No`

Correlation

While propagating switching activity through the design, PrimePower makes certain statistical assumptions. However, the logic states of the gate inputs can have interdependencies that affect the accuracy of any statistical model. This interdependency of inputs is called correlation.

Correlation affects the accuracy with which the tool propagates the toggle rates. Because accurate analysis depends on accurate toggle rates, correlation also affects the accuracy of power analysis.

PrimePower uses correlation within combinational and sequential logic, resulting in more accurate analysis of switching activity for several designs. The designs with re-convergent fanouts, multipliers, and parity trees exhibit high internal correlation. However, the tool cannot access correlation information that is external to the design. If any correlation exists between the primary inputs of the design, the tool cannot recognize the correlation.

Deriving State-Dependent and Path-Dependent Switching Activity

PrimePower must estimate the state-dependent and path-dependent toggle rates and static probabilities based on the activity on the pins if the state-dependent and path-dependent activity is not annotated. After obtaining the simple switching activity, the tool estimates the state-dependent static probability information and the state-dependent and path-dependent toggle rate information for every cell pin. The tool extracts this information from the switching activity of each input and output pin for the cell. Use gate-level SAIF files with state-dependent and path-dependent information for the most accurate power results, although the estimation mechanism of the state-dependent and path-dependent activity produces nearly accurate power results.

Based on the activity on the outputs and inputs of every leaf cell, PrimePower calculates the Boolean difference for each power arc to determine the contribution of each power arc. The tool extracts power arcs information from the library power table.

For black box cells, because the tool is not aware of the cell functionality, the state-dependent and path-dependent estimation is not as accurate as it is for standard cells. Exact state-dependent and path-dependent activity of black boxes, required for accurate power calculation, can be obtained from the VCD file by specifying the `-sdpd_tracking` option. Accurate black box power can also be obtained by annotating the state-dependent and path-dependent activity or power values from time-based analysis.

To annotate accurate state-dependent and path-dependent switching activity, read the SAIF file generated by the `write_saif` command when performing time-based analysis with state-dependent and path-dependent tracking.

See Also

- Power Analysis With User-Defined Internal, Leakage and Switching Power
- Extracting State-Dependent and Path-Dependent Activity

Reporting Switching Activity

PrimePower supports several commands to debug the annotation of switching activity and propagation. These commands return average activity data that can help you identify the problematic areas in your design. In addition, you can use this information to improve the quality of your testbenches through analysis of the actual data against the activity you expect to see.

To report or query the annotated information used for power analysis, use the following commands:

- `report_switching_activity`, as described in Reporting Switching Annotation Using report_switching_activity
- `get_switching_activity`, as described in Reporting Switching Annotation Using get_switching_activity

The commands use the switching activity generated from the `set_switching_activity` command and the SAIF or VCD files that you specify. After you specify the `read_saif`, `read_vcd`, and `set_switching_activity` commands, you can query or report the annotation on various design objects. Use the `get_switching_activity` and `report_switching_activity` commands to get the value of the annotation source indicator.

Reporting Switching Annotation Using report_switching_activity

The `report_switching_activity` command reports information about various sources of switching activity data. You can report switching activity data after you run the `read_saif` or `read_vcd` command.

Use the `-list_not_annotated` option to display the design objects that do not have user-annotated switching activity. For more information, see Switching Activity Annotation Sources Using the activity_source Attribute.

As shown in Example 4-2, the default report shows the percentage of design objects annotated from the activity file, the `set_switching_activity` command, the `set_case_analysis` command, and the `create_clock` command. In addition, the report shows the categorization of the nets driven by certain types of drivers.

Example 4-2 An Activity Report Generated by the report_switching_activity Command

```
pwr_shell> report_switching_activity

report_switching_activity
****************************************
Report : Switching Activity
Design : test1
Version: <version>
Date   : <time>
****************************************

Object         File(%)     SSA    SCA Clock Default Propagated Implied Annotated Total
Type
--------------------------------------------------------------------------------------
Nets       3489(100.0%) 0(0.0%) 0(0.0%) 0(0.0%) 0(0.0%) 0(0.0%) 0(0.0%) 0(0.0%) 3489
--------------------------------------------------------------------------------------
Nets Driven by
--------------------------------------------------------------------------------------
```

```
Primary        150(100.0%) 0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)    150
Input
Tri-State      300(100.0%) 0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)    300
Black Box       80(100.0%) 0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)     80
Sequential     956(100.0%) 0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)    956
Combinational 2012(100.0%) 0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)   2012
Memory           0(0%)     0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)      0
Clock Gate       0(0%)     0(0%)  0(0%)  0(0%)  0(0%)  0(0%)   0(0%)     0(0%)      0
----------------------------------------------------------------------------------------
```

Sometimes the VCD or SAIF file is not annotated completely due to incorrect name mapping. If this is the case, use the `report_switching_activity` command to identify the non-annotated objects in the design. For more information about debugging issues with respect to name mapping, see Reporting Mapping Information Using report_activity_file_check.

Use the following attributes to get the value of toggle rate and static probability:

```
glitch_rate
toggle_count
toggle_rate
static_probability
```

For example, if you suspect that the tool has incorrectly propagated low-activity rates through a series of sequential elements, resulting in zero activity at the output of some sequential elements, use the following commands to verify this issue:

```
#For propagation debugging, you must first
#specify the update_power command
update_power
#To provide information about nets driven by sequential
#cells that have zero toggle
report_switching_activity -coverage \
      -include_only sequential
report_switching_activity -list_low_activity \
      -include_only sequential
```

To determine average switching activity at the output of sequential elements with nonzero switching activities, use the `update_power` command. As shown in Example 4-3, the average activity report includes both toggle rates and toggle counts. The toggle rates are calculated with respect to the primary clock, which you can specify by using the `-base_clock` option.

Example 4-3 The Average Activity Report Showing Toggle Rates and Toggle Counts

```
pwr_shell> report_switching_activity -average_activity -hierarchy

****************************************
Report : Switching Activity
-average_activity -hierarchy
Design : counter
...
****************************************
```

```
Switching Activity Statistics for "counter"
-------------------------------------------------------------------------
Switching Activities per clock is with respect to clock ‘CLK’
Simulation time 10ns (1000 clock periods of clock ‘CLK’)
-------------------------------------------------------------------------
                  Toggle Count  Toggle Rate   Glitch Count    Glitch Rate
Object            Per Net       Per Clock     Per Net         Per Clock
-------------------------------------------------------------------------
counter              200           0.2            1              0.001
counter_low_bits     150           0.15           1              0.001
counter_high_bits    50            0.05           0              0
reset_block          1             0.001          0              0
overflow_detect      1             0.001          0              0
```

Listing Name Mapping Types When Reporting Switching Activity

By default, PrimePower lists the total number of netlist objects mapped from the activity files to the netlist. To report the numbers of netlist objects mapped by different mapping techniques in the switching activity checking report, use the `-include_mapping_types` option with the `report_switching_activity` command.

When the `-include_mapping_types` option is enabled, the activity report lists the number of the design objects by the following mapping techniques: name mapping file, exact name mapping, and built-in name mapping.

For more information about name mapping techniques used in the tool, see Name Mapping.

Figure 4-8 Generated Reports With and Without the -include_mapping_types Option

A switching activity report generated without the `-include_mapping_types` option

```
pwr_shell> report_switching_activity

Switching Activity Overview Statistics for "design"
-------------------------------------------------------------------------
                From Activity      From          From          From
Object Type     File (%)           SSA (%)       SCA (%)       Clock
------------------------------ ------------------------------ ----------
                13549(44.99%)      0(0.00%)      0(0.00%)      0(0.0
-------------------------------------------------------------------------
Nets Driven by
-------------------------------------------------------------------------
Primary Input      807(97.82%)     0(0.00%)      0(0.00%)      0(0.0
Tri-State            0(0%)         0(0%)         0(0%)         0(0%)
Black Box            0(0%)         0(0%)         0(0%)         0(0%)
Sequential        7237(92.03%)     0(0.00%)      0(0.00%)      0(0.0
Combinational     5268(24.87%)     0(0.00%)      0(0.00%)      0(0.0
Memory               0(0%)         0(0%)         0(0%)         0(0%)
Clock Gate         237(98.34%)     0(0.00%)      0(0.00%)      0(0.0
-------------------------------------------------------------------------
```

A switching activity report generated with the `-include_mapping_types` option

```
pwr_shell> report_switching_activity -include_mapping_types

Switching Activity Overview Statistics for "design"
-------------------------------------------------------------------------
                From Activity  From Activity  From Activity
                File Exact     File Map       File Built-in  From     Fr
Object Type     Mapping (%)    File (%)       Mapping (%)    SSA (%)  SC
------------------------------------------------------------------------
4020(13.35%)   9168(30.44%)    361(1.20%)   0(0.00%)   0(0.00%)  0(0.
-------------------------------------------------------------------------
Nets Driven by
-------------------------------------------------------------------------
Primary Input    0(0.00%)   807(97.82%)    0(0.00%)     0(0.00%)   0(
Tri-State        0(0%)        0(0%)        0(0%)        0(0%)      0(
Black Box        0(0%)        0(0%)        0(0%)        0(0%)      0(
Sequential       0(21.74%)  198(66.10%)   29(4.18%)     0(0.00%)   0(
Combinational  147(10.13%)  089(14.58%)   32(0.15%)     0(0.00%)   0(
Memory           0(0%)        0(0%)        0(0%)        0(0%)      0(
Clock Gate     163(67.63%)   74(30.71%)    0(0.00%)     0(0.00%)   0(
-------------------------------------------------------------------------
```

Reporting Switching Annotation Using get_switching_activity

To analyze and debug the annotation, use the `get_switching_activity -toggle_rate` command. This command returns the toggle rate for the specified design object as shown in the following example:

```
pwr_shell> get_switching_activity \
            -toggle_rate smas_read_data[29] \
           {"smas_read_data[29]" 0.000017 file}
```

Use the `get_switching_activity` command to report the switching activity attribute values and their sources in the following order:

- Toggle rate and activity source
- Glitch rate and activity source
- Static probability and activity source

In the following example, the `get_switching_activity` command reports the toggle rate as 0.00017, the glitch toggle rate as 0 and the static probability as 0.4998, with the annotation source as file.

```
pwr_shell> get_switching_activity \
           -toggle_rate smas_read_data[29] \
           {"smas_read_data[29]" 0.000017 file 0 file 0.4998 file}
```

Reporting Relative Toggle Rates

In PrimePower, toggle rates are based on one time unit. To report the related toggle rate information based on the period of the `power_base_clock`, run the `get_switching_activity -relative_toggle_rate` command.

For example, for a clk with a period of 12 nanoseconds, PrimePower reports its `toggle_rate` value as 0.1666666716337204 with respect to the time unit. When the `power_base_clock` period is set as 2 nanoseconds, PrimePower reports its `relative_toggle_rate` value as 0.3333333432674408.

Identifying Activity Sources

Use the `activity_source` attribute to identify the source of switching activity information for a net. The attribute value can be one of the following: `file`, `set_switching_activity`, `set_case_analysis`, `propagated`, `implied`, `default`, or `UNINITIALIZED`.

Table 4-5 shows the `activity_source` attribute value and switching activity source information in a descending order of priority:

Table 4-5 Switching Activity Annotation Sources Using the activity_source Attribute

Attribute value	Annotation source
`file` (Equivalent priority with `set_switching_activity`)	Toggle rate derived from a SAIF or VCD file using the `read_vcd` and `read_saif` commands.
`set_switching_activity` (Equivalent priority with `file`)	Toggle rate derived using the `set_switching_activity` command.
`set_case_analysis`	Toggle rate derived using the `set_case_analysis` command.
`create_clock`	Toggle rate derived using the `create_clock` command.
`implied`	Implied activity without random vector propagation from the annotated design points, Q/QB, buffers/inverters, multidriven resolution, and netlist constraints.
`default`	Toggle rate value is the tool default.
`propagated`	Toggle rate is propagated using random vectors from the annotated design points (zero-delay simulation).
`UNINITIALIZED`	Not user-annotated, implied, or default. The uninitialized nets or pins annotation is derived from cycle simulation, and the `activity_source` attribute value is updated to `propagated` after power analysis.

Debugging RTL Activity Annotation

To identify the causes for the unannotated nets, use the `report_activity_file_check` command. The `report_activity_file_check` command processes the activity file just like the `read_vcd` and `read_fsdb` commands without the actual annotation, and reports the following information:

- The VCD or SAIF signals and the design objects to which they are annotated

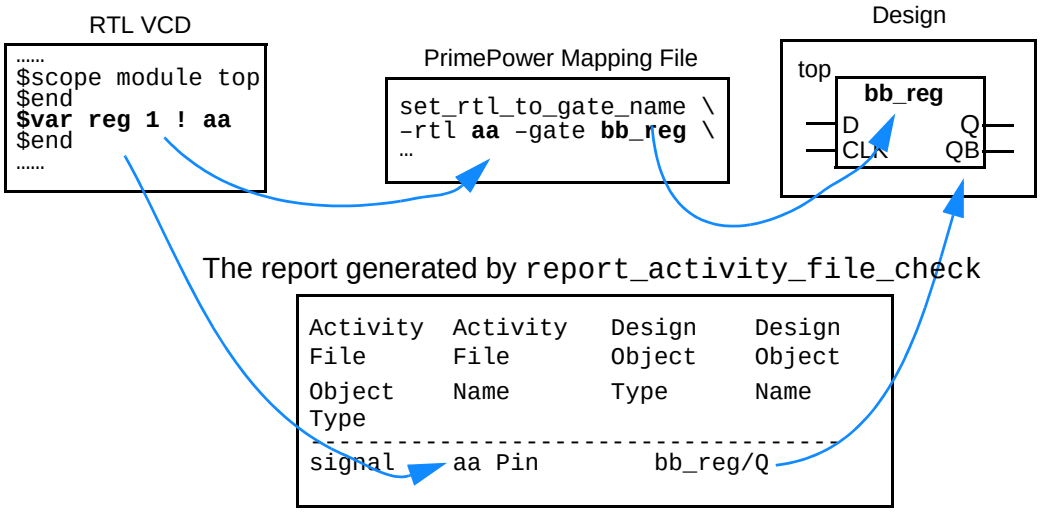
  For each hierarchical module (in the VCD file) or instance (in the SAIF file), the tool lists the corresponding hierarchical cell in the design, if found. For each signal (in the VCD file) or net (in the SAIF file), the tool lists its corresponding net or pin in the design, if any. The exact method used to arrive at the mapping is also reported, as well as the

information about any logical inversion between the signal in the VCD file (or the net in the SAIF file) and the net or pin in the design.

- The name mapping mechanism that is user-specified (with the `set_rtl_to_gate_name` command), exact name mapping, default name mapping, or a combination of these methods.
- Whether the logic inversion is specified using the `set_rtl_to_gate_name -inverted` command, or detected by default in the tool when the inversion attribute indicates the logic inversion between the RTL object and the gate-level object.
- The exact gate-level object to which the annotation is applied.

Figure 4-9 illustrates how you can debug missing annotation by checking the mapping information reported by the `report_activity_file_check` command.

Figure 4-9 Debugging Missing Annotation



Use the reported mapping information to debug the possible causes for missing annotation. The possible causes can be

- Name mapping not enabled for activity annotation

  You must specify the `-rtl` option to enable name mapping when annotating activity with the `read_vcd` command in the time-based mode. You do not need to specify the `-rtl` option in the averaged power mode because name mapping is always performed in the averaged mode.

  For more information, see Name Mapping.

- Unmatched VCD instances to design instances in the mapping file

  When annotation is complete by using the `read_vcd` or `read_saif` command, check the percentage of annotation reported in the summary report.

If the number of annotated nets is reported low in the summary report generated by the `read_vcd` or `read_saif` command, use the `–strip_path` or `-path` option with the `report_activity_file_check` command to check if the hierarchical VCD or SAIF paths to the design cell instances are correct. If not, fix the incorrect path in the mapping file.

The percentage of the RTL annotation should be equal to the percentage of synthesis invariants. The gate-level annotation should be 100%.

- Missing VCD signals during RTL annotation

  When annotation is complete, run the `report_switching_activity` command with the `–include_only`, `-rtl` and `–list_not_annotated` options to report the RTL invariant annotation. The generated activity report indicates the annotation percentage for RTL invariants, including primary inputs, tristate outputs, black boxes, and registers.

  To locate the signals that do exist in the VCD file but are missing or incorrectly specified in the name mapping file, use the `report_activity_file_check -find` command to view signals within a particular VCD hierarchy or with a particular prefix.

  For example, if the report lists the following unannotated nets;

```
a_r[6]
a_r[5]
a_r[13]
a_r[3]
a_r[0]
a_r[4]
a_r[7]
a_r[1]
a_r[2]
```

  You can run the `report_activity_file_check –find a_r*` command to check the mapping for the nets with the prefix a_r*.

```
Activity     Activity     Design     Design
File         File         Object     Object
Object Type  Name         Type       Name
-------------------------------------------------------------------
signal       a_rp[7]      None Found
signal       a_rp[4]      None Found
signal       a_rp[3]      None Found
signal       a_rp[2]      None Found
signal       a_rp[1]      None Found
signal       a_rp[0]      None Found
```

  In this case, you can modify the mapping for annotation correction by using the `set_rtl_to_gate_name` command to resolve the problem. For example:

```
pwr_shell> set_rtl_to_gate_name –rtl a_rp[7] –gate a_r[7]
```

  The following report example indicates the RTL object mapping to a cell, a net, or a pin.

```
   pwr_shell> report_activity_file_check -strip_path top/M1 -path M1 \
                 myvcd.vcd -find A221_m
****************************************************************************
Report : report_activity_file_check
…
Activity File :
myvcd.vcd ...
****************************************************************************
Mapping Method
--------------
     d - Default name mapping applied
     e - Exact name matching applied
     m - User-defined RTL-to-gate name mapping applied
     s - User-defined mapping by string substitution
-----------
Attributes
----------
v - inversion
-----------------
Activity File    Activity Design    Design    Name         Attributes
File             File     Object    Object    Mapping
Object Type      Name     Type      Name      Method
----------------------------------------------------------------------------
module           u0       Hier Cell u0_1      m
module           u0       Hier Cell u0_2      m
signal           u0/x     Net       u0_1/x    e,m
signal           u0/x     Net       u0_2/x    e,m
…
```

Removing Switching Activity Annotation

Use the `reset_switching_activity` command to remove switching activity annotation from individual design objects, as shown in the following example:

```
pwr_shell> reset_switching_activity objects
```

This command removes the simple, state-dependent, and path-dependent switching activity annotation from the specified objects.

When you specify the `reset_switching_activity` command without any option, it removes all previously annotated and propagated switching activity for the complete design hierarchy. By default, this command reports all warning messages.

Use the `reset_switching_activity` command to remove the switching activity information that you already annotated, before specifying new SAIF files.

5

Averaged Power Analysis

This chapter describes how PrimePower uses the annotated activity to calculate power in the averaged power analysis mode, and how the tool uses the annotated power for power consumption calculation.

In the averaged power analysis mode, the tool calculates average power based on the toggle rates. The sources of annotated activity can be the default toggle rates, user-defined switching activity, SAIF files, or VCD files generated from simulation. For more detailed descriptions about switching activity annotation, see Annotating Switching Activity.

To calculate the power consumption for the complete design, the tool uses zero-delay simulation to propagate the switching activity to non-annotated nets.

This chapter contains the following topics:

Annotating Activity in the Averaged Mode

In the averaged power mode, the switching activity consists of toggle rates and static probabilities (see Definition of Switching Activity in the Averaged Mode). The accuracy of the power analysis results depends on the accuracy of the switching activity values. Therefore, the switching activity obtained from the RTL or gate-level simulation must be annotated correctly.

The tool supports the annotation of simulation-based switching activity file formats: VCD, FSDB, VPD, and SAIF. If activity files are not available, use the user-defined switching activity commands to provide realistic activity for accurate analysis results.

To calculate power in the averaged analysis mode, the tool uses the state-dependent and path-dependent leaf-cell pin toggle rates and static probabilities. If not available, the tool first checks for switching activity on all the nets in the design; it assigns the defaults to all the black box outputs and primary inputs that do not have switching activity. For other non-annotated nets, the tool uses implied switching activity or derives the switching activity from zero-delay simulation of random vectors that match the toggle rates and static probabilities of the annotated nets.

From the net switching activity, the tool estimates the state-dependent and path-dependent leaf-cell pin toggle rates and static probabilities to calculate power.

Table 5-1 lists the activity annotation types supported in the averaged power mode in precedence order from high to low:

Table 5-1 Supported Activity Annotation Types in the Averaged Mode

Precedence Order	Activity Annotation Types
1	`set_switching_activity -force`
2	Activity files/`set_switching_activity`
3	Implied from `set_switching_activity -force`, activity files, and `set_switching_activity`
4	`set_case_analysis`
5	Logic constants
6	Implied from `set_case_analysis` and logic constants
7	`create_clock`
8	Implied from `create_clock`

Table 5-1 Supported Activity Annotation Types in the Averaged Mode (Continued)

Precedence Order	Activity Annotation Types
9	Propagated activity data
10	The tool's default

Note:

To give priority to annotations from the `set_case_analysis` command over other user annotations, set the `power_activity_file_precedence_over_sca` or `power_activity_file_ssa_precedence_over_ssa_force_implied` variable to `true`. The default is `false`.

See Also

- Annotating Switching Activity

Definition of Switching Activity in the Averaged Mode

Switching activity in the averaged mode is defined by the following parameters:

- Static Probability

  Static probability (SP) is the probability that a signal is at a specific logic state; it is expressed as a number between 0 and 1. SP1 is the static probability that a signal is at logic 1. Similarly, SP0 is the static probability that the signal is at logic 0.

  The static probability is the ratio of the time period for which the signal is at a certain logic state relative to the total simulation time. For example, if SP1 = 0.70, the signal is at logic 1 state 70 percent of the time. The tool uses SP1 when modeling switching activity.

- Toggle Rate

  The toggle rate is the number of 0-to-1 and 1-to-0 logic transitions of a design object per unit of time, such as a net, a pin, or a port. The toggle rate is denoted by TR.

You can annotate the following types of switching activity 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 design object has a logic value of 1. It defines the percentage of time the signal is at a high state. By default, the value is 0.5, meaning the signal is high for one half the time. This default value might be appropriate for data bus lines but for signals that seldom change, such as `reset` or `test_enable`.

  The toggle rate is the rate at which the design object switches between logic values 0 and 1 within a time period.

- State-dependent 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 path-dependent toggle rates on output pins of leaf cells

  The internal power characterization of output pins can be state-dependent, path-dependent, or both. Output pins of cells with such characterization can be annotated with state-dependent and path-dependent toggle rates.

- State-dependent static probability on leaf cells

  Cell leakage power can be characterized with state-dependent leakage power tables. Such cells can be annotated with state-dependent static probability.

Setting the Power Derating Factor

Use the `set_power_derate` command to specify the power derating factor for the rails on different objects in the current design. Derating means to adjusting the calculated power by a specified factor. You can also specify derating factors on the following power components:

- Internal power
- Switching power
- Leakage power

PrimePower applies the power derating factors on the following objects with a precedence order from high to low:

- Design rail

  When not specified, the tool applies scaling factors to all rails in the design. To set derating factors per power rail, use the `-rails` option.

- Leaf cell and power group

  The power derating feature treats a collection of cells as a power group. To set derating factors per PG pin of a leaf or library cell, use the `-pg_pin` option.

- Hierarchical cell
- Library cell
- Design and design cell

Note:

You can set the power derating factor on the same design object and for the same power component multiple times. The last value that you set overrides the previous settings.

Power Derating Per PG Pin

For complex macro cells, set different derating factors for each PG pin of a leaf or library cell by running the `set_power_derate` command with the `-pg_pin` option. You cannot set derating factors for the PG pins in hierarchical cells or the deign. The specified derating factors can be applied on different power components, including leakage, switching, and internal power.

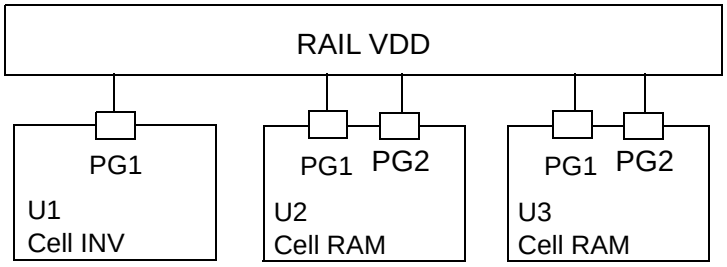
Note:

The derating value specified by the `set_power_derate –pg_pin` command has the highest precedence.

You need to set the derating factors explicitly on the internal PG pins. Derating factors set on the external PG pins are not automatically inferred on internal PG pins.

In Figure 5-1, the memory cell U2 has two PG pins: PG1 and PG2. PG1 contributes power for some part of the memory cell, and PG2 contributes power to other parts of the memory cell.

Figure 5-1 An Example for Setting Derating Factors for Each PG Pin for Multivoltage Designs



In this example, both PG1 and PG2 connect to the same rail VDD net but have different power consumptions. To set different derating factors on PG1 and PG2, use the following commands:

```
pwr_shell> set_power_derate 2.0 [list U1 U2 U3]
```

```
pwr_shell> set_power_derate –pg_pin PG1 [list U2 U3] 3.0
pwr_shell> set_power_derate –pg_pin PG2 [list U2] 4.0
```

Then run the `report_power_derate` command to list the derating values per PG pin.

```
pwr_shell> report_power_derate -pg_pin

Object Name     Object Type         Switching     Internal     Leakage
-------------------------------------------------------------------------
U2/PG1           PgPin                 4.0          4.0          4.0
U3/PG1           PgPin                 4.0          4.0          4.0
```

The following example first sets different derating factors on PG1 and PG2, and then sets a derating factor on the design. As shown in Table 5-2, the PG pin derating values have a higher precedence over other derating values.

```
pwr_shell> set_power_derate -pg_pin PG1 [list U2 U3] 3.0
pwr_shell> set_power_derate -pg_pin PG2 [list U2] 4.0
pwr_shell> set_power_derate 2.0
```

Table 5-2 Derating Factors Assigned Without and With the -pg_pin Option

	Instances				
	U1	**U2**		**U3**	
	PG1	PG1	PG2	PG1	PG2
Setting the derating value for the design	2.0	2.0	2.0	2.0	2.0
Setting the derating value per PG pin	2.0	3.0	4.0	3.0	2.0

Resetting Power Derating Factors

Run the `reset_power_derate` command to reset all the specified rail-based power derating factors for the specified object list or power groups. If no object or power group is specified, the command resets the power derating factors of all objects in the current design, including those set on the power groups.

Use the `-pg_pin` option to reset the derating factors that are applied per PG pin for selected groups or instances.

Reporting Power Derating Factors

Run the `report_power_derate` command to list rail and PG pin power derating factors. If the derating factor is applied to a rail in the current design, the rail is listed as an object to which the derating factor is applied when set at the design level or on hierarchical cells. If the derating factor is applied to a leaf cell, the object type is the PG pin to which the rail is connected.

Use the `-pg_pin` option to list the derating factors per PG pin.

For a detailed example, see Generating Reports on Power Derating Factors.

Commands Affected by the Power Derating Factor

The following commands use the power derating factor:

- `report_power`

  Reports the power of the design, with power values adjusted by the specified derating factors.

  Use the `-derate` option to display the power derating factors used in the power calculation. This option applies only to the cell-based averaged power report and time-based power report. For multivoltage cells with different derating factors specified per PG pin, the derating value is calculated by dividing the sum of the original derated values.

  The `report_power` command generates:

  - The derated averaged power report in the averaged and time-based power analysis modes.
  - The derated time-based power report in either the .fsdb or .out output format and the peak power report.

- `report_power_calculation`

  Uses the power derating factors to calculate and report power values.

- `reset_design`

  Resets the power derating factors while resetting the design.

- `estimate_clock_network_power`

  Applies the specified power derating factors to the buffer or the current design while estimating the clock network power. This command is supported only in the averaged power analysis mode.

- `get_attribute`

  Gets the derated power numbers, based on the specified power derating factor. This command can also be used to retrieve the power derating factors on cells and the current design using the power derating attributes.

Performing Averaged Power Analysis

When switching activity annotation is complete, set the `power_analysis_mode` variable to `averaged`, and then run the `update_power` command to apply the annotation on the required design objects. When you run the `update_power` command, PrimePower propagates the annotated and implied activity, and estimates the state- and path-dependent toggle rates to calculate the power values.

To scale state probabilities for internal power arcs, set the `power_scale_internal_arc` variable to `true` before running the `update_power` command. When enabled, PrimePower matches the sum of the probabilities of internal arc with the toggle rate of the pin, even if there is no default arc for the pin in the library. The default is `false`.

By default, PrimePower scales power for all the cells and library cells in the design. If you want to exclude certain library cells from internal power arc scaling, such as macro cells, specify the list of library cells to be excluded from power arc scaling using the `set_power_analysis_options -exclude_arc_scaling_libcells` command.

Run the `report_power` command to report the power values. Or you can run the `report_power_calculation` command to display the calculation of the internal power for a pin, the leakage power for a cell, or the switching power for a net. For a detailed description about how to generate power reports, see RedHawk Analysis Fusion in IC Compiler II.

PrimePower also supports power analysis with user-defined annotated power values, different clock domains, and switched power domains, as described in the following topics:

- Power Analysis With User-Defined Internal, Leakage and Switching Power
- Power Analysis With Different Clock Frequencies
- Power Analysis With Switched Power Domains

Power Analysis With User-Defined Internal, Leakage and Switching Power

Use the `set_annotated_power` command to annotate power on hierarchical blocks, black box cells, or leaf cells; and switching power on nets for budgeting. The tool overrides the internally estimated power values with the annotated power values.

To remove or list the annotated values in a report, use the `remove_annotated_power` or `report_annotated_power` command, respectively.

Note:

All annotated power represents an average power value. Power annotation is not supported in the time-based analysis flow.

Table 5-3 Commonly Used Options for the set_annotated_power Command

Option	Description
`-internal_power internal_power_value`	Annotates `internal_power_value` (in Watts) on the specified cells. When enabled, the annotated value replaces the internal power for the cells, and the switching power for all the cells which drive nets internally to the block.
`-leakage_power leakage_power_value`	Annotates `leakage_power_value` (in Watts) on the specified cells. When enabled, the annotated value replaces the leakage value calculated for the cells.
`-switching_power switching_power_value`	Annotates the `switching_power_value` on the specified nets. Do not specify this option with the `-rails` option, because net switching power can only be associated with a single rail.

Note for Annotating Power on the Hierarchical Blocks

When annotating power on the hierarchical blocks, consider the following:

- Hierarchical blocks become black boxes when they are annotated with power values. The calculated power of the cells within a block is replaced with the user-annotated value, so the cells are no longer considered in the analysis.
- Annotation of internal power on a hierarchical block takes precedence over power annotation on nets within the block. If you run the `set_annotated_power -switching_power` command on nets within a hierarchical block with the annotated internal power, the tool ignores the annotated switching power with a warning message.
- When annotating power on a lower-level cell and its parent cell, the power annotation of the parent cell has a higher precedence.

- The `-rails` option is not supported for hierarchical blocks. Internal and leakage power values are associated with the default design rail. If the `-rails` option is specified, the tool issues a warning and applies the values to the default design rail.

Reporting Annotated Power

Use the `report_annotated_power` command to report the annotated power of the current design. To report a list of cells or nets with the annotated power, specify the `-list_annotated` option. The listed cells include leaf and hierarchical cells.

When reporting annotated power for a hierarchical cell, note that:

- The power values of cells and nets within the annotated hierarchical block are reported as N/A (not available). This ensures the results of `report_power -net_power` and `report_power -cell_power` match those of the default `report_power` command.
- A hierarchical block is treated as a black box, so the power distribution among groups changes when annotation is performed on a hierarchical block and is transferred to the `black_box` power category.
- The `get_power_per_pgpin` and `get_power_per_rail` commands report all annotated power, including leakage, switching and internal power.

Here is an example:

```
pwr_shell> report_annotated_power -list_annotated
****************************************
Report: annotated power -list_annotated
Design: test
...
****************************************
Annotated powers:
----------------------------------------
1. m1_reg[15] (internal: 0.498 leakage: 0.0037]
2. m1_reg[14] (internal: 0.498 leakage: 0.0037]
3. m1_reg[13] (internal: 0.498 leakage: 0.0037]
4. m1_reg[12] (internal: 0.498 leakage: 0.0037]
                           |        |           |    NOT    |
    Cell type              | Total  | Annotated | Annotated |
---------------------------+--------+-----------+-----------+
unresolved black box cell  |     5  |        4  |        1  |
leaf cell                  |  7482  |        0  |     7842  |
---------------------------+--------+-----------+-----------+
                           |  7487  |        4  |     7843  |
```

Removing Annotated Power

Use the `remove_annotated_power` command to remove the previously annotated power for a list of cells or nets. Specify the `-all` option to remove all annotated power in the design.

Here is an example:

```
pwr_shell> remove_annotated_power -all
```

Power Analysis With Different Clock Frequencies

To perform power analysis at multiple clock frequencies by scaling the activity relative to the SDC clock, first set the `power_enable_clock_scaling` variable to `true` before the `update_power` command. The default is `false`.

Then use the `set_power_clock_scaling` command to specify the simulation clock used in event file generation and to specify the period or scaling ratio to be applied. The tool does not scale the switching activity specified by the `set_switching_activity` and `set_case_analysis` commands.

Use this feature to perform power analysis at varying clock frequencies without having to regenerate the simulation activity file.

Example 5-1 A Script Example Using the Clock Definitions From SDC File

```
set_app_var power_analysis_mode averaged
set_app_var search_path         "../src/hdl/gate ../src/lib/snps . "
set_app_var link_library        " * core_typ.db"
read_verilog            mac.vg
current_design          mac
link
read_sdc ../src/hdl/gate/abc.sdc
read_parasitics ../src/annotate/abc.spef.gz
read_saif "../sim/mac.saif" -strip_path "tb/abcinst"

set_power_clock_scaling –period 24 [get_clock clk]
set_app_var power_enable_clock_scaling true
update_power
report_power
quit
```

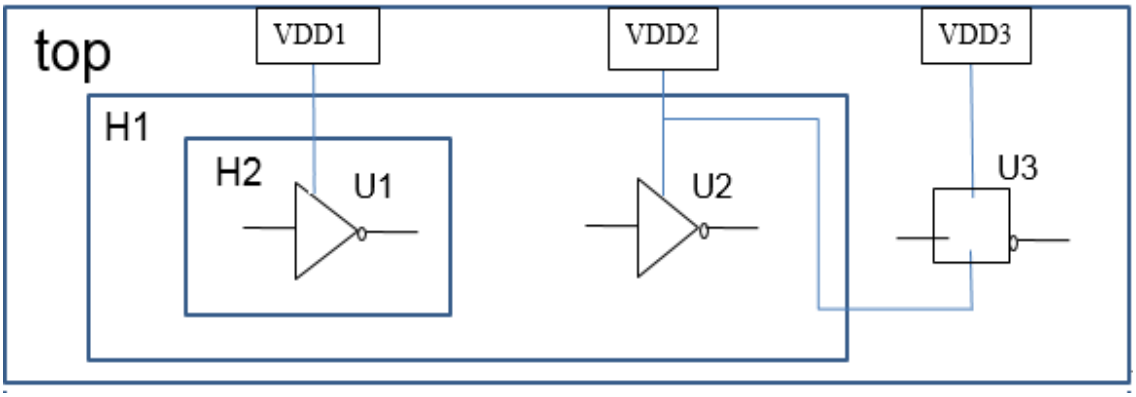
Power Analysis With Switched Power Domains

To analyze power for switched power domains in the averaged mode, first use the `set_supply_net_probability` command to set static probability on the supply nets whether or not the nets are connected to an on-chip switch cell. PrimePower uses the supply net probability to determine in which time fraction the cells connect to the powered supply nets and calculates the shutdown power.

To remove the annotated probability from a supply net, use the `-remove` option.

Taking the design in Figure 5-2 as an example, the hierarchical cells, H1 and H2, have combinational cells which are connected to the supply nets VDD1 and VDD2. The hierarchical cell U3 which is connected to VDD3 has combinational and register type cells. Supply nets VDD1, VDD2, and VDD3 are not connected to any switch cell.

Figure 5-2 A Switched Power Domain



To analyze power of a switched power domain,

1. Run the `report_power` command to generate a power report for the design.

```
pwr_shell> report_power
                          Internal   Switching  Leakage   Total
Power Group    Power      Power      Power      Power     (%)     Attrs
------------------------------------------------------------------------
clock_network  1.417e-06  2.210e-06 1.774e-07  3.804e-06  (13.05%)  i
register       8.463e-07  2.352e-07 8.988e-06  8.377e-06  (28.73%)
combinational  4.683e-06  4.081e-06 2.154e-05  3.030e-05  (71.33%)
sequential     0.0000     0.0000    0.0000     0.0000     (0.00%)
 memory        0.0000     0.0000    0.0000     0.0000     (0.00%)
 io_pad        0.0000     0.0000    0.0000     0.0000     (0.00%)
 black_box     0.0000     0.0000    0.0000     0.0000     (0.00%)

  Net Switching Power  = 6.526e-06   (15.36%)
  Cell Internal Power  = 5.254e-06   (12.37%)
  Cell Leakage Power   = 3.071e-05   (72.27%)
                         ---------
Total Power            = 4.249e-05  (100.00%)
```

2. Run the `set_supply_net_probability` command to set probability to 0 for supply nets VDD1 and VDD2. Then run power analysis and examine the changes in the reported internal, switching and leakage power values for the combinational power group.

```
pwr_shell> get_supply_nets  {"VDD1", "VDD2", "VDD3"}
pwr_shell> set_supply_net_probability [list VDD1 VDD2 ]  0 \
             {"VDD1" 0 annotated} {"VDD2" 0 annotated}
pwr_shell> update_power
pwr_shell> report_power
```

```
                          Internal   Switching  Leakage    Total
Power Group     Power     Power      Power      Power       (%)    Attrs
--------------------------------------------------------------------------
clock_network   1.417e-06  2.210e-06  1.774e-07  3.804e-06 (13.05%)  i
register        8.463e-07  2.352e-07  8.988e-06  8.377e-06 (28.73%)
combinational   2.683e-06  1.609e-06  1.069e-05  1.498e-05 (58.22%)
sequential      0.0000     0.0000     0.0000     0.0000    (0.00%)
memory          0.0000     0.0000     0.0000     0.0000    (0.00%)
io_pad          0.0000     0.0000     0.0000     0.0000    (0.00%)
black_box       0.0000     0.0000     0.0000     0.0000    (0.00%)

  Net Switching Power  = 4.054e-06   (13.90%)
  Cell Internal Power  = 5.254e-06   (18.02%)
  Cell Leakage Power   = 1.985e-05   (68.08%)
                         ---------
Total Power            = 2.916e-05  (100.00%)
```

6

Time-Based Power Analysis

This chapter describes how to annotate the switching activity files for calculating power in the time-based power analysis mode. In this mode, the tool calculates the power per event to generate power waveforms over time, as well as the averaged power. Based on the type of event-based switching activity file provided, the tool performs instantaneous peak power analysis or Cycle-Accurate Peak Power Analysis.

PrimePower supports the Delay-Aware Peak Power Analysis Flow feature to shift all the events to a given time stamp by a delay factor associated with the corresponding pin or net.

This chapter contains the following topics:

- Viewing Power Waveforms From Time-Based Analysis

Overview of Time-Based Power Analysis

In the time-based power analysis mode, PrimePower uses the events in the event-based switching activity file to calculate the power per event. It supports both RTL and gate-level activity files. When RTL-based switching activity files are provided, the tool propagates the RTL events per clock cycle to determine the activity on the non-annotated nets.

For both types of event files, the tool can accurately identify the related pin for each event to calculate power per event.

To perform time-based power analysis, set the `power_analysis_mode` variable to `time_based`. To specify additional options for the analysis, use the `set_power_analysis_options` command.

Reading Switching Activity Information in Time-Based Mode

In the time-based power analysis mode, you must provide the switching activity information for accurate power analysis results. PrimePower reads either VCD or FSDB activity files generated from the gate-level or RTL simulation.

When you run the `read_vcd` or `read_fsdb` command, the tool reads the header of the event file, and matches the design nets to the nets in the event file. Internal clock pins of ETM models must be annotated with the activity specified in the event file. This is because in the time-based mode, the tool does not derive the switching activity for these pins from the model definition.

Unannotated nets in the time-based mode are assigned with zero activity (that is, TR=0), an `activity_source` of DEFAULT, and a static probability equal to the `set_case_analysis` value found on the nets.

Table 6-1 lists the activity annotation types supported in the time-based power analysis in order of precedence from high to low.

Table 6-1 Supported Activity Annotation Types in the Time-Based Mode

Precedence Order	Activity Annotation Types
1	`set_switching_activity -force`
2	Activity files
3	Propagated from the activity files; implied from `set_switching_activity -force`
4	`set_case_analysis`

Table 6-1 Supported Activity Annotation Types in the Time-Based Mode (Continued)

Precedence Order	Activity Annotation Types
5	Logic constants
6	Implied from `set_case_analysis` and logic constants

After you run the `read_vcd` or `read_fsdb` command, the tool generates a summary report showing the percentage of nets with switching activity annotation from the event file, including nets with multiple drivers, specifically the tristate bus. Use the `report_switching_activity` command to report annotation of the switching activity.

However, when you specify the `update_power` command, the tool determines the related pins based on the conditions and events specified in the event file.

Specifying multiple event files is not supported in the time-based power analysis mode. If you use multiple `read_vcd` or `read_fsdb` commands, PrimePower generates a warning message and uses the activity information from the last `read_vcd` or `read_fsdb` command executed before the `update_power` command.

See Also

- Annotating Switching Activity

Handling VCD Files With Nonmodule Scopes

Use the `read_vcd -format systemverilog` command to specify the RTL-VCD files that are generated from SystemVerilog or VHDL simulation. The procedures contain scopes such as `begin` or `fork`. The VCD file parser ignores these scopes unless you specify the `-format` option.

Mapping the Testbench Instance to the Design Module

PrimePower creates the activity file at the testbench level, where the design under test is an instance within the HDL testbench hierarchy. To map the testbench instance to the design module that you specify as the current design, use the `read_vcd -strip_path` command. For example, if the testbench TB instantiates the design module TOP as U, specify the following command:

```
pwr_shell> read_vcd -strip_path TB/U1 ...
```

If the VCD hierarchy is unknown, use the `report_vcd_hierarchy` command to identify the appropriate path specified with the `-strip_path` option.

Note:

Even if you do not specify the top-level design in the $dumpvars task, the VCD file still lists the complete hierarchy starting with the testbench.

Using Zero-Delay and SDF-Delay Activity Files

PrimePower reads both zero-delay and SDF-delay gate-level simulation activity files in either VCD or FSDB format by using the `-zero_delay` and `-rtl` options of the `read_vcd` and `read_fsdb` command.

Use the `-zero_delay` option if the activity file is generated from zero-delay simulation. When you specify this option, the tool assumes the design is fully annotated and does not perform name mapping. The tool generates power waveforms per cycle. To calculate the instantaneous peak power, the tool requires SDF delay information.

Use the `-rtl` or `-zero_delay` option if the event file is generated from RTL simulation or gate-level zero-delay simulation. When both options are specified, PrimePower uses the name mapping set by the `set_rtl_to_gate_name` command to process the event file; the `update_power` command in the subsequent stage uses zero-delay simulation to create events on the unannotated nets.

Table 6-2 shows how the tool handles different types of event files in the time-based power analysis mode with or without the `-rtl` or `-zero_delay` option.

Table 6-2 Supported Event File Types With or Without the -rtl and -zero_delay options

	-rtl option	-zero_delay option	Name mapping	Event propagation	Enforce cycle-accurate peak power
VCD	Disabled	Disabled	Off	Off	No
FSDB	Disabled	Enabled	Off	Off	Yes
SAIF	Enabled	Enabled	On	On	Yes

When neither the `-rtl` nor `-zero_delay` option is specified, the tool assumes the activity file is generated from a gate-level SDF simulation. PrimePower generates power waveforms over time, at the time resolution of the activity file, and identifies the instantaneous peak power.

When you specify the gate-level activity file using the `read_vcd` or `read_fsdb` command, the tool generates a warning message if more than 5 percent of the nets are annotated and the switching activity is not propagated through these nets. The non-annotated nets are assumed to have zero activity and remain at a constant logic value of 0.

Handling Large VCD Files

The gate-level VCD file can be very large. Although PrimePower can process VCD files larger than two gigabytes on all platforms, avoid generating and storing large VCD files.

You can generate compressed formats, such as FSDB or gzipped VCD, to reduce the activity file size. Internally, the tool converts the data from the compressed files to the VCD format and inputs to the analysis engine. The converted data is not stored by the tool.

Alternatively, the activity data can also be sourced directly from the HDL simulation to PrimePower. 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 source the activity data directly, follow these steps:

1. Set up the HDL simulation to generate an activity file. For example, when generating a large VCD file from a Verilog simulation, the testbench must include the $dumpvars or $dumpfile tasks to generate the VCD data. Use the same file name in the $dumpfile task and the `read_vcd` command.

   This file name is used as a pipe name, so both tools know where to write and read the data. The process does not create a large data file on disk.

2. Specify the commands or run a script to invoke the HDL simulation with the `-pipe_exec` option. For example, you run a script that executes a VCS simulation and contains the following command:

   ```
   vcs -R -f arguments -l log
   ```

   Use one of the following commands to invoke simulation within PrimePower to source the VCD data directly:

   ```
   read_vcd -pipe_exec run_vcs
   ```

   or

   ```
   read_vcd -pipe_exec "vcs -R -f arguments -l log" ...
   ```

Checking Activity Data

After you load either RTL or gate-level VCD or FSDB files, run the `check_activity` command on the specified pins (either hierarchical or leaf) to check the quality of the vectors before proceeding to the power analysis stage. When checking is complete, PrimePower updates the activity status of the queried pins with the `activity_checker_status` attribute using the initial state information in the activity file for each pin, and reports the number of the pins for each state in the summary report.

Use the `-trace_fanin` option if you want to check the fanin cone of the pins for the X and Z states. The tool identifies pins in the fanin cone and reports the pin status in the summary report.

Table 6-3 lists the values of the `activity_checker_status` attribute.

Table 6-3 The activity_checker_status Attribute Values

The activity_checker_status attribute value	Pin annotation
state_x	Pin that is annotated with the constant X values
state_z	Pin that is annotated with the constant Z values
unannotated	Unannotated pins
unconnected	Pin that is not specified or unconnected.
pass	Pin that is annotated with valid logic states

The `check_activity` command reports the following information in the report.

- The number of unannotated pins
- The number of unconnected pins
- The number of any X state pins with time interval
- The number of any Z state pins with time interval
- The number of the annotated pins
- The total number of the specified pins

Note:

The `check_activity` command is supported only in the time-based power analysis.

The number of the unconnected pins should always be zero in the summary report. If not, fix the connection issues for these unconnected pins.

Example

The following example reports the activity states for each pin in the design, and queries the `activity_checker_status` attribute for the I_fw_ram_rdata[0] pin.

```
pwr_shell> set power_analysis_mode time_based
pwr_shell> read_fsdb -time {t1 t2}
pwr_shell> check_activity [get_pins *]

Information: Enabling native fsdb reader.
Information: Reading fsdb file 'test.fsdb'
```

```
** FSDB version : 5.5
Created by check_activity command on ...
=============================================================================
State         State         State     State           Pass
Unconnected   X             Z         Unannotated     0/1
-----------------------------------------------------------------------------
    0         1056          8         0               628
( 0.00%)     (62.41%)       (0.47%)   (0.00%)         (37.12%)
=============================================================================

=============================================================================
State     Time Interval                Net
-----------------------------------------------------------------------------
  Z     (77425.12300000001 unknown)   u_apb2eb/O_pslverr
  Z     (77425.12300000001 unknown)   SYNOPSYS_UNCONNECTED_1
  Z     (77425.12300000001 unknown)   SYNOPSYS_UNCONNECTED_2
  Z     (77425.12300000001 unknown)   SYNOPSYS_UNCONNECTED_3
……
  X     (999817.007 999867.007)       I_fw_ram_rdata[0]
  X     (999877.007 999897.007)       I_fw_ram_rdata[0]
  X     (999907.007 unknown)          I_fw_ram_rdata[0]
=============================================================================

pwr_shell> get_attribute [get_pins u_clue_top/I_fw_ram_rdata[0]] \
   activity_checker_status
state_x
```

Performing Vector Analysis

During gate-level analysis, use the `write_activity_waveforms` command to identify the simulation windows with the highest activity which are likely to produce the peak power. Vector analysis provides information about the vectors by plotting the average toggle rate per interval, over the specified time period. PrimePower reads the activity file and writes an activity waveform for the top-level module of the activity file.

The tool partitions the total simulation time into intervals. For each interval, the average toggle rate is calculated using the following formula:

$$\text{Average Toggle Rate} = \frac{\text{Number of Toggles on All Signals Per Interval}}{\text{Number of Signals}}$$

View the activity waveforms to determine if the testbench is simulated as expected and if the vectors have sufficiently covered the design to use them as inputs for power analysis. The coverage per interval is calculated using the following formula:

$$\text{Coverage} = \frac{\text{Number of Signals With at Least One Toggle}}{\text{Number of Signals}}$$

To analyze power only for the high activity windows, specify the analysis windows using the `read_vcd -time` command.

Example 6-1 shows how the `-peak_window` option affects the report generated by the `report_activity_waveforms` command. By specifying the `-peak_window` option as 5, the reported peak interval is 5 nanoseconds, although the interval for the waveform is 0.2 nanoseconds.

Example 6-1 Generating and Reporting Activity Waveforms

```
pwr_shell> write_activity_waveforms -vcd myvcd.vcd -output myvcd.out \
          -interval 0.2 -peak_window 5 -hierarchical_levels 2
pwr_shell> report_activity_waveforms
**************************************
Report : Time-Based Switching Activity
Activity File: myvcd.vcd
**************************************
Block                  # Signals  Peak           Peak         Average
Name                              Interval (ns)  Toggle Rate  Toggle Rate
-------------------------------------------------------------------------
GOOD_upstream_router_tb    1383  4.400-9.400    0.00145      0.000882
test_GOOD_upstream_router  1335  4.400-9.400    0.000899     0.000548
-------------------------------------------------------------------------
```

Setting the Options for Time-Based Power Analysis

The `set_power_analysis_options` command controls the cells that are analyzed, the waveform data output, and the tracking of state-dependent, path-dependent toggle rates.

Generating Power Waveform Data

By default, in time-based mode, PrimePower generates power waveforms and reports peak power for all the design hierarchies when you run the `update_power` command. The generated waveforms combine both dynamic and leakage power of the design for all rails.

To generate waveform data for the leaf cells or the top-level design,

```
pwr_shell> set_power_analysis_options -include top
pwr_shell> update_power
```

To prevent generation of power waveforms,

```
pwr_shell> set_power_analysis_options -waveform_format none
pwr_shell> update_power
```

To generate separate power waveforms for dynamic and leakage power,

```
pwr_shell> set_power_analysis_options \
    -separate_dyn_and_leak_power_waveform
pwr_shell> update_power
```

To generate power waveforms per rail or supply net,

```
pwr_shell> set_power_analysis_options -multi_rail_waveform
pwr_shell> set_app_var power_enable_multi_rail_analysis true
pwr_shell> update_power
```

This allows you to analyze the leakage or dynamic power dissipation pattern over simulation time.

For example, if the design has 2 hierarchical instances, u1 and u2, and 2 rails, VDD1 VDD2, 8 waveforms will be generated:

u1_VDD1_dyn, u1_VDD1_leak

u1_VDD2_dyn, u1_VDD2_leak

u2_VDD1_dyn, u2_VDD1_leak

u2_VDD2_dyn, u2_VDD2_leak

See Also

- Multirail Power Analysis

State-Dependent, Path-Dependent Tracking

By default, PrimePower does not store the state-dependent, path-dependent toggle rates in the time-based analysis mode. When the tool processes the VCD file, it computes power for each event in the file.

To store the state-dependent, path-dependent toggle rates so that they can be written to a SAIF file, use the `set_power_analysis_options -sdpd_tracking` command. This causes the PrimeTime tool to store the toggle rates for black box cells, as the tool uses the accurate toggle rate information for accurate power results.

Use the `-sdpd_tracking_cells` option to specify a list of cells for which you enable the `-sdpd_tracking` option. When power analysis is complete, use the `write_saif` command to write out the time-based toggle rate information to a SAIF file, and use this file in the averaged power analysis mode.

Performing Time-Based Power Analysis

To analyze power in the time-based mode, set the `power_analysis_mode` variable to `time_basedd`, specify the necessary options with the `set_power_analysis_options` command, and then run the `update_power` command. The tool processes the events in the VCD file and writes out power waveforms.

Before processing the events in the activity file, the tool calculates the input transition time and output load for every instance in the design. These values are stored to access the appropriate energy values in the library power tables for the events in the activity file.

For transitions on the output pins of the cells, PrimePower finds the related pin by searching for the input transitions that caused the output transition. After identifying the pin, the tool accesses the appropriate power arc and analyzes the energy for that transition based on the power table indexes.

Use the `report_power` command to generate a report showing the largest power value (peak power) and the time at which it occurs. You can also query for internal, switching and leakage power for cells though the `get_attribute` command for debugging purposes. Querying for switching power on nets is also supported.

By default, PrimePower reports internal power at the cell boundary, which makes it difficult to break down the internal power and debug the source of the internal power. To query for internal power on pins, set the `power_enable_pin_internal_power_query` variable to `true` before the `update_power` command.

For a detailed description about how to generate power reports, see Chapter 13, “RedHawk Analysis Fusion in IC Compiler II.”

To run a power update without generating power waveforms, set the `waveform_format` option of the `set_power_analysis_options` command to `none`.

Instantaneous or Event-Based Peak Power Analysis

When you specify gate-level SDF simulation activity, PrimePower calculates the power for every event. To calculate peak power values, PrimePower

1. Analyzes every event in the event-based activity file.
2. Checks for associated power arcs and related pins.
3. Accesses the appropriate energy table in the logic library.
4. Distributes the energy evenly for every event over the transition time for the inputs, and over the path delay for the outputs.
5. Generates power waveforms from the superposition of the distributed energy.

From the power waveforms, you can identify the instantaneous peak power.

By default, the tool generates power waveforms at the event-based activity file resolution. To change the resolution, use the `set_power_analysis_options -waveform_interval` command.

Figure 6-1 shows the peak power analysis result using a gate-level VCD file with the default time interval. The tool calculates energy for every event and distributes it over the path delay.

Figure 6-1 Peak Power Analysis for Gate-Level VCD With Default Time Interval

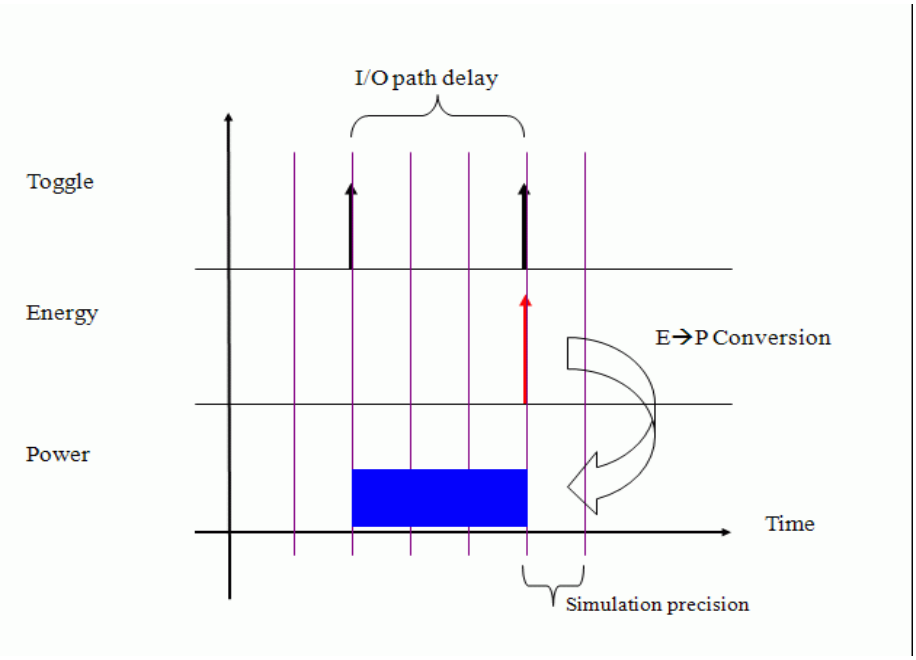
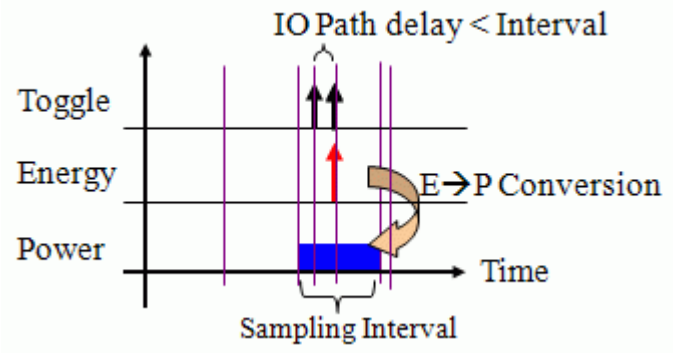


Figure 6-2 shows the peak power analysis result using a gate-level VCD file with a user-specified time interval. PrimePower calculates the energy for every event and distributes it over the specified time interval if the time interval is greater than the I/O path delay. If the time interval is less than I/O path delay, the tool distributes the energy over the I/O path delay.

Figure 6-2 Peak Power Analysis for Gate-Level VCD With User-Specified Time Interval

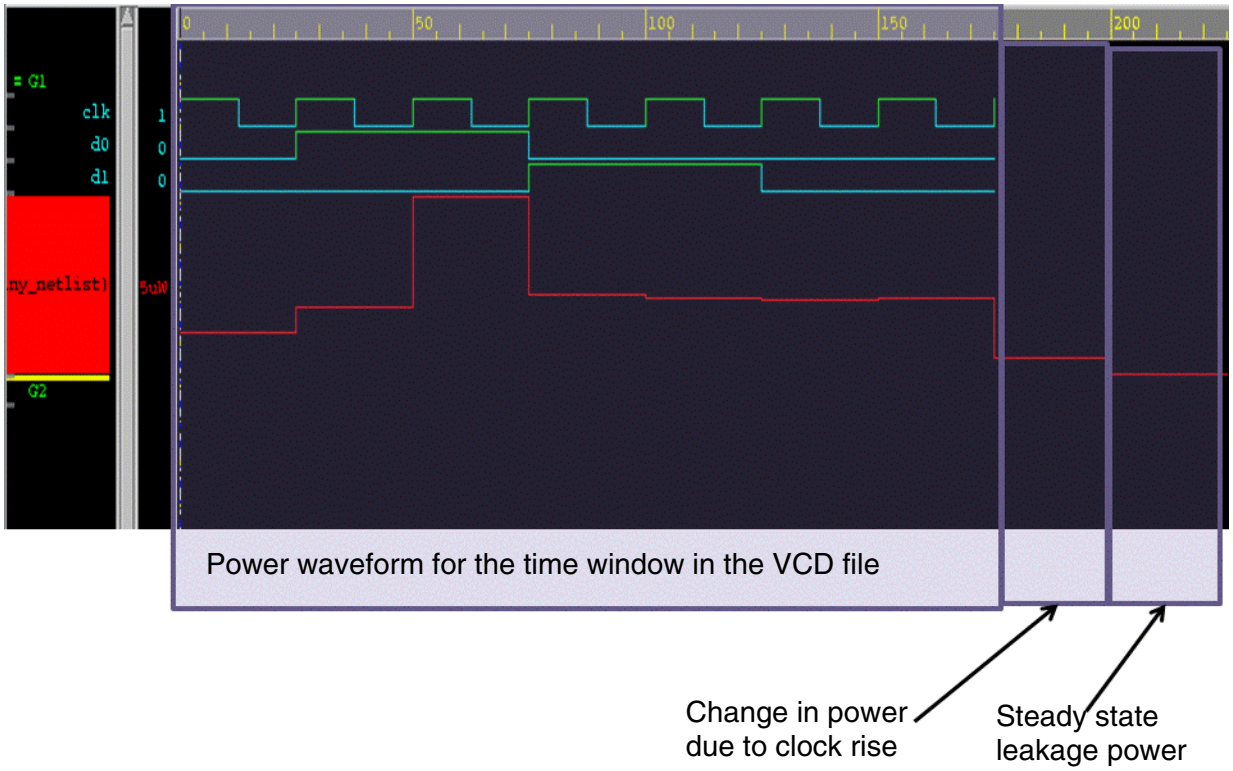


Peak Power Waveforms

During peak power analysis, PrimePower generates a time-based power waveform and reports the peak power. The tool reports power variation at every clock cycle and is not limited by the time window in the activity file. Sometimes the power waveform contains more cycles than those in the activity file. This is because the tool monitors the cycles until it detects no further power variations, as shown in Figure 6-3.

Use the `report_power` command to report the power values that match the values shown in the power waveform.

Figure 6-3 Understanding Instantaneous or Event-Based Peak Power Waveforms



When calculating the power per event, PrimePower considers the value setting of several variables described in the following topics:

- Variables Affecting Instantaneous Peak Power Analysis
- Modeling Special Conditions
- Initial X-State Handling
- Unmatched States

Variables Affecting Instantaneous Peak Power Analysis

The `power_table_include_switching_power` variable indicates how the power tables have been characterized. The default is `true`.

PrimePower assumes that the library power tables are characterized such that half the switching activity was subtracted from both the rising and the falling edges, as shown in the following formulas.

Formula 1

```
Internal_energy_rise =
     Total_energy_rise - leakage_energy -
     0.5*switching_energy
```

Formula 2

```
Internal_energy_fall =
     Total_energy_fall - leakage_energy -
     0.5*switching_energy
```

Because switching energy (CV^2) incurs only on the rising edge, the tables' values in this case do not reflect the true internal energy.

$\texttt{Eint\_table(rise)} = \texttt{Eint(rise)} + 0.5CV^2$

$\texttt{Eint\_table(fall)} = \texttt{Eint(fall)} - 0.5CV^2$

When you set the variable to `false`, the tool assumes that the library internal energy tables contain only the internal energy. During characterization, the tool derives the rising and falling energy using the following formulas.

Formula 3

```
Internal_energy_rise =
     Total_energy_rise - leakage_energy - switching_energy
```

Formula 4

```
Internal_energy_fall =
     Total_energy_fall - leakage_energy
```

Specifying the `power_table_include_switching_power` variable does not affect the average power because multiple events average out the difference. However, the variable setting affects the power waveforms, particularly for small designs and vector sets. For optimal peak power analysis, set this variable appropriately.

Contact the library vendor to identify the best formula used for characterization. A workaround is to inspect the falling power tables. If most of them have negative numbers, it is likely that formulas 1 and 2 are used in characterization.

Example 6-2 shows a time-based report indicating peak power values, including the glitch, xtransition, and peak power values.

Example 6-2 Reporting Peak Power Values: Glitch, Xtransition, Peak Power

```
*****************************************
Report: Time Based Power
Design: mac
...
*****************************************
  Attributes
      i  -  Including register clock pin internal power
      u  -  User-defined power group
Power            Internal   Switching   Leakage    Total
Group            Power      Power       Power      Power   (     %)  Attrs
--------------------------------------------------------------------------
clock_network    7.133e-04 0.0000000  0.0000000   7.133e-04(19.43%)   i
register         4.703e-04 1.095e-04  6.660e-08   5.799e-04(15.80%)
combinational    8.964e-04 1.208e-03  1.836e-07   2.104e-04(57.33%)
sequential       5.289e-05 2.200e-04  9.163e-09   2.729e-04( 7.43%)
memory           0.0000    0.0000     0.0000      0.0000(0.00%)
io_pad           0.0000    0.0000     0.0000      0.0000(0.00%)
black_box        0.0000    0.0000     0.0000      0.0000(0.00%)
--------------------------------------------------------------------------

Net Switching Power  = 1.537e-03 (41.88%)
Cell Internal Power  = 2.133e-03 (58.11%)
Cell Leakage Power   = 2.594e-07 ( 0.01%)
 ---------------------------------------
Total Power          = 3.670e-03 (100.00%)
-----------------------------------------

X Transition Power   = 2.171e-07
Glitching Power      = 2.942e-05
Peak Power           = 0.0970
Peak Time            = 1956
```

Modeling Special Conditions

The power consumption for the following conditions is part of the dynamic switching power. It affects both internal power and switching power.

- Glitches
- Z states
- X states

Glitches

Glitches are small pulses, which do not achieve full logic levels because of the narrow pulse width. In time-based mode, the tool determines whether a pulse is small enough to be considered a glitch by comparing the pulse width with the sum of the rise and fall transition time obtained from timing analysis, and then scales power accordingly.

Use the `report_power` command to generate a report on the estimated glitch power.

For a detailed description about calculating glitch power in time-based mode, see Glitch Power Analysis in Time-Based Mode.

Z State

The tool assumes no power is consumed when a logic level changes to or from the Z state. For example, no power is counted for the 0->Z or 1->Z transition.

If a logic state after the Z state is different from the original state during a transition, this transition is considered a full transition and the power is counted. For example, the transition 0 -> Z -> 1 or 1 -> Z -> 0 is considered one full transition, while transition 1 -> Z -> 1 or 0 -> Z -> 0 is not a transition and consumes no power.

When processing the Z state, the tool checks the value before and after the Z state to determine whether there was a logic level change.

X State

By default, every time a logic level changes to or from the X state, the tool considers it to be 1/2 of a normal transition. For example, 0 -> X is a 1/2 transition and 0 -> X -> 0 or 0 -> X -> 1 is one full transition. This behavior is controlled by the `power_x_transition_derate_factor` variable, which by default is set to `0.5`.

To change the scale factor, modify the setting of the variable. Setting this variable to `1` results in pessimistic power reports.

The `power_match_state_for_logic_x` variable determines how the tool interprets X state in the Boolean function. The default is `0`, meaning the tool by default considers X state logic to be 0. This variable is not used for propagating the logic values; instead it is used to evaluate the WHEN conditions and the functionality for the pins which have the logic X.

Initial X-State Handling

When processing VCD data, PrimePower processes every event. Several VCD files contain X states at initialization when the tool is propagating input values.

The `power_include_initial_x_transitions` variable determines if the tool calculates power during initialization or not.

By default, the variable is set to `false` and the tool ignores the switching and internal power consumed by the transition of nets from an unknown to a known state.

Set the `power_include_initial_x_transitions` to `true` if you are concerned with the power at initialization, so that the power peak generated during the circuit initialization is included in the power analysis.

Unmatched States

When PrimePower processes the events in the switching activity file, events for which there is no matching condition in a power table might occur. PrimePower generates statistical information that includes

```
X % tables on output pins matched
X % tables on input  pins matched
```

Most cells do not have input-pin-based tables, thereby resulting in 100 percent pin matching. However, for output cells, if the event does not match any of the state definitions in the power table, the event reduces the percentage of tables on output pins that matched. For these events, by default, PrimePower provides an estimate of the power consumption based on the average value of all the available tables for that output pin.

To ignore the event, set the `power_estimate_power_for_unmatched_event` variable to `false` before the `update_power` command.

Glitch Power Analysis in Time-Based Mode

Glitches, in electronics, are undesired transitions before a signal settles to its intended value. They often produce pulses whose duration is less than the specified minimum, but the definition can be extended to include any additional transition within a clock cycle. These unwanted transitions can cause the circuit to malfunction and consume power.

Identifying Glitches

In time-based mode, PrimePower first checks if the cell's input pulse width is smaller than the maximum cell delay whenever an input event occurs.

```
input_pulse_width < (max_cell_delay + tolerance)
```

where `max_cell_delay` is derived from timing analysis using the following command:

```
pwr_shell> report_delay_calculation -from input_pin -to output_pin
```

If the input pulse width is less than the maximum cell delay, PrimePower determines whether the associated output pulse is small enough to be considered a glitch by comparing the pulse width with the sum of the rise and fall transition time obtained from timing analysis:

$pulse\_width < (Transition_{rise} + Transition_{fall}) / 2$

You can obtain the pulse width from the simulation and use the `report_power_calculation` command to obtain the rise and fall transition time on the specified instance.

Calculating Glitch Power

After a pulse is identified as a glitch, PrimePower calculates the power for the glitch using the following formula:

```
glitch power = scaling_ratio * (dynamic power when there is no glitch)
```

where

$scaling\_ratio = [(pulse\_width*2) / (rise\_ramp+fall\_ramp)]^2$

Both the internal and net switching power of the glitch transitions are scaled. The leakage power is not scaled, because it is independent on the frequency of the activity.

To achieve a more accurate glitch power value, use the `power_full_transition_glitch_scaling` variable to specify a scaling factor for scaling the transition time calculated by the tool. The scaling factor must be greater than 1 and less than 10. The default is 1.

However, if the pulse width is smaller than the `average_slew` value, the tool calculates the scaling ratio using the `average_slew` value, without applying the specified scaling factor. If no glitch is detected, the tool checks for the glitch using the scaled slew.

For example, if `pulse_width` is 0.4; `avg_slew` is 0.5; and the scaling factor is set to 2.5, the calculated `scaling_ratio` is

```
glitch_ratio = (0.4/0.5)^2 =  0.64
```

If the pulse width is 0.6; the average slew is 0.5; and the scaling factor is set to 2.5, the scaling ratio is calculated by the following formula:

```
scaling_ratio  = (0.6/(2.5*0.5))^2 =  0.96
```

Note:

The `power_full_transition_glitch_scaling` variable is available only in the time-based mode.

The scaled glitches are tracked as glitch toggles, and can be queried with the `glitch_rate` attribute.

Reporting Glitch Power

During power analysis, power consumption of transitions identified as glitches are scaled to accurately report power consumption of pulses as glitch power. Use the `report_power` command to generate a report on the estimated glitch power.

Example

```
pwr_shell> report_power
Net Switching Power = 2.252e-03 (44.82%)
Cell Internal Power = 2.773e-03 (55.18%)
Cell Leakage Power = 2.594e-07 (0.01%)
---------
Total Power = 5.025e-03 (100.00%)
X Transition Power = 2.171e-07
Glitching Power = 7.569e-04
Peak Power = 0.0968
Peak Time = 1956.00
```

Scaling Internal Power

PrimePower honors the Liberty attribute `internal_power_calculation` under the `char_config` group when scaling internal power during power calculation. The attribute has precedence over the `power_table_include_switching_power` variable.

Here are the values for the attribute in the cell description in Liberty format:

```
   char_config() {
     internal_power_calculation : exclude_switching_on_rise |
        exclude_switching_on_rise_and_fall | include_switching;  }
```

Use the `internal_power_calculation` attribute to specify whether switching power is included during internal power calculation, or whether both or either of `rise_power` and `fall_power` are excluded from internal power calculation.

Example 1

```
char_config() {
  internal_power_calculation : exclude_switching_on_rise
    rise_power table value = total dynamic energy - C * V2
    fall_power table value = total dynamic energy }
```

Example 2

```
char_config() {
  internal_power_calculation : exclude_switching_on_rise_and_fall
    rise_power table value = total dynamic energy - 1/2 * C * V2
    fall_power table value = total dynamic energy - 1/2 * C * V2 }
```

Example 3

```
char_config() {
  internal_power_calculation : include_switching }
```

Setting the `internal_power_calculation` attribute to `include_switching` indicates that the values of both `rise_power` and `fall_power` groups are total dynamic energy.

Native Distributed Analysis in Time-Based Mode

If you have a large gate-level FSDB file (either zero-delay or SDF annotated) for time-based power analysis, use the `power_enable_concurrent_event_analysis` variable to enable the multi-process distributed processing for runtime improvement.

To enable the multi-core distributed processing flow for time-based power analysis,

1. Set the `power_enable_concurrent_event_analysis` variable to `true`.

   Note:
   PrimePower can optimize the maximum number of the core processes based on the size of the FSDB file and the design, with a maximum limit of 4.

2. Run the `read_fsdb` command to read the activity information. Specify other options as necessary.

3. Run the `update_power` command to perform time-based power analysis.

Script Example

```
## PrimePower VARIABLES ##
set power_analysis_mode time_based
set power_enable_concurrent_event_analysis true

## READ ACTIVITY (gate level - zero_delay of SDF )
read_fsdb –time {t1 t2} -strip_path tb/dut test
## Run power analysis and report power
update_power
report_power
################################################
Power Report
################################################
  Net Switching Power  =    0.6750   (48.50%)
  Cell Internal Power  =    0.7050   (50.66%)
  Cell Leakage Power   =    0.0117   ( 0.84%)
                         ---------
Total Power            =    1.3918  (100.00%)
X Transition Power     =    0.1100
Glitching Power        =    8.861e-03
Peak Power             =    64.0206
Peak Time              =    10525.712
```

Conditional Power Analysis

To select specific time segments, use the `-time` and `-when` options with the `read_vcd` or `read_fsdb` command.

Use the `-time` option to analyze the selected time segments of a VCD or an FSDB file. With this, you can analyze power when the design is operating in a particular state or mode. Alternatively, you can identify the time segments of the simulation in which the testbench forced the design to be in a particular mode. Specify these time segments using the `-time` option. The unit for this option is nanoseconds.

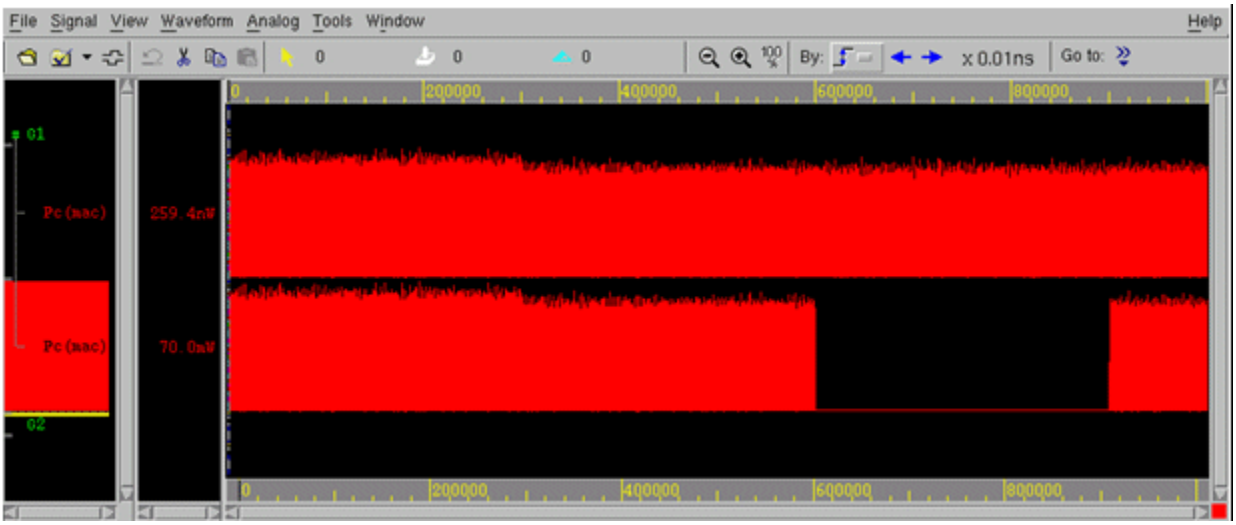
Use the `read_vcd -when` command to simplify the process of identifying the time segments. The `-when` option takes a Boolean expression as an argument, and it accepts design nets or pin names as variables in the expression. When the Boolean expression evaluates to `true`, the tool selects the time segments from the event file. The tool excludes those time segments where the logical values on nets and pins are set to `false` when performing analysis.

The following example shows the portion of the event file in which the N5 signal is `true`.

```
pwr_shell> read_vcd "../sim/vcd.dump" -strip_path "tb/macinst" -when {N5}
```

In Figure 6-4, the power waveform at the top is generated without the `-when` option. The power waveform at the bottom is generated with the `read_vcd -when {N5}` command. The `N5` signal is `true` from the beginning of the simulation until 6µs, `false` between 6µs to 9µs, and `true` from 9µs to the end of the simulation.

Figure 6-4 Waveforms With and Without Using the -when Option of the read_vcd Command



Viewing Power Waveforms From Time-Based Analysis

When time-based power analysis is complete, PrimePower generates power waveforms that display power numbers with respect to time, in the following supported formats:

- FSDB (.fsdb): Compact binary format that can store both analog and digital signals. This is the default format.
- Synopsys .out: ASCII format that supports both analog and digital signals.

To specify the output waveform format, run the following command before performing power analysis:

```
pwr_shell> set_power_analysis_options –waveform_format [fsdb|out|none]
```

When the `–waveform_format` option is to `none`, PrimePower does not generate any power waveforms.

You can view the waveform data either in the PrimePower GUI or any waveform viewer that supports the FSDB or .out format. Table 6-4 lists the supported waveform viewers and waveform file types.

Table 6-4 Supported Waveform Viewers

Waveform Viewers	FSDB	Out
nWave	Supported	Supported
Custom WaveView	Supported	Supported
CosmoScope	Supported	Not Supported

See Also

- Viewing Power Waveforms

7

Multivoltage Power Analysis

This chapter describes multivoltage power analysis in the following topics:

Introduction to the Multivoltage Infrastructure

PrimePower analyzes power for designs with different power supply voltages for different cells. Using the multivoltage infrastructure, you can choose between two distinct methods to calculate power for designs with different power supplies. These methods are not interchangeable.

- The IEEE 1801 standard also known as Unified Power Format (UPF)
  - Use the UPF file to specify your power intent.
- Alternate or non-UPF method
  - Use power domains, power nets and power and ground pins to specify your power intent.
  - Group cells based on the design power rails, and identify the power consumption of the design per rail.

All the methods support the following capabilities:

- Concurrent multirail power analysis.
- Accurate power calculation for cells with multiple power supplies by using internal and leakage power tables with liberty power-and-ground pin syntax.
- Power-gating-aware power calculation in which the tool recognizes power on and off states for design blocks and reflects power-saving technology in leakage power calculation.

  PrimePower supports the following types of power switches:
  - Fine-grain and coarse-grain switches
  - Header and footer switches
- Voltage scaling for power calculation

Voltage and Temperature Scaling Between Libraries

Multivoltage designs that contain cells operating at different operating conditions require libraries characterized for multiple voltages. To simplify multivoltage analysis, PrimePower supports voltage and temperature scaling. This feature enables you to analyze the power consumption of the specified multivoltage designs that do not have libraries characterized at the preferred operating condition. When libraries for the preferred temperature and voltage are available, the tool uses the library data to calculate power.

However, if the libraries are not characterized for the preferred operating conditions, the tool uses interpolation to derive appropriate power values. This reduces the need to have separate libraries for each incremental voltage and temperature when performing multivoltage analysis. This feature is supported in both the averaged and time-based power analysis modes.

Scaling is supported for multirail cells as well as single-rail cells. If libraries are not available in the scaling group, the tool performs single-rail scaling. The following guidelines apply when defining libraries that are a part of a scaling group:

- The libraries that you specify must satisfy the following consistency checks:
  - The same set of cells is characterized across the libraries. If this check fails, the scaling group is not created.
  - The same set of power tables is present for each cell across the libraries within a scaling group. If this check fails, scaling does not occur and the tool generates a warning message.
  - The power tables are uniquely identified. If this check fails, the tool assumes that the power tables are presented in an identical order across all libraries.
- Define the scaling relationships between the libraries using the `define_scaling_lib_group` command.
- The specified libraries must be characterized at different voltage and temperature coefficients.
- List the libraries in scaling groups only one time.
- Define the default libraries using the `link_path` variable. If scaling is not performed, PrimePower uses the libraries specified by the `link_path` variable. If scaling is needed, link the design after you read in the scaling libraries. For example,

  ```
  pwr_shell> define_scaling_lib_group {lib1.db lib2.db lib3.db}
  ```

  Then set the `link_path` variable.

  ```
  pwr_shell> set_app_var link_path "* lib2.db"
  pwr_shell> link_design
  pwr_shell> define_scaling_lib_group {lib1.db lib2.db lib3.db}
  ```

- Every library, which is part of a scaling group, is removed if one of the scaling groups is removed by using the `remove_lib` command.
- If relinking the design, rerun the `define_scaling_lib_group` command to identify the libraries that are part of a scaling group.

Specify the operating conditions for analysis using the `set_operating_conditions` and `set_voltage` commands. If libraries that are characterized for the specified operating condition are available, PrimePower uses the data directly. If not available, the tool verifies

if the operating conditions are within the range defined by scaling group. If the operating conditions are outside the range, the tool generates an error message and accesses the data from the primary linked library. If within the range, the power values are scaled. Temperature scaling of internal power is performed with linear interpolation. Both voltage scaling of leakage and internal power, and temperature scaling of leakage power are performed using the nonlinear interpolation method.

If scaling libraries are not available, use the `link_path_per_instance` variable to specify to libraries for different instances of the same cell type.

For more information about the `link_path_per_instance` variable, see the *PrimeTime User Guide*.

Multirail Power Analysis

During multi-process distributed power analysis, by default the tool analyzes power for all power rails. To check power per rail, specify the rails or supply nets using the `current_power_rail` command and then run the `update_power` command on each rail or supply net.

To check the power contribution of each rail in a single run for time-saving purposes, enable the multirail mode by setting the `power_enable_multi_rail_analysis` variable to `true`. Then calculate power consumption of the design by running the `update_power` command, and report power per rail by using the `report_power -rails` command.

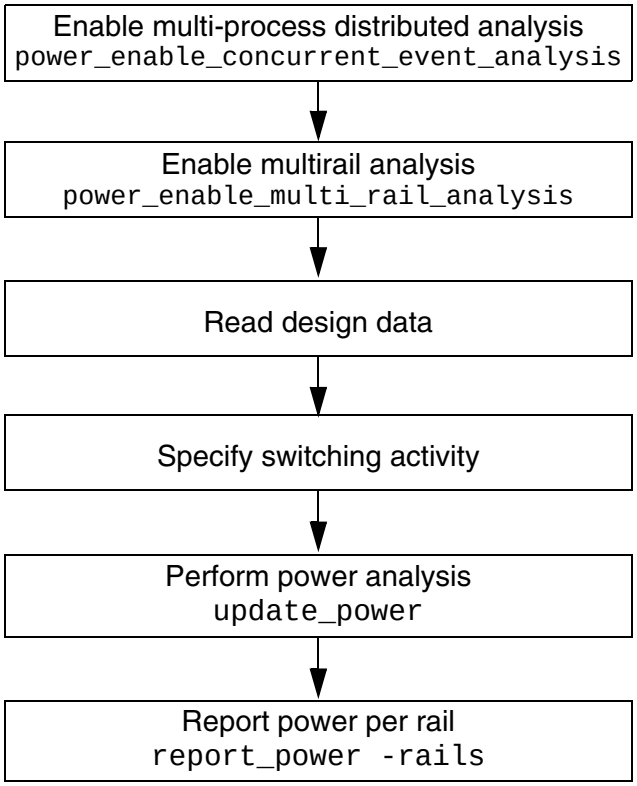
To generate waveforms for the specified rails in time-based analysis mode, specify the supply nets by using the `-multi_rail_waveform` *`supply_net_list`* option of the `set_power_analysis_options` command before the `update_power` command.

Note:

To perform multirail power analysis along with multi-process distributed analysis, you need to enable both the multirail and multi-process distributed analyses by setting the `power_enable_concurrent_event_analysis` and `power_enable_multi_rail_analysis` variables to `true`.

Figure 7-1 shows the steps to calculate power per rail in a design.

Figure 7-1 Multirail Analysis Flow



Annotating Power Per Rail

When annotating power values on multivoltage macro cells, use the `-rails` option of the `set_annotated_power` command to perform annotation of each rail or UPF supply net. If you do not specify the `-rails` option, the power values specified using `set_annotated_power` command are divided evenly between the power rails of the multivoltage macro cell. During analysis, when you run the `update_power` command, the tool uses the annotated power values for the specified macro cells when calculating the total design power for each rail.

As shown in Example 7-1, cell U1 has rail connections of VDDA, VDDB, and VDDC. The first `set_annotated_power` command annotates the internal power with 0.1 and leakage power with 0.0 for the cell U1 on rail VDDA. The second `set_annotated_power` command annotates the internal power with 0.2 and leakage power with 0.1 for cell U1 on both VDDB and VDDC power rails. The total annotated internal power on cell U1 is 0.5, and the total annotated leakage power on cell U1 is 0.2.

Example 7-1 Setting the Annotated Power on Multiple Rails

```
pwr_shell> set_annotated_power -internal_power 0.1 -leakage_power 0.0 \
    -rails VDDA U1
pwr_shell> set_annotated_power -internal_power 0.2 -leakage_power 0.1 \
    -rails {VDDB VDDC} U1
```

Example 7-2 shows rail-independent values applied to cell U1. The `set_annotated_power` command annotates internal power of 0.3 and leakage power of 0.0 on cell U1. During power analysis, these rail-independent annotated power values are divided equally over all the associated power rails of the cell. The `report_power` command reports the annotated power of 0.1 for the cell.

Example 7-2 Setting and Reporting the Annotated Power on Power Rail

```
pwr_shell> set_annotated_power -internal_power 0.3 -leakage_power 0.0 U1
pwr_shell> report_power -rails VDDA -cell_power U1
```

Both the `report_power` and the `report_annotated_power` commands support the `-rails` option. When you specify a list of rails with the `-rails` option, the tool reports the rail-based information and statistics for the annotated power. To report rail information of all the rails, set the `-rails` option to `all`.

Example 7-3 shows the report generated by the `report_annotated_power` command using the `-rails` and `-list_annotated` options.

Example 7-3 Report Generated by the report_annotated_power -rails Command

```
**************************************
Report : annotated_power -rails -list_annotated
Design : test
...
**************************************
Annotated cell powers:
------------------------------
1.  U0(VDD1)  (internal: 1e-07  leakage: 1e-11)
2.  U0(VDD2)  (internal: 2e-07  leakage: 2e-11)

                           |        |           |    NOT    |
 Cell type                 | Total  | Annotated | Annotated |
-----------+--------------+--------+-----------+-----------+
unresolved black box cell | 0      | 0         | 0         |
leaf cell                 | 14     | 1         | 13        |
```

Mapping Power Rails

To report power by rails, group the cells in a design by the power or ground rails if the library contains both the power and ground pins. For multivoltage cells, such as level shifters, the library power rails need to be linked to the design power rails.

Use the `create_power_rail_mapping` command to group cells. In the following example, the power of G1, G2, and G3 is associated with the V1 design rail; B1 and B2 are tied to the V2 design rail, and R1 to the V3 design rail.

```
pwr_shell> create_power_rail_mapping V1 -cells "G1 G2 G3"
pwr_shell> create_power_rail_mapping V2 -cells "B1 B2"
pwr_shell> create_power_rail_mapping V3 -cells R1
```

The following example groups cells based on the ground rail for libraries with power and ground pins.

```
pwr_shell> create_power_rail_mapping VSS1 -cells "G1 G2 G3"
pwr_shell> create_power_rail_mapping VSS2 -cells "B1 B2"
```

The report generated by the `report_power` command contains the power consumption only for the last specified rail. The rails for which power is calculated are defined in the Current Voltage Rail heading of the report, as shown in Example 7-4.

Example 7-4 Report Example Showing Power Consumption for the Specified Rail

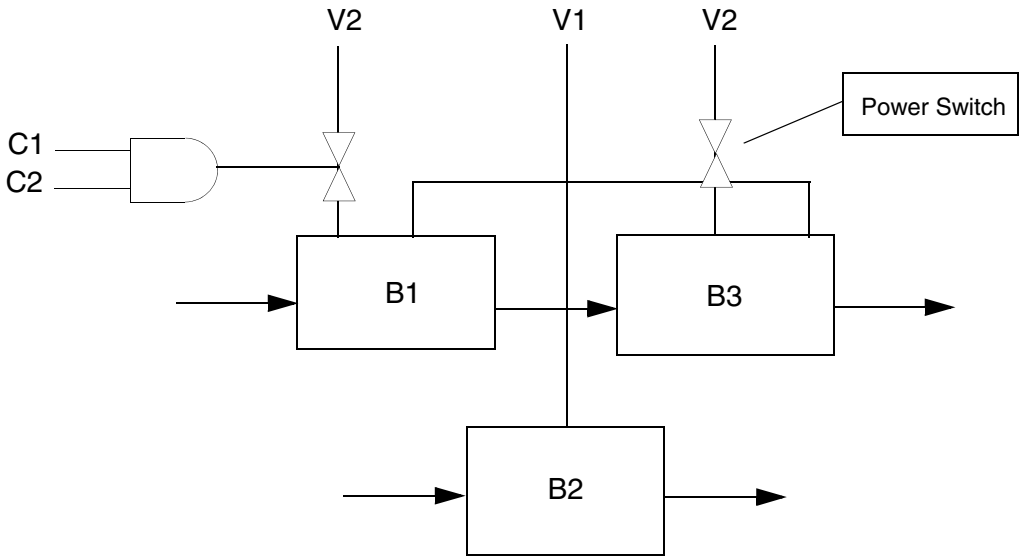
```
****************************************
Report : power
Design : d
...
****************************************
Sampling Interval: 1 ns
Current Voltage Rail: VDD2

Library(s) Used:
    pg_pin_lib (File: power_level.db)
Operating Conditions: WORST   Library: power_level
Wire Load Model Mode: top
<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  = 3.398e-05  (59%)
Net Switching Power  = 2.335e-05  (41%)
Total Dynamic Power  = 5.733e-05  (100%)
Cell Leakage Power   = 1.902e-10
Peak Power           = 2.947e-04
Peak Time            =         1
```

For multirail cells, map the power rails defined in the libraries to the physical power rails existing in the design using the `create_power_rail_mapping` command. Use the `-lib_rail_name` option to map the power rails in each library associated with the library cells. For example,

```
pwr_shell> create_power_rail_mapping V3 -lib_rail_name Vdd0 -cells B1
```

Figure 7-2 shows a design example that has multiple power rails. The V2 rail, which feeds Block 3 and portions of Block 1, can be cut off when the blocks are not used for a certain duration.

Figure 7-2 Design With Multiple Power Rails



As shown in Figure 7-2, the power rail V2 is power-gated. The second power rail mapping command shown in the following example is mandatory (see texts in bold); it defines the power-off condition and the power rail controlled by power gating. You can check the rail mapping by running the `report_power_rail_mapping` command.

Script Example

```
set_app_var link_library "merged.db"
set_app_var link_library "* lib1.db"
set_app_var link_path_per_instance [list [list {instance1} \
     {* lib2.db}] [list {instance3} {* lib3.db}]]
read_verilog $myvlog
current_design $topdesign
link
update_timing
read_vcd -strip_path tb/top vcd.dump
report_power_rail_mapping
create_power_rail_mapping V1 -lib_rail_name Vdd1 -cells "B1 B2 B3"
create_power_rail_mapping V2 -lib_rail_name Vdd2 \
```

```
     -off_condition "C1 & C2" -cells R1
report_power_rail_mapping
current_power_rail V1
report_power
```

Reporting Power Rail Mapping

For a design with multiple VDD cells, use the `report_power_rail_mapping` command to identify the power rails in the library before creating the power rail mapping between the rails in the design and libraries. After linking the design and the libraries, this command reports the libraries and their power and ground rails.

By default, the tool reports the total power consumption for all the rails in the design based on the rail voltages defined in the libraries. To report power per rail, the tool needs to know the power rails in the design as well as the rails in the library.

Specifying Power Rails for Power Analysis

By default, PrimePower analyzes or reports power for all power rails in a multirail design. To specify the power rails to be included in power analysis or in power reporting, use the `current_power_rail` command.

For example, to report power consumption per power rail, first run the `current_power_rail` command on each power rail, and then rerun the `update_power` command to analyze power per power rail.

The command behavior is dependent on the setting of the `power_enable_multi_rail_analysis` variable.

- When multirail analysis is disabled by setting the `power_enable_multi_rail_analysis` variable to `false`, PrimePower calculates power for the power rails that are specified by the `current_power_rail` command. To report power consumption per power rail, the command must be applied for each power rail, and the power reanalyzed per power rail.
- When the `power_enable_multi_rail_analysis` variable is set to `true`, PrimePower calculates power per power rail in the design, and reports power for the power rails that are specified by the `current_power_rail` command.

Before using this command, you define the design power rails by using the `create_power_rail_mapping` command. Use the `report_power_rail_mapping` command to show the defined design power rails, library power rails, and rail mapping information.

In Example 7-5, the script example reports the power consumption per rail for a design that contains two power rails, VDD1 and VDD2:

Example 7-5 A Script Example Using the current_power_rail Command

```
current_power_rail VDD1
update_timing
report_power > pp_rail_VDD1
current_power_rail VDD2
report_power > pp_rail_VDD2
```

Voltage Scaling With Power Rails

When analyzing power on the designs with multiple voltage rails, PrimePower checks the value of the `output_signal_level` attribute to calculate the switching power of the net.

For libraries with power and ground pins, PrimePower determines the signal voltages based on the `related_power_pin` and `related_ground_pin` syntax for the library pin.

For partial-swing cases, the switching power calculation takes into account the Vh/Vl voltage swing range for the net.

You can set the values of single and multirail cells and modify the default rail voltages of single-rail or multirail cells. The tool uses the specified values to calculate the correct power consumption.

Use the `set_rail_voltage` command for a particular cell to override the default voltage. This is particularly useful for what-if analysis. Another application for `set_rail_voltage` is to apply operating conditions to hierarchical blocks and then back-annotate rail voltages that are due to voltage drop on individual cells to ascertain the effects.

Power-Gating With Power Rails

Power gating is an effective leakage power saving mechanism; it saves power by dynamically disabling the power supply to certain parts of the design. Both dynamic and leakage power consumptions take place inside the block. If a block is not used for a certain duration, its power supply can be cut off for saving leakage power.

To analyze the power of designs with power gating, use the `create_power_rail_mapping -off_condition` command to specify the conditions when the power rail is disabled. For example,

```
pwr_shell> create_power_rail_mapping V2 -cells "B1 B2" \
          -off_condition "C1 & C2"
```

When you specify VCD activity, PrimePower monitors the activity over time, and accurately analyzes power only during the time windows when the power-off condition is `false`. When the condition is `true`, the power consumption is zero.

If you provide SAIF or user-defined activity, PrimePower uses the static probability of the power-off condition to scale the leakage power only; the tool assumes that the activity captures the toggles when the power is on. If the activity is obtained from simulation, which assumes that the power is always on, set the `power_scale_dynamic_power_at_power_off` variable to `true`. Both dynamic power and leakage power are scaled.

Reporting Power and Power Attributes Per Rail

When you set the `power_enable_multi_rail_analysis` variable to `true`, PrimePower performs power analysis for all the supply nets or power rails in the design. By default, the `report_power` command reports the total power of all the supply nets or rails.

To selectively report power for a supply net or rail, run the `report_power -rails` and `get_power_per_rail` commands on the specified rail, without recalculating the power values. Run the `report_attribute` and `get_attribute` commands to report power attributes for the specified rail.

Multirail power analysis can be performed for all flows and all four types of reporting: summary, cell-based, net-based, and hierarchy-based reporting.

You can extract the total power attributes for each rail, generate custom per-rail reports for hierarchical blocks, and include the power consumption of rails.

Reporting Power Using report_power -rails

Run the `report_power -rails` command to report the power values for the specified rails.

Example 7-6 Report Generated by the report_power -rails Command

```
****************************************
Report : Averaged Power
...
****************************************
Current Power Rail: all
Power Report For Rails: VDD1 VDD2
Attributes
 ----------
 i  -  Including register clock pin internal power
 u  -  User-defined power group
               Internal  Switching Leakage  Total
Power Group    Power     Power     Power    Power      (    %)      Attrs
------------------------------------------------------------------------
io_pad         0.0000    0.0000    0.0000    0.0000    ( 0.00%)
memory         0.0000    0.0000    0.0000    0.0000    ( 0.00%)
```

```
black_box      0.0000    0.0000    0.0000    0.0000    ( 0.00%)
clock_network  0.0000    0.0000    0.0000    0.0000    ( 0.00%)    i
register       0.0000    0.0000    0.0000    0.0000    ( 0.00%)
combinational  1.842e-05 4.886e-07 9.315e-11 1.891e-05 (100.00%)
sequential     0.0000    0.0000    0.0000    0.0000    ( 0.00%)

   Net Switching Power  = 4.886e-07   ( 2.58%)
   Cell Internal Power  = 1.842e-05   (97.42%)
   Cell Leakage Power   = 9.315e-11   ( 0.00%)
                          ---------
Total Power             = 1.891e-05  (100.00%)
```

Reporting Power Using get_power_per_rail

Use the `get_power_per_rail` command to report power on the given rails of the design for a given list of hierarchical cells. You can specify the name of the rail to report by specifying `leaf`, `top`, or `through_switch` with the `-return_supply_level` option.

Example 7-7 Reporting Power Per Rail Using the get_power_per_rail Command

```
pwr_shell> get_power_per_rail -rails vddcx vddmx \
            -return_supply_level leaf
{vddcx inst_blck1 0.0} {vddcx inst_blck2 0.0} {vddcx not_included 0.500}
{vddmx inst_blck1 0.0} {vddmx inst_blck2 0.0} {vddmx not_included 0.6}
{unconnected inst_blck1 0.0} {unconnected inst_blck2 0.0}
{unconnected not_included 0.0} {other u_inslt_blck1 0.3 }
{other inst_blck2 0.6} {other not_included 0.0}
```

Reporting Power Attributes

Use the `report_attribute` and `get_attribute` commands to list power attributes for the specified rail. The power attributes available for each power rail are: `internal_power`, `switching_power`, `leakage_power`, `dynamic_power`, and `total_power`.

The attributes for each rail of each cell can be used in conjunction with the PrimeTime `pg_pin_info` cell attribute, which returns a Tcl collection of the PG pins and their associated rails to generate a custom report. The report includes the power, the supply net, and the PG pin names for all the cells in the design.

Note:

The attributes correspond to the value for the specified rail.

The `peak_power` and `peak_time` attributes report the total power for all the rails unless you set the `power_enable_multi_rail_analysis` variable to `false`.

When Multirail Analysis is Disabled

When the `power_enable_multi_rail_analysis` variable is set to `false`, use the `set_current_power_net` or `current_power_rail` command to specify the power rails to include in the power report. The tool analyzes and reports only the power consumption of the current power net.

Example 7-8 Reporting Power Attributes for Specific Rails

```
pwr_shell> set_app_var power_enable_multi_rail_analysis true
pwr_shell> update_power
pwr_shell> set_current_power_net vdd1
pwr_shell> get_attribute [get_cell U1] internal_power
 1.1e-07
pwr_shell> set_current_power_net vdd2
pwr_shell> get_attribute [get_cell U1] internal_power
 2.2e-07
```

Macro Cells With the Built-In Multiplexer

Macro cells do not contain internal logic information. In a UPF design implementation using macro cells, the reference macro library cell should be modeled using appropriate Liberty attributes that describe the UPF intent. For a multirail macro with specific power state requirements, you should also provide a UPF intent that is consistent with the intent described in the macro cell library model.

Use the built-in power multiplexer to model the input PG pin rails for fine-grained switch cells with the `pg_function` attribute specified as a Boolean expression. This allows you to change the power based on the enable pin logic, and report correct power numbers for various power components per rail.

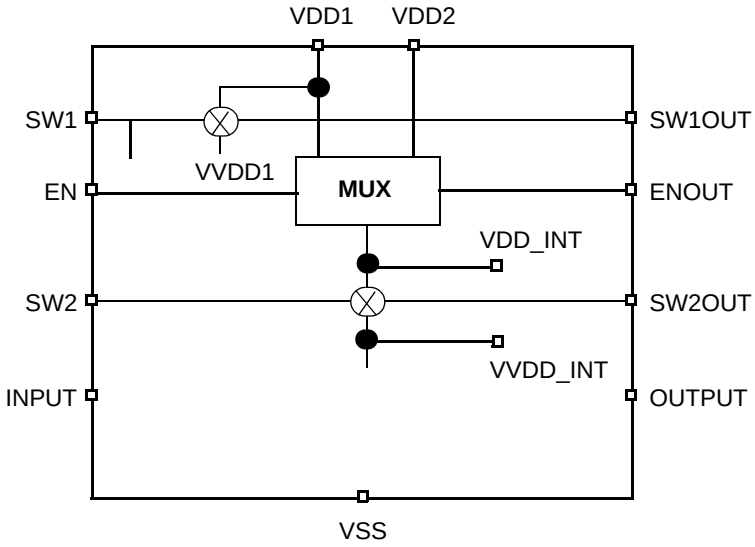
In the following Liberty model, the internal PG pin, VVDD_INT, does not have the `switch_function` attribute. The Boolean expression of the `VVDD_int pg_function` attribute includes the signal pin, SIG_EN.

```
cell(MACRO) {
  is_macro_cell : true;
  switch_cell_type : fine_grain;
…..
….
..
pg_pin(VVDD_INT) {
  voltage_name :VDD1;
  pg_type : internal_power;
  direction : internal;
  switch_function : "SW2";
  pg_function : "(VDD1 * !EN + VDD2 * EN)";
   }
….
  pin(SW2) {
   direction : input;
   switch_pin : true;
   related_power_pin : VDD1;
   related_ground_pin : VSS;
```

The power to VVDD_INT is dependent on the **logic value** of the pin EN

Figure 7-3 shows a macro cell with fine-grained internal power switches and a built-in power multiplexer with input PG pin rails, VDD1 and VDD2. In this example, VDD1 and VDD2 are the primary input power pins, VDD_INT and VVDD_INT are internal power pins. For the VDD1 power pin, the related pins are SW1, SW1OUT, SW2, SW2OUT, EN, and ENOUT. For the VVDD1 power pin, the related pins are INPUT and OUTPUT.

Figure 7-3 A Macro Cell With Built-In Multiplexer



Multivoltage Power Analysis Using UPF

In PrimePower, use the UPF file to specify your power intent for the analysis of your multivoltage design. The UPF file provides the ability to specify the power intent early in the design process, and is supported throughout the design flow.

When you specify the power intent for UPF designs, by default PrimePower converts and updates the library power and ground (PG) pins by using the specifications provided in a Tcl file.

You can specify the UPF commands directly at the tool prompt or by using the `load_upf` command if they are contained in a separate UPF file. Table 7-1 lists the UPF commands for running multivoltage power analysis.

Table 7-1 UPF Commands for Multivoltage Power Analysis

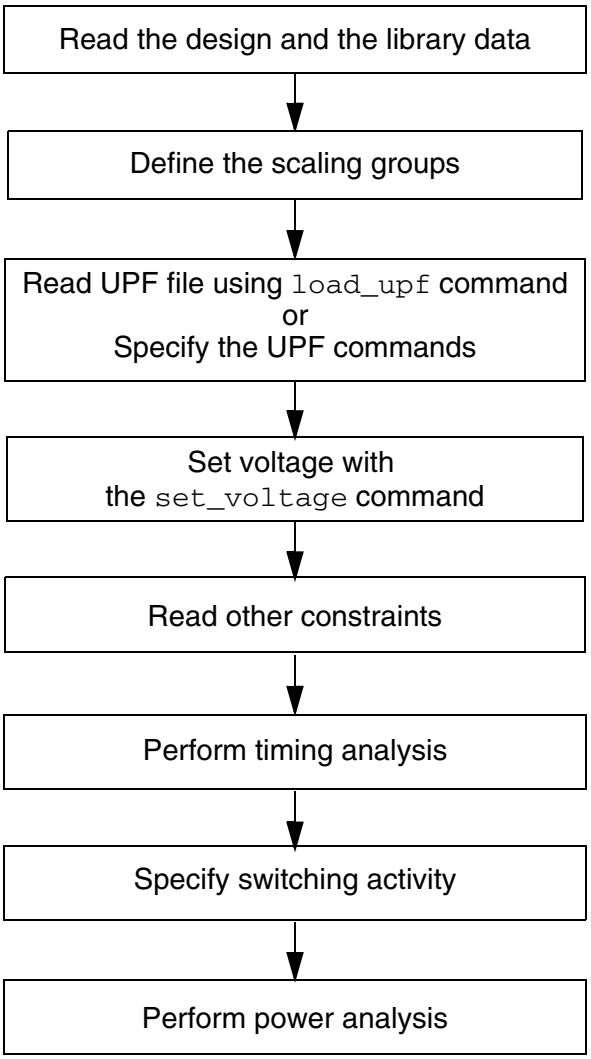
Commands	Descriptions
`create_power_domain`	Creates power domains
`create_supply_port`	Creates supply ports

Table 7-1 UPF Commands for Multivoltage Power Analysis (Continued)

Commands	Descriptions
`create_supply_net`	Creates supply nets
`connect_supply_net`	Creates power connections
`set_domain_supply_net`	Defines power connections

Figure 7-4 shows the steps for running multivoltage power analysis with a UPF file.

Figure 7-4 Multivoltage Power Analysis Flow Using UPF



For more information about low-power flow and various Synopsys tools that support UPF, see the *Synopsys Multivoltage Flow User Guide*.

Power Domains

Multivoltage designs contain design partitions that have specific power behavior compared to the rest of the design. A power domain is a basic concept in the Synopsys low-power flow, and it drives many important low-power features across the flow. A power domain describes a design partition, bounded within logical hierarchies, which has a specific power behavior with respect to the rest of the design.

Note:

A power domain is a logic construct, not a netlist object.

The logical hierarchy where the power domain is created is called the scope of the power domain. Use the `set_scope` command to specify the scope or level of the hierarchy of the power domain.

Use the `create_power_domain` command to create a power domain. Use the `-elements` option to specify the list of hierarchical cells, input and output pad cells, and macro cells, that are added as extent of the power domain. Use the `-include_scope` option to specify that all the elements in the current scope share the primary supply of the power domain, but are not necessarily a part of the power domain. Use the `-scope` option to specify the logical hierarchy or the scope where the power domain is to be defined.

Use the `create_supply_net` and `create_supply_port` commands to create ports and nets respectively, for the specified power domain and `connect_supply_net` command to connect the supply net to the specified supply ports and pins.

Every power domain must have a primary supply power net and a primary ground net. Use the `set_domain_supply_net` command to define the primary supply and ground net for an existing power domain.

Isolation Cells

For a design with power switching, an isolation cell is required where each logic signal crosses from a power domain that can be powered down to another power domain that is not powered down. The isolation cell functions as a buffer when the input and output of the isolation cell are both powered up. When the input of the cell is powered down, the isolation cell provides a constant output signal. The enable input controls the operating mode of the isolation cell.

Precedence Rules

To identify the supply sets or nets and the corresponding voltages for isolation, PrimePower uses the following precedence rules:

- Isolation supply nets

  The source or sink property of a net for isolation corresponds to all the net segments connected, including the nets that connect to the level shifters and isolation cells. The dangling isolation and level-shifter cells are treated as source or sink. The tool stops the tracing at the retention cells, standard cells, black box cells and boundary ports. It chooses the supply set or nets for isolation based on the following criteria, in the following order of precedence:

  1. Supply set or nets that have the `iso_source` or `iso_sink` attribute set by the `set_port_attributes` command.
  2. Supply set or nets that are related supply nets, defined by the `set_related_supply_net` command.
  3. Top-level supply set or nets at the boundary port.

  Note:
  When the supply set or nets are found, no further selection is made.

- Supply net voltages

  To determine the supply voltage of each PG pin for the cells in the design, PrimePower uses the following order of precedence, from highest to lowest. The UPF port voltage has higher precedence over the non-UPF port voltage.

  - The `-driver_supply` or `-receiver_supply` option with the `-ports` option for the driver supply on the input ports and the receiver supply on the output ports.
  - The `-driver_supply` or `-receiver_supply` option with the `-elements` option for the driver supply on the input ports and the receiver supply on the output ports.
  - The `set_related_supply_net` command.
  - The primary supply of the power domain of the top-level design.
  - The `set_voltage` command on a PG pin. This requires a PrimeTime SI license.
  - The `set_voltage` command on a supply net.
  - The `set_operating_conditions` command specified at the design level.
  - The default supply voltage in the library cell definition specified by the `voltage_map` attribute in the Liberty syntax.

Commands to Define Isolation Strategy

Use the `set_isolation`, `set_isolation_control`, `set_port_attributes`, and `set_design_attributes` UPF commands to specify isolation strategies in the PrimePower UPF flow.

set_isolation

Use the `set_isolation` command to define the UPF isolation strategy for the specified power domains. An isolation strategy includes specification of the enable signal net, the clamp value, and the location; inputs, outputs, or both. This command creates an explicit connection between the isolation power and ground nets and the power and ground pins of the isolation cell.

Every isolation strategy that you define using the `set_isolation` command must have a corresponding `set_isolation_control` command, unless you have specified the `-no_isolation` option.

set_isolation_control

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

set_port_attributes

Use the `set_port_attributes` command to specify information that is relevant to the ports on the interface of the power domains. The information is used to verify the isolation and guard requirements for the port.

set_design_attributes

Use the `set_design_attributes` command to specify a collection of cells where the attributes are set for the source or sink of the power domains, when you use with the `set_isolation` command.

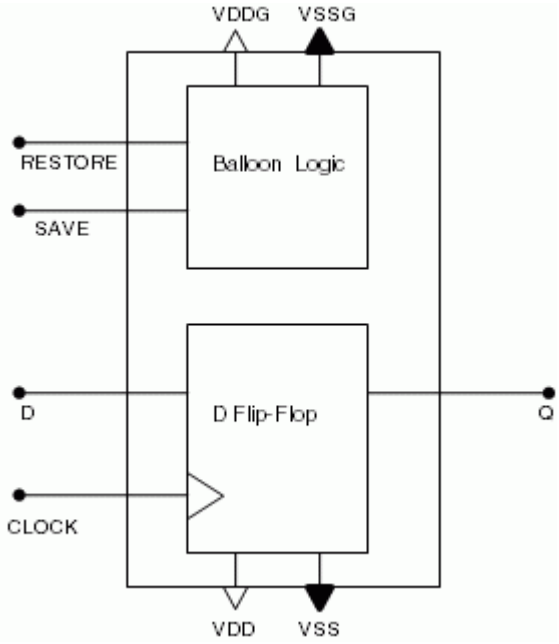
The `-elements` option specifies a collection of cells where the attributes must be set. Use the `-attribute` option to specify the *name* and *value* of the attributes to set on the cells when you specify the `-elements` option. When you do not specify `-elements` option, the `-attribute` option is set on the current top-level design. The *name* is the one set by `external_supply_map` and the *value* is the reference name of a supply set or a supply-net pair.

Retention Registers

In multivoltage designs, when a power domain is shut down and later powered up, it is often necessary for the power domain to resume operation based on its last good state. Special cells called retention registers can store the state during the shutdown.

As shown in Figure 7-5, a retention register has two control signals, save and restore, to save and restore the data. Retention cells occupy more area than regular flip-flops. These cells continue to consume power when the power domain is powered down.

Figure 7-5 Two Pin Retention Register



Retention Commands

Use the retention commands to specify the strategy for inserting retention cells inside switched (power-down) domains during synthesis. When specified, the tool verifies the syntax and power connections.

set_retention

Use the `set_retention` command to specify the registers in the domain implemented as retention registers and identifies the save and restore signals for the retention function. Specify one of the `-retention_power_net` and `-retention_ground_net` options at the least. Specifying both options indicates the supply nets are retention power and ground nets. If you specify only the retention power supply net, the tool uses the primary ground net as

the retention ground supply. If you specify only the retention ground net, the tool uses the primary supply net as the retention power supply. The tool connects the retention power and ground nets to the implicit save and restore processes and shadow register.

The following strategies have a decreasing level of precedence, irrespective of the order in which they are executed:

```
set_retention -domain -elements
set_retention -domain
```

Define a retention strategy using the `set_retention` command with a corresponding `set_retention_control` command.

set_retention_control

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

Power Gating in UPF Mode

In the UPF mode, PrimePower supports power-gating-aware power calculation to measure the reduction in the leakage power. When cells in the design are turned off, power gates disconnect the cells from the power supply resulting in reduced or no leakage power consumption. A power switch transforms a supply net from an input port to an output port based on the state of the control port. The state of the control port is defined by a Boolean expression.

Use the `create_power_switch` command to define a power switch in a power domain. The power switch is created within the scope of the specified power domain. This command specifies the power-down control so that when cells are turned off, they are disconnected from the power supply. When the Boolean expression associated with the `on_state` option evaluates to `true`, the switch is enabled and the value at the input supply port is propagated to the output supply port.

The power pin of a cell is switched off only when the following conditions are satisfied:

- The power domain that the cell belongs to has power switches specified.
- The power supply net of the cell is connected to the output supply net of one of the power switches.
- The value of the Boolean expression, associated with the `on_state` of the power switch, is FALSE.

PrimePower supports single input, single output, multiple control switches. Use the `-domain` option to specify the power domain where the power switch is to be created. Use the

`-output_supply_port` option to specify the output port of the power switch and the supply net to connect this output port.

Use the `-control_port` option to specify the control port of the power switch and the net where this control port is to be connected. Because multiple control ports are supported for the power switch, use the option to specify multiple pairs of control ports and the associated nets. Use the `-on_state` option to specify the state, the associated input port and the Boolean function.

Note:

PrimePower does not support additional options of the `create_power_switch` command, such as `-ack_port`, `-on_partial_state`, `-ack_delay`, `-off_state`, and `-error_state`, and ignores them without generating any warning message.

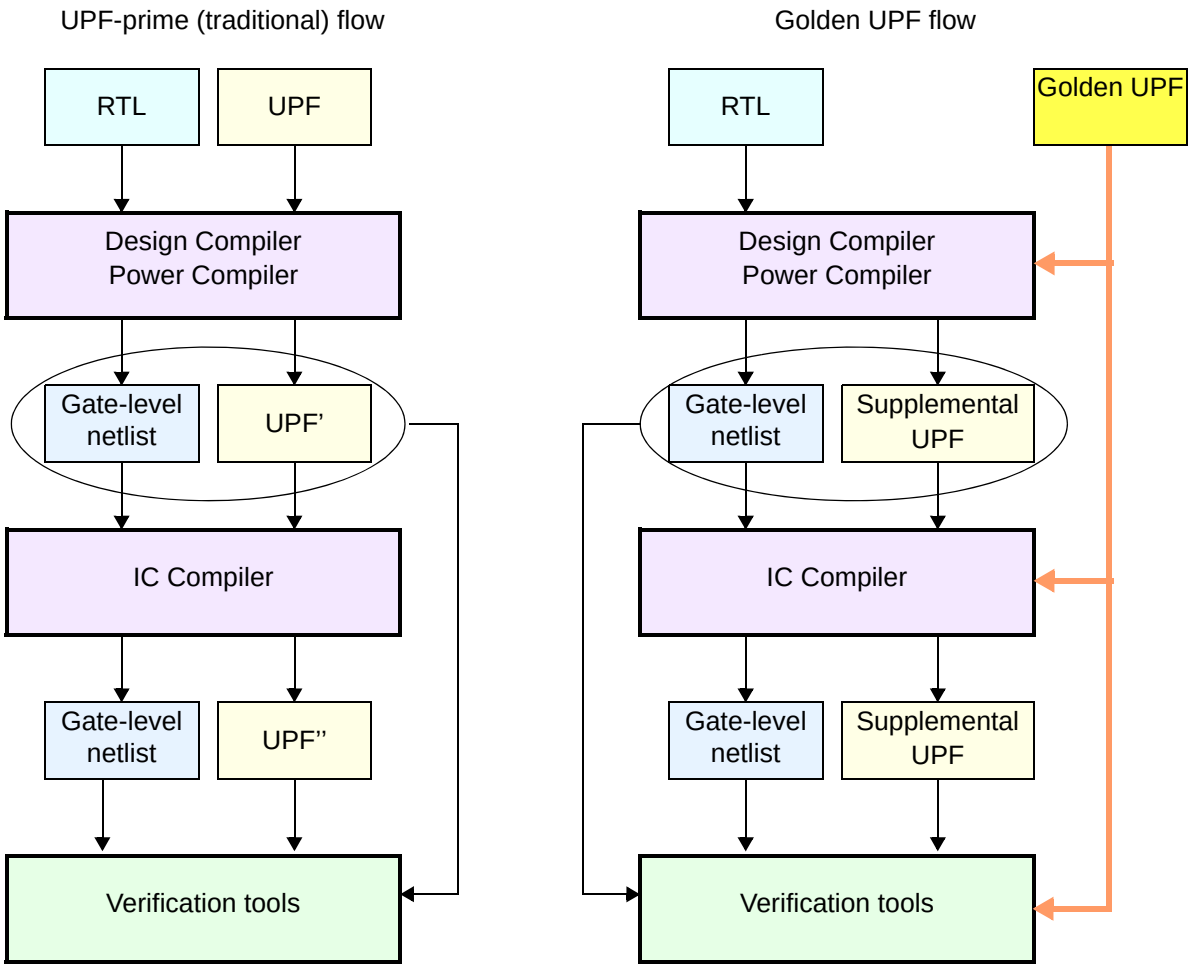
The tool does not support other UPF commands related to power switches, such as `map_power_switch` and `set_power_switch`, and ignores by generating a warning message.

Use the `report_power_switch` command to report all the power switches defined in the design. This command reports details such as the names of the power switches, the power domains where the switches are defined, input, output and control port details. This command is not an IEEE 1801 compliant UPF standard command.

Golden UPF Flow

The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Design Compiler and IC Compiler tools. Figure 7-6 compares the traditional UPF flow with the golden UPF flow.

Figure 7-6 UPF-Prime (Traditional) and Golden UPF Flows



The golden UPF flow maintains and uses the same, original “golden” UPF file throughout the flow. The Design Compiler and IC Compiler tools write power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF file.

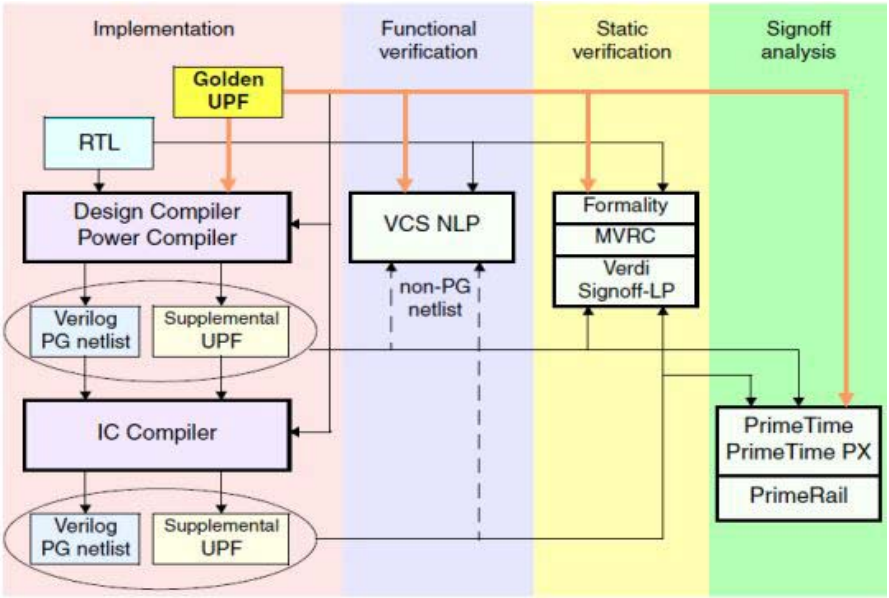
The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.

- You can optionally use the Verilog netlist to store all PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

  Figure 7-7 shows how the golden UPF file is used by the PrimePower and other Galaxy verification tools throughout the flow, in combination with a supplemental UPF file generated by the Design Compiler or IC Compiler II tool.

Figure 7-7 Verification Tools in the Golden UPF Flow



To use the golden UPF flow, you must enable it by setting a variable:

```
pwr_shell> set_app_var enable_golden_upf true
```

For more information about using the golden UPF flow, see SolvNet article 1412864, "Golden UPF Flow Application Note."

Switch Cells for Power Analysis in UPF Mode

Figure 7-8 shows a power switch for a block.

The block Block1 has

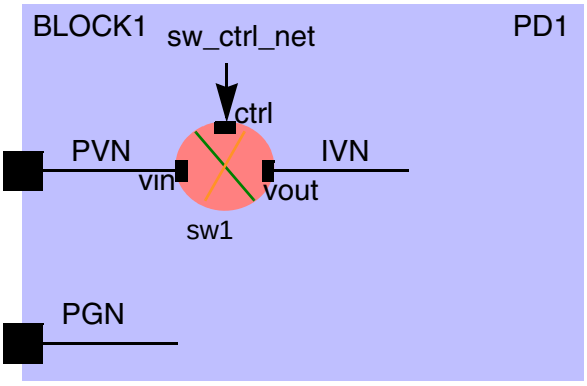
- Power domain: PD1
- Two supply ports: PVN and PGN
- Three supply nets: PVN, PGN, and IVN
- A control net: sw_ctrl_net; and
- A power switch: sw1

The power switch sw1 has

- One input supply port: vin
- One output supply port: vout
- One control port: ctrl

All the cells in Block1 are powered down by the switch, sw1, based on the Boolean function that you specify with the `create_power_switch -on_state` command.

Figure 7-8 Power Switch



Example 7-9 is a typical script used for power analysis.

Example 7-9 Script Example for Power Analysis

```
# Read the target libraries and input designs
set_app_var power_enable_analysis true
set_app_var power_analysis_mode time_based

set_app_var link_library lib.db
read_verilog design.v
current_design design_top
link

# Read the UPF file containing following UPF commands
# load_upf design_upf.tcl

# Alternately specify the UPF commands
create_power_domain PD1
create_supply_net PVN -domain PD1
create_supply_port PVN -domain PD1
connect_supply_net PVN -ports PVN

create_supply_net IVN -domain PD1

create_power_switch sw1 -input_supply_port {vin PVN} -output_supply_port
{vout IVN} -control_port {ctrl sw_ctrl_net} -on_state {state1 vin
{sw_ctrl_net}}
create_supply_net PGN -domain PD1
create_supply_port PGN -domain PD1
connect_supply_net PGN -ports PGN

set_domain_supply_net PD1 -primary_power_net IVN -primary_ground_net PGN
set_voltage on_supply_net
set_temperature on_blocks

# Specify other constraints
read_sdc design.sdc

# Read parasitics
read_parasitics design.spef

# Perform timing analysis
update_timing

# Specify switching activities
# The input file can be in VCD or SAIF formats.
read_vcd file_name

# Do the power analysis, either averaged or timed-based
update_power

# Get the report after the analysis
report_power
```

Support of UPF Footer Switches

PrimePower supports UPF power switches on both the power supply net and the ground net. Power switches specified on the ground net are also known as UPF footer switches. This support is available in the averaged and time-based power analysis modes. The tool supports the single and multiple ground cells controlled by the UPF footer switches in the following ways:

- Single-ground cells

  When the ground net is switched off, power consumption of the entire cell is shut off.

- Multiple ground cells

  When all the grounds are switched off, the power consumption of the entire cell is shut off. If any of the grounds is not switched off, the cell is not shut off and it continues to consume power.

Note:

To use this feature, the target library must comply with the PG pin Liberty library syntax. Both the PrimeTime and PrimePower tools support the conversion and update of the PG pins of the library. They also support converting non-PG pin libraries to PG pin libraries.

For more information, see the *PrimeTime User Guide*.

Voltage Scaling in UPF Mode

In multivoltage designs, voltage scaling improves the accuracy of the power calculation. Voltage scaling is supported for both internal power and leakage power calculation in the NLPM power library format. The library cells are characterized at different operating conditions and present in separate libraries. Because of the support of voltage and temperature scaling, during power analysis, the data can be interpolated from these separate libraries for the preferred operating condition.

Use the `set_voltage` command to define the operating voltage on power and ground pins and the power and ground nets. By defining the operating voltage, the parts of the design powered by these nets and power and ground pins are optimized for the specified voltage. The effective voltage used by the tool for power calculation is the difference between VDD and VSS.

Because the `set_voltage` command is available in the UPF mode to define the operating voltage, the `set_rail_voltage` command is not supported in PrimePower in the UPF mode.

Note:

The operating voltage that you define on the power nets using the `set_voltage` command is not propagated across the power switch. To have similar operating voltage

for nets across the power switch, use the `set_voltage` command on the supply net on both sides of the power switch.

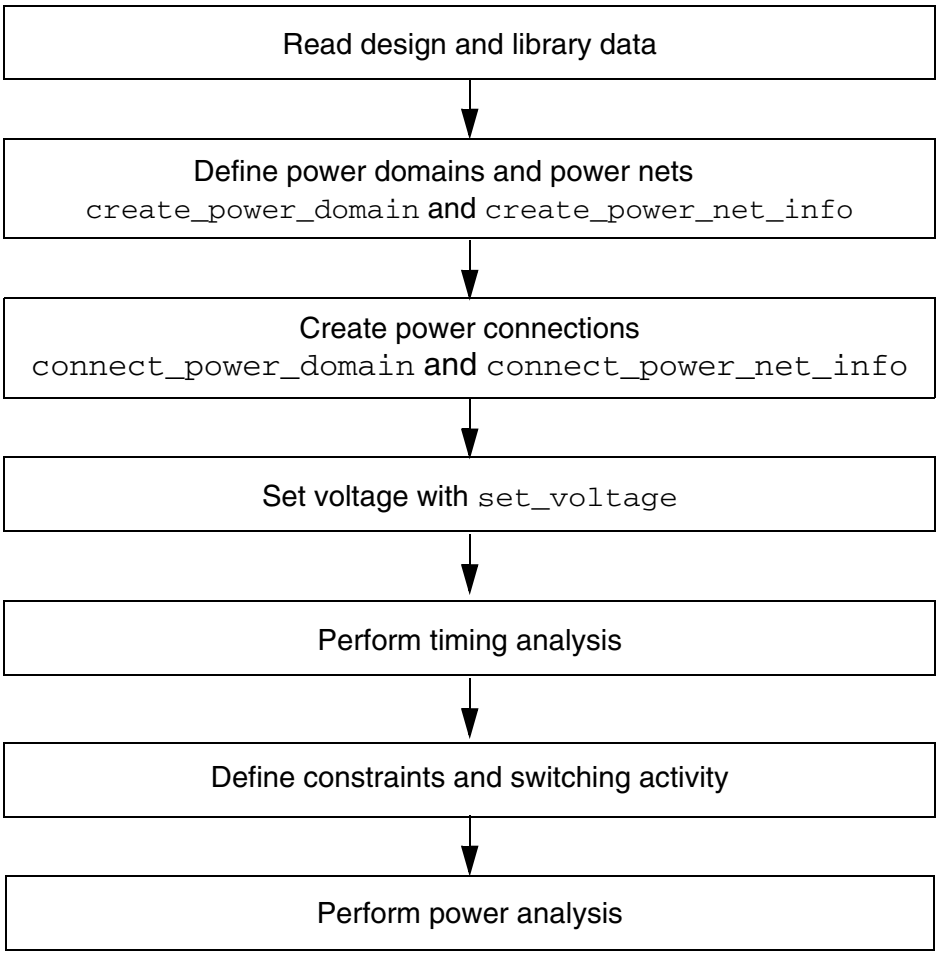
Non-UPF Multivoltage Power Analysis Using Power Domains

To invoke the PrimePower tool in the non-UPF mode, set the `power_domains_compatibility` variable to `true`. This variable reverts back to power domains and disables UPF mode for power analysis. The default is `false`. To get a list of commands supported in this mode, run the following command:

```
pwr_shell> help "power domains"
```

Figure 7-9 shows the steps for running multivoltage power analysis with power domains.

Figure 7-9 Multivoltage Power Analysis Flow Using Power Domains



The following commands support multivoltage power analysis using power domains:

- `create_power_domain`

  Specifies the name of the power domain and lists the hierarchical cells associated with the domain. The power domain applies to the top level if you do not specify a list. There can be only one top-level domain.

- `connect_power_net_info`

  Defines power net connections for a specific power pin of a leaf cell. The pin-level connections override the domain-level connections made with the `connect_power_domain` command.

- `connect_power_domain`

  Creates logical power connections for a specified power domain. You can specify the primary, backup, and internal power and ground nets for a specified power domain. All cells in the power domain inherit the specified power connections. Linking the design again causes the tool to discard power domain information.

Use the `create_power_net_info` command to specify the name of a power supply net or ground net in the design. You can also specify the voltage and the condition to disable a power supply net.

Voltage Scaling Using Power Domains

Power domains can contain multiple power nets, where the type and default voltage values are defined using the `create_power_net_info` command. The voltage of the power nets can be overwritten using the `set_voltage` command. PrimePower scales the power consumption per power net, based on the specified voltage.

Use the `set_voltage` command to define the operating voltage on the power nets created using the `create_power_net_info` command. You can specify a single voltage or minimum and maximum voltages for the power nets. If you do not specify this command, the tool uses the existing operating condition settings. To report the operating voltages of power nets, use the `report_power_net_info` command.

Example 7-10 shows how to perform multivoltage power analysis using power domains.

Example 7-10 Script to Perform Multivoltage Analysis Using Domains

```
# Read libraries, design, enable power analysis
# and link design
set_app_var power_enable_analysis true
set_app_var link_library slow_pgpin.db
read_verilog power_pins.v
link
# Create back-up power nets
```

```
create_power_net_info vdd_backup -power
create_power_net_info vss_backup -gnd
# Create domain power nets
create_power_net_info t_vdd -power -switchable \
 -nominal_voltages {1.2} -voltage_ranges {1.1 1.3}
create_power_net_info a_vdd -power
create_power_net_info b_vdd -power
# Create domain ground nets
create_power_net_info t_vss -gnd
create_power_net_info a_vss -gnd
create_power_net_info b_vss -gnd
# Create internal power nets
create_power_net_info int_vdd_1 -power \
      -nominal_voltages {1.2} -voltage_ranges {1.1 1.3} \
      -switchable
create_power_net_info int_vdd_2 -power \
      -nominal_voltages {1.25} -voltage_ranges {1.1 1.3}
create_power_net_info int_vdd_3 -power \
      -nominal_voltages {1.2} -voltage_ranges {1.1 1.3}
create_power_net_info int_vdd_4 -power
# Create power domains
create_power_domain t
# Connect rails to power domains
connect_power_domain t -primary_power_net t_vdd \
      -primary_ground_net t_vss
connect_power_domain a -primary_power_net a_vdd \
      -primary_ground_net a_vss \
      -backup_power_net vdd_backup \
      -backup_ground_net vss_backup
connect_power_domain b -primary_power_net b_vdd \
      -primary_ground_net b_vss
# Set voltages of power nets
set_voltage 1.15 -object_list{t_vdd a_vdd b_vdd}
# Read SDC and other timing or power assertions
set_input_transition 0.0395 [all_inputs]
set_load 1.0 [all outputs]
# Perform timing analysis
update_timing
# Read switching activity
set_switching_activity...
set_switching_activity...
...
report_power
```

Querying Power on PG Pins

You can extract the power for each PG pin by using the `get_power_per_pgpin` command and use the output of this command to generate custom reports. The syntax is:

```
get_power_per_pgpin -power_type [switching | leakage | internal | total]
```

By default, the command reports the total power of a hierarchical net connected to the PG pins of a cell. In Example 7-11, the PG pin vdd of cella is connected to the supply net vdd1, and the PG pin vdds is connected to supply net vdd2.

Example 7-11 Reporting the Power for Each PG Pin

```
pwr_shell> get_power_per_pgpin [get_cells *cell*]
{cella {vdd 'vdd1' 1.056e-06} {vdds 'vdd2' 6.424e-09}}
{cellb {vdd 'vdd1' 1.865e-06} {vdds 'vdd3' 1.347e-08}}
{cellc {vdd 'vdd1' 1.374e-06} {vdds 'vdd2' 1.237e-08}}
```

You can extract the power for each PG pin in the following multivoltage analysis flows:

- UPF

  You can extract the power for each PG pin by querying the power for each power net and associating it with the PG pin, unless the power net is connected to multiple PG pins. Using the `get_power_per_pgpin` command, you can extract the power for each PG pin even if multiple cell PG pins are connected to the same power net.

- Rail-mapping

  In rail-mapping mode, the `get_power_per_pgpin` command enables power reporting for each PG pin without the additional scripting to identify the supply nets connected to each PG pin. Because the `power_pin_info` attributes do not include the connected supply net per PG pin, mapping the supply net to the PG pin requiring parsing of the rail-mapping scripts is not necessary.

The output of the `get_power_per_pgpin` command lists all PG pin types except for generic ground pins (primary ground, backup ground, internal ground) whose voltage and associated power is always 0. The command reports the following PG pin types:

- PG power pins: primary power, backup power, and internal power
- Substrate bias pins: pwell, nwell, deeppwell, and deepnwell

8

Clock Network Power Analysis

In PrimePower, you can classify cells into different power groups using the `create_power_group` command. Each power group is defined in a unique name. Cells in each group need not be mutually exclusive.

This chapter describes how to propagate activity through clock network elements and calculate power consumption of the clock network and registers in the averaged mode.

Note:
The clock network power analysis feature is available only in the averaged mode.

This chapter contains the following topics:

Propagating Activities Through Clock Networks

By default, PrimePower uses zero-delay simulation for data-path activity propagation for clock networks in the averaged power mode. To propagate toggle rates through a clock network, enable the `through_mode` propagation by running the `set_power_analysis_options` command with the `-through_mode` option. To derate the toggle rate at every cell in the clock network, use the `set_activity_derate` command.

Both `through_mode` propagation and derating of toggle rates are supported only in averaged power analysis. Derating of toggle rates is supported only when the `through_mode` propagation is enabled.

Through-Mode Propagation

Figure 8-1 shows how PrimePower propagates activities through a clock network with the `-through_mode` option.

Figure 8-1 An Example of through_mode Activity Propagation

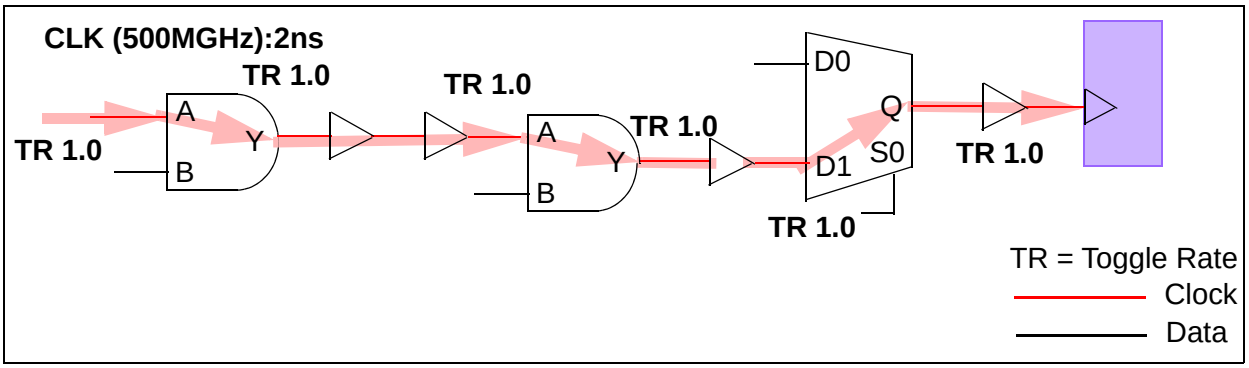


Table 8-1 lists the tasks available in the `through_mode` activity propagation.

Table 8-1 Commands Used in the through_mode Propagation

Tasks	Commands
Enable the `through_mode` propagation	`set_power_analysis_options -through_mode`
Check if the `through_mode` propagation has been set on a cell	`report_power_analysis_options`
Disable the `through_mode` propagation on a cell	1. Rerun the `set_power_analysis_options` command without the `-through_mode` option. 2. Run the `reset_switching_activity` command to remove the switching activity annotation from the individual design objects.

The `through_mode` propagation supports all combinational and clock-gating cells in a clock network, as well as two-input MUX cells. If multiple clock signals arrive at the input pin of a MUX cell, the fastest clock signal (that is, the largest toggle rate value) is propagated to the next stage.

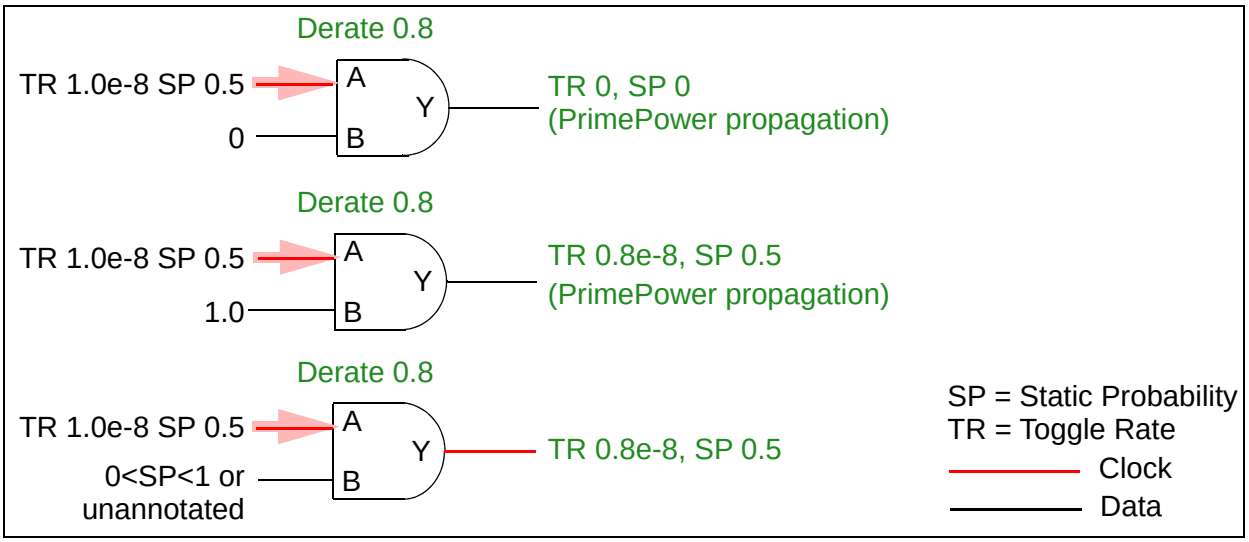
Through-Mode Propagation With Toggle Rate Derating

When propagating activities through a clock network, the output pin toggle rate is dependent on the value of static probability of the input pin and the derating factor of the cell instance. The default derating factor for all cells is 1.0.

Before propagating activities from inputs to outputs, PrimePower first checks the static probability of the input pin. When static probability (SP) is either unannotated or 1 (SP=1) or between 0 and 1 (0<SP<1), the tool derates the input pin toggle rate in calculation of the output pin toggle rate. Currently, no derating factor is applied to buffers and inverters in the design.

Figure 8-2 shows how PrimePower calculates toggle rates on the AND gate cells with the consideration of the static probability when propagating activity through a clock network.

Figure 8-2 Examples of Propagating Activities Through AND Gates



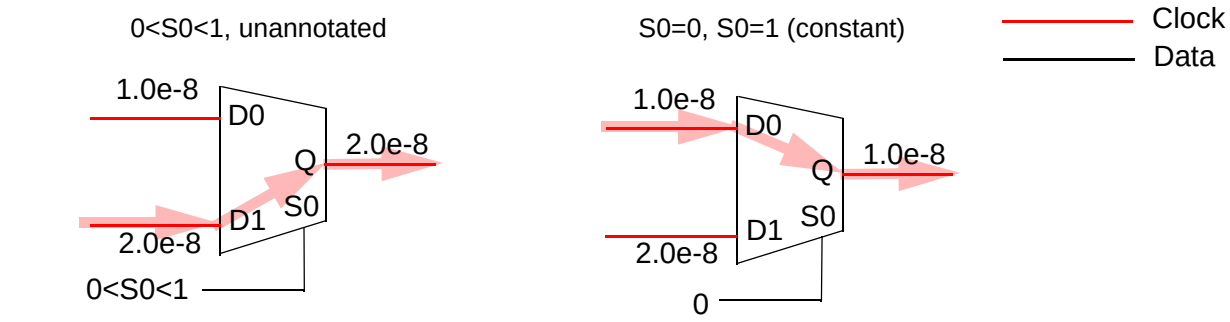
Show in Figure 8-3, for a MUX cell, when multiple clock signals are present at the data inputs and SP is between 0 and 1 (0<SP<1), the fastest clock signal at the MUX input is propagated to the output. When SP is 0 or 1, propagation is performed based on the function of the cell; that is, select D0 input when SP=0 or select D1 input when SP=1.

When a single clock signal is present at the input of a MUX cell, propagation of clock signal is the same as for other combinational cells as described before. When SP is 0 or 1, PrimePower propagates activity based on the function of cell (i.e., select D0/D1 input pin of

MUX). When SP is unannotated or fractional (0<SP<1), the clock signal is propagated to the output pin.

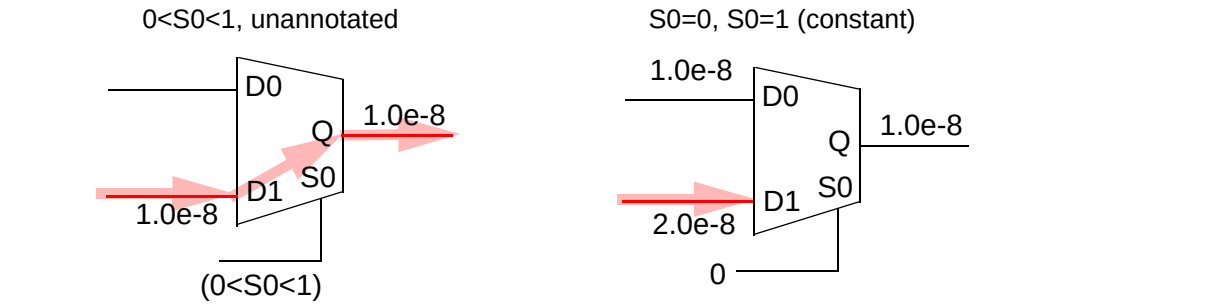
Figure 8-4 includes examples of several combination cells, including AND gate cells, ICG cells, and MUX cells.

Figure 8-3 Examples of Propagating Activities Through MUX Cells



- Multiple clock inputs, the larger toggle (the worst) has to be propagated

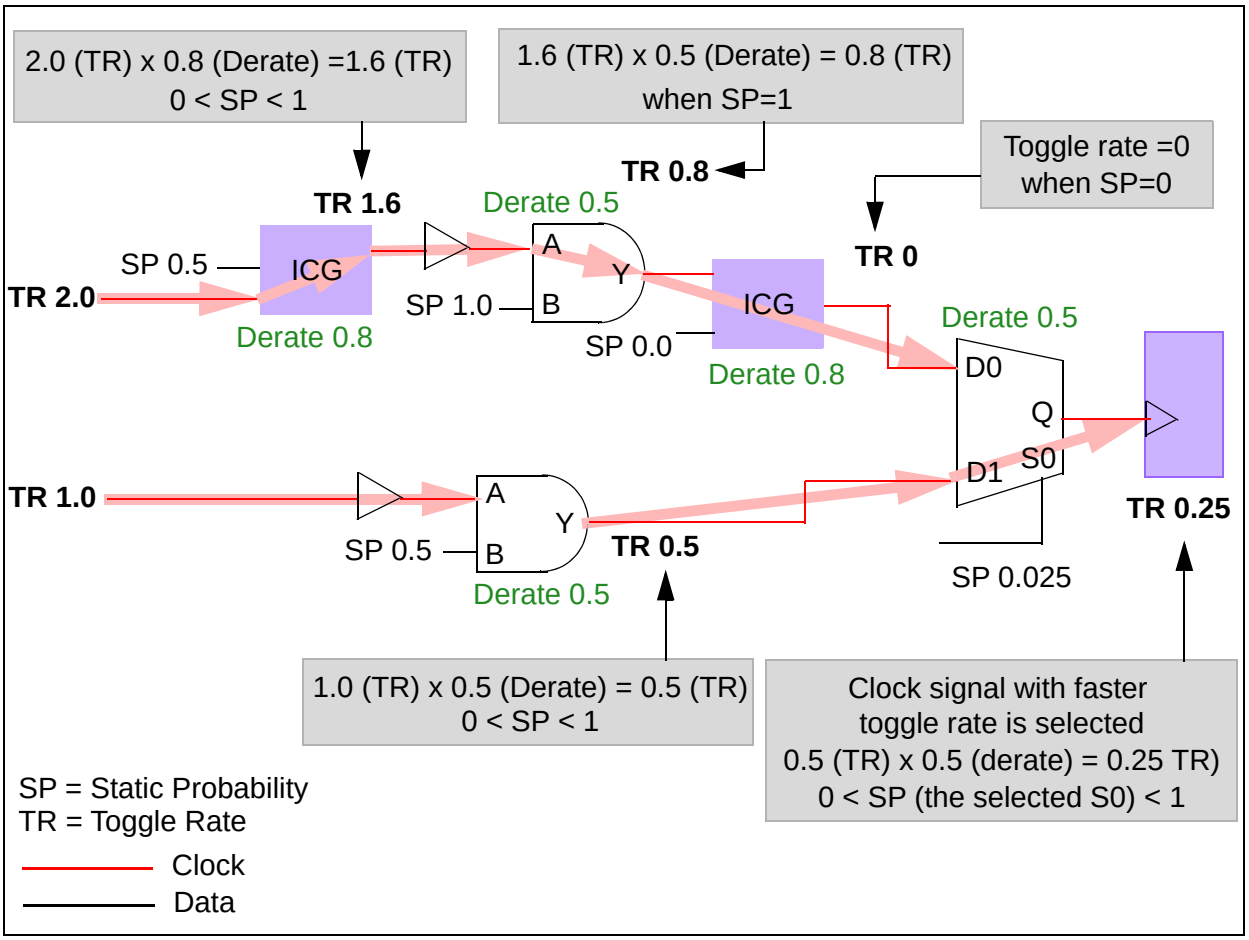
- Propagation follows the function of the cell
- No change to the current PrimePower behavior



- Clock toggle propagates activity at D1 to Q without modification
- Propagated activity can be scaled also when needed (the default is 0)

- Propagation follows the function of the cell
- No change to the current PrimePower behavior

Figure 8-4 An Example of Propagating Activities Through Clock Networks



Using the set_activity_derate Command

When the `through_mode` propagation is enabled, you can set a derating factor on the toggle rate at the cells in the clock network by running the `set_activity_derate` command, and then propagate the derated activity to the downstream logics. The logic cells multiply the activity by the specified derating factor when passing through from the inputs of the cells to the outputs. The toggle derating factor set by the `set_activity_derate` command takes precedence over the one set by the `set_switching_activity -clock_derate` command.

Use the `-cell_type` option to specify if the derating factor is applied to the clock-gating cells or the combinational cells.

To reset the derating factor on a cell to the default value (that is, 1.0), use the `reset_activity_derate` command.

To check the derating factor set on a cell, use the `report_activity_derate` command.

Examples

The following example enables the `through_mode` propagation in the clock network.

```
pt_shell> set_power_analysis_options -through_mode
```

The following example sets a derating factor of 1 on the clock gating cells in the design, excluding buffers and inverters.

```
pt_shell> set_activity_derate -type clock_gating_cell -derate_factor 1
```

The following example sets a derating factor of 0.1 on all cells in the design.

```
pt_shell> set_activity_derate -derate_factor 0.1 [get_cells *]
```

Assigning Annotated Clock Network Power

For gate-level designs, you might estimate clock network power based on certain parameters controlling the clock tree synthesis in other tools, such as Design Compiler topographical mode or a transistor-level simulator. You can then annotate these estimated power values to the specified clock domains in PrimePower by using the `set_annotated_clock_network_power` command.

By specifying the related clock, you can annotate the power of each clock domain separately in the clock network. You can also annotate the total power either as switching power, internal power, or leakage power separately.

When you specify annotated values for clock network power, PrimePower replaces the estimated or calculated clock network power with the annotated power values. The succeeding `report_power` command uses the annotated clock network power in the summary reports. As a result, the power report generated by the `report_power` command is as accurate as those reported by transistor-level simulation tools or the physical aware tools, such as the Design Compiler topographical technology.

To remove the previously annotated power values, use the `remove_annotated_clock_network_power` command.

Example

The following example annotates internal power and switching power separately for clock domains test1 and test2, respectively.

```
pwr_shell> create_clock -name test1 -period 10 clka
pwr_shell> create_clock -name test2 -period 5 clka
pwr_shell> set_annotated_clock_network_power -internal_power 1.0e-03 \
             -switching_power 2.0e-03 -clock test1
pwr_shell> set_annotated_clock_network_power -switching_power 2.0e-03 \
-clock test2
pwr_shell> report_power
```

Output Report:

```
****************************************
Report : Statistical Average Power
Design : my_design
Version: my_version
Date   : ....
****************************************
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group
      e  -  Annotated clock network power excluded

                Internal  Switching  Leakage    Total
Power Group     Power     Power      Power      Power   (%)       Attrs
------------------------------------------------------------------------
io_pad          0.0000    0.0000    0.0000    0.0000 (0.00%)
memory          0.0000    0.0000    0.0000    0.0000 (0.00%)
black_box       0.0000    0.0000    0.0000    0.0000 (0.00%)
clock_network   3.536e-03 0.0000    0.0000    3.536e-03 (36.81%)  ei

register        3.060e-03 0.0000    1.020e-05 3.071e-03 (31.96%)
combinational   0.0000    0.0000    0.0000    0.0000 (0.00%)
sequential      0.0000    0.0000    0.0000    0.0000 (0.00%)

                     Internal  Switching  Leakage  Total
Clock                Power     Power      Power    Power       (%)Attrs

------------------------------------------------------------------------
CKB                  0.0000     2.000e-03  0.0000  2.000e-03
CKA                  1.000e-03  0.0000     0.0000  1.000e-03
------------------------------------------------------------------------
All Annotated Clocks  1.000e-03  2.000e-03  0.0000  3.000e-03   (31.23%)
  Net Switching Power  = 2.000e-03   (20.82%)
  Cell Internal Power  = 7.596e-03   (79.07%)
  Cell Leakage Power   = 1.020e-05   ( 0.11%)
                         ---------
Total Power            = 9.607e-03  (100.00%)
```

Estimating Clock Network Power Consumption

To estimate the additional power incurred by a clock network before actually inserting it to the design, use the `estimate_clock_network_power` command. This allows you to perform a more accurate analysis of the total design power when clock network synthesis has not yet been performed.

Note:

The `estimate_clock_network_power` command is supported only in the averaged power analysis mode.

You must identify the design clocks before the `estimate_clock_network_power` command.

Based on the constraints and the type of the buffer you specify, PrimePower inserts buffers for all the nets in the clock network, and builds a virtual balanced clock network for power calculation. The clock trees are built on the fly and the original design is not touched or modified.

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, the SAIF file, or the VCD file.

The following guidelines are used to build the clock network:

- The fanout of each buffer in the clock tree is no bigger than the maximum fanout.
- The delay from the root to the leaf of the network is minimized.
- The driven registers are put at the same and lowest level of the network.
- The deviation of buffer fanouts at the same network level is minimized.
- For clock-gating cells, by default separate networks are built based on the fanout of the clock-gating cell, using the same rules as specified above. If you want PrimePower to consider clock-gating cells to be part of the clock network, set the `power_clock_network_include_clock_gating_network` variable to `true`. The default is `false`.

  To ignore clock-gating cells when estimating the clock tree power, use the `-ignore_clock_gating` option. This allows you to investigate savings from clock-gating by comparing the power numbers with or without the clock gating effect.
- To include the power consumed by the registers driven by the clock in the report, use the `-include_registers` option.

Example

The following example estimates power for clock CLK by using the buffer buf1a1 of library ssc_core_typ to build the clock tree. The wire load model in the design is used to calculate the wire capacitance. The clock input transition is 0.2.

```
pwr_shell> create_clock -name CLK -period 10 clk
pwr_shell> set_clock_transition 0.2 CLK
pwr_shell> estimate_clock_network_power ssc_core_typ/buf1a1
```

Example 8-1shows the output of the `estimate_clock_network_power` command. It includes the power consumption as well as detailed information about the pseudo-clock network, driven registers, and timing propagation.

Example 8-1 Report Generated by the estimate_clock_network_power Command

```
**************************************
Report : estimated clock network power
Design : reg16
...
**************************************

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
```

Showing Transitions and Clock-Gating Circuitry

The presence of clock-gating circuitry leads to a nonzero transition time on the gated clock signal. This increases with the number of flip-flops being gated by the signal. A large transition time at the clock pin of the gated flip-flop leads to a very high internal power usage. However, this is not realistic because PrimePower inserts buffers to reduce clock edge transition time. Thus, setting the clock transition to 0 ensures the most accurate power analysis result after clock-gating circuitry insertion and before clock tree synthesis.

To avoid huge cell internal power numbers in the design, run the following commands:

- Set the transition time for the ideal clock to 0.

  ```
  pwr_shell> set_clock_transition 0 [all_clocks]
  ```

- Run the `set_ideal_network` command to set all the clock nets as sources of the ideal network.

Reporting Clock Network Power

Use the `report_power` command to report power values for clock network power groups, register power groups or clock domains, by using the options listed in Table 8-2.

Table 8-2 Commonly Used Options for Reporting Clock Network Power

Option	Description
`-groups clock_network|register`	Reports power for the cells and nets in the specified clock networks or register power groups.
`-clock`	Reports power for the cells and nets in the specified clock domains.
`-include_estimated_clock_network`	Reports the total power of the design that is calculated by the `estimate_clock_network_power` command. The clock network power is added to the total design power.

For more information about how to generate cell-based power reports, see RedHawk Analysis Fusion in IC Compiler II.

9

PST-Based Power Analysis

A power state table (PST) is a UPF construct which defines the legal combinations of voltage values and states of the power and ground supplies for all supply sets in the design.

PrimePower supports UPF PST-based power analysis in the average mode. The tool reads the necessary information from the PST commands, performs power analysis, and reports power consumptions per valid power state. The report lists different power components, including internal, switching and leakage power, at both the design and instance levels.

PrimePower supports UPF 3.0 PST commands.

This chapter contains the following topics:

Power Analysis With PST Commands

PrimePower supports a subset of UPF 3.0 PST commands and issues an error message for any unsupported UPF PST commands and options. The tool reads and ignores legacy UPF PST `add_port_state`, `create_pst`, and `add_pst_state` commands.

For power saving purposes, sometimes a block is shut down when inactive by using power switches to shut off the power supply to the block. UPF PST commands are used to define the state (either ON or OFF) for power supplies. During PST-based power analysis, PrimePower adds the state information to a supply set or a group object defined in the power state tables created using the `add_power_state` command, and calculates power consumption for the valid power state combinations.

Though a design block can operate at different voltage levels, dynamic voltage scaling is not supported in PST-based power analysis. When two power state tables are identical only in voltage values, PrimePower considers them equivalent for power analysis; the power consumption results are expected to be the same for the equivalent power state tables. When the `-pst` option of the `report_power` command is not specified, PrimePower reports the regular power.

During averaged power analysis, PrimePower uses the statistical activity information for power calculation, and assumes no toggle happened at the power-off state. If nonzero toggle rate is on the cell pin that is being powered off, PrimePower automatically applies a zero toggle rate on the cell pin for power calculation.

PST-Based Power Analysis Flow

Execute the following steps to perform averaged power analysis with power state tables:

1. Set the `upf_enable_pst` variable to TRUE to enable the PST-based power analysis capability.

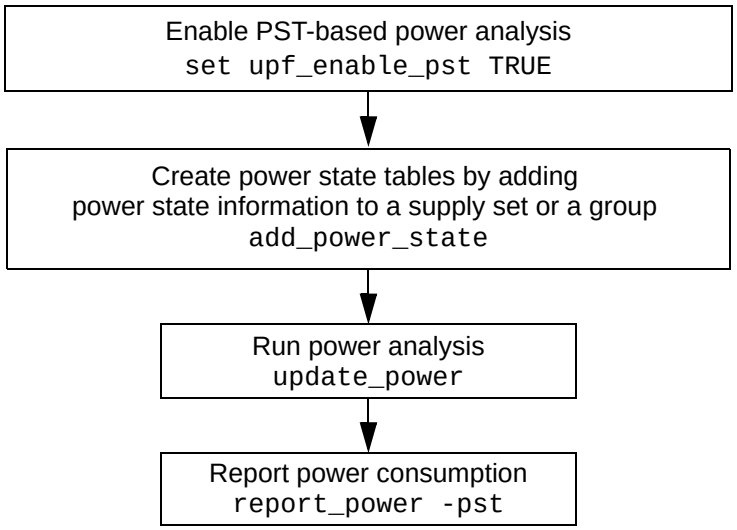
   When PST-based power analysis is complete, PrimePower reports power consumptions per valid power state table at the top level.

2. Create the power state table using the commands listed in Creating Power State Tables.

3. Run the `update_power` command to perform power analysis in the averaged mode.

4. Run the `report_power -pst` command to examine power consumptions per valid power state tables at the top-level design, including internal, switching and leakage power values.

Figure 9-1 Analysis Flow for Running Power Analysis With Power State Tables



Creating Power State Tables

Before running PST-based power analysis, you can create power state tables with the following commands:

- `add_power_state`: Adds the state information to a supply net or a group created by the `create_power_state_group` command. Use the `-state` and `-supply_expr` options to specify the power state status on a supply net (in the Boolean expression).

  For example, to define two power states (ON and OFF) for the supply set TEST where the supply expression is a Boolean expression of power and ground with a voltage value, type

```
prompt> add_power_state -supply TEST \
          -state ON  {-supply_expr \
{power=={FULL_ON 0.8} && ground={FULL_ON 0}}}
          -state OFF {-supply_expr \
{power=={OFF} && ground=={FULL_ON 0}}}
```

  The following example adds power states to a power state table group, called PST_GROUP. The group is in state RUN when both supply sets TEST1 and TEST2 are in state ON. The system is in state SLEEP when supply set TEST1 is in state ON while supply set TEST2 is in state OFF.

```
prompt> create_power_state_group PST_GROUP
prompt> add_power_state -group PST_GROUP
        -state RUN   {-logic_expr {TEST1 == ON && TEST2 == ON}}
        -state SLEEP {-logic_expr {TEST1 == ON && TEST2 == OFF}}
```

- `create_power_state_group`: Defines a power state table group to be used by the `add_power_state` command. For example,

  ```
  prompt> create_power_state_group PST_GROUP
  ```

- `report_pst`: Reports the power state table defined by the `add_power_state` command in the current design. Use the `-verbose` option to report the power state of a net set.

  For example,

  The following UPF script defines the power states on the supplies and then the legal power state combinations for the design. The `report_pst` command reports the resulting power state table.

  ```
  prompt> add_power_state -supply TEST1 -state ON  \
          {-supply_expr {power=={FULL_ON 0.8} && ground=={FULL_ON 0}}}
  prompt> add_power_state -supply TEST2 -state ON  \
          {-supply_expr {power=={FULL_ON 0.8} && ground=={FULL_ON 0}}}
          -state OFF \
          {-supply_expr {power=={OFF} && ground=={FULL_ON 0}}}
  prompt> create_power_state_group MY_PST_GROUP
  prompt> add_power_state -group MY_PST_GROUP
          -state RUN   {-logic_expr {TEST1 == ON && TEST2 == ON}} \
          -state SLEEP {-logic_expr {TEST1 == ON && TEST2 == OFF}}
  prompt> report_pst

    Total of 1 power state group defined for design 'mydesign'.
  ------------------------------------------------------------------------
      Power State Group : MY_PST_GROUP
        Scope : <top level>
        Group Members : TEST1        TEST2
        Total of 2 power state combinations on this group :
        RUN           : ON           ON
        SLEEP         : ON           OFF

  prompt> report_pst -verbose
    Total of 1 power state group defined for design 'mydesign'.
  ------------------------------------------------------------------------
      Power State Group : MY_PST_GROUP
        Scope : <top level>
      Group Members : TEST1          TEST2
       Total of 2 power state combinations on this group :
        RUN           : ON           ON
        SLEEP         : ON           OFF
    Power states defined on supply sets for design 'top'.
  ------------------------------------------------------------------------
      Supply Set : TEST1 has 1 state defined:
        State : ON, power : FULL_ON 0.90, ground : FULL_ON 0.00
  ------------------------------------------------------------------------
      Supply Set : TEST2 has 2 states defined:
        State : ON, power : FULL_ON 0.90, ground : FULL_ON 0.00
        State : OFF, power : OFF, ground : FULL_ON 0.00
  ```

Script Examples

The following script example contains the commands to analyze and report shutdown power in averaged mode using UPF power state table constructs. The script first reads design and activity data, loads UPF information, and adds power state information. It then checks the defined PST states and groups using the `report_pst` command and specifies the static probability on nets if any. When finished, the script runs power analysis with the `update_power` command and reports shutdown power per UPF power state using the `report_power -pst` command.

The script also saves the current session for later retrieval. It allows you to restore the saved UPF PST power session for what-if purposes.

Table 9-1 lists the power state information used in this script example.

Table 9-1 Power State Tables

State	SSI	SS2	SS3
RUN	ON (0.9V)	ON (0.9V)	ON (0.9V)
SLEEP	ON (0.8V)	ON (0.9V)	OFF (0.0V)

```
## Variables ##
set upf_enable_pst true #Enable UPF PST analysis
set power_enable_analysis true
set power_analysis_mode  averaged

## Read design data ##
set search_path ". "
set link_library "* my_upf_pst.db"
read_verilog ./test.v
link
set_input_transition 0.2 [all_inputs]
set_load 0.3 [all_outputs]

##  Load UPF ##
create_supply_set SS1
create_supply_set SS2 -function { ground SS1.ground }
create_supply_set SS3 -function { ground SS1.ground }

##  Add PST state information ##
add_power_state -supply SS1 -state ON \
     {-supply_expr {power == {FULL_ON 0.9} && ground == {FULL_ON 0}}}
add_power_state -supply SS2 -state ON \
     {-supply_expr {power == {FULL_ON 0.9} && ground == {FULL_ON 0}}}
add_power_state -supply SS3 -state ON \
     {-supply_expr {power == {FULL_ON 0.9} && ground == {FULL_ON 0}}} \
```

```
      -state ON2 {-supply_expr {power == {FULL_ON 1.0} && ground ==
      {FULL_ON 0}}} \
      -state OFF {-supply_expr {power == {OFF} && ground == {FULL_ON 0}}}
create_power_state_group MY_PST
add_power_state -group MY_PST -state RUN \
      {-logic_expr {SS1==ON && SS2==ON && SS3==ON}} \
      -state RUN2 {-logic_expr {SS1==ON && SS2==ON2 && SS3==ON2}} \
      -state SLEEP {-logic_expr {SS1==ON && SS2==ON && SS3==OFF}}

## Report PST based on the group and the power states defined on  ##
## supply sets  ##
report_pst -verbose

## Run power analysis ##
update_power

## Report power per UPF power state  ##
report_power –pst

## Save the session for later retrieval  ##
save_session ./upf_session
remove_design –all

## Restore a UPF PST power session and perform what-if analysis  ##
restore_session ./upf_session
report_pst
report_power -pst
set_supply_net_probability SS1.power 0.5
report_power -pst
```

10

Analyzing Hierarchical Designs

PrimePower supports distributed peak-power analysis using the distributed multi-scenario analysis (DMSA) infrastructure and PrimeTime distributed timing analysis to help reduce memory resources or excessive runtime when analyzing a large chip design.

This chapter contains the following topics:

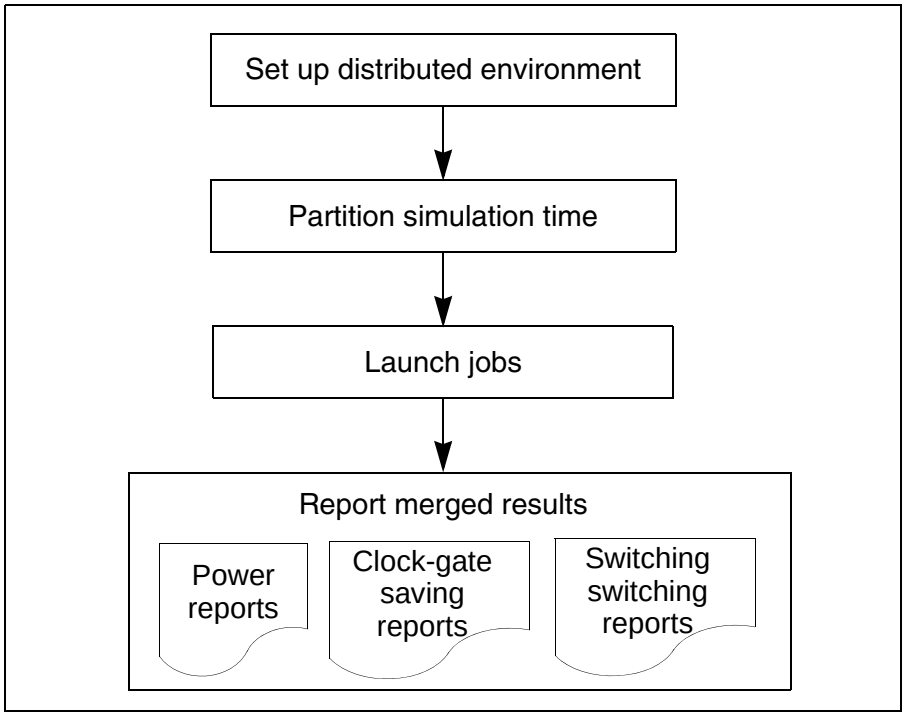
Distributed Peak-Power Analysis

To speed up the peak power analysis process, PrimePower performs distributed peak-power analysis using the distributed multi-scenario analysis (DMSA) infrastructure available in the PrimeTime tool.

The DMSA flow evaluates several scenarios in parallel by using multiple processors. This mechanism provides faster turnaround time and the ability to analyze the results from the multiple scenarios in parallel. For timing analysis, a scenario is a set of operating conditions and modes; for power analysis, a scenario is a time window in the logic simulation activity file.

The steps for setting up the design for a distributed environment, mapping the analysis to scenarios, launching multiple scenario runs in parallel, and merging the results are similar to the steps in PrimeTime and are compatible for both timing and power analysis, as shown in Figure 10-1.

Figure 10-1 Distributed Peak-Power Analysis Flow



To perform distributed peak-power analysis,

1. Set up a distributed computational environment in which the loaded design can be used to analyze power for multiple simulation windows simultaneously.

In this step, you specify the working directory, the log files, and the licensing mode. PrimePower also supports the PrimeTime core-based licensing model which considers the total number of CPU cores, instead of the number of scenarios. For more information, see Scenario-Based and Core-Based Licensing Models.

This step also includes setting up and starting a distributed pool of machines. The variables and commands that you use in setting up the distributed environment for peak power analysis are the same as the ones that you use in the PrimeTime tool for timing analysis.

Example 10-1 shows the commands for a distributed environment for peak power analysis using two machines in the LSF, two PrimeTime licenses, and two PrimePower licenses.

Example 10-1 Setting Up Distributed Environment for Peak Power Analysis

```
set_app_var power_enable_analysis true
set multi_scenario_working_directory ./work
set multi_scenario_merged_error_log ./work/error_log.txt
set_host_options -32bit -num_processes 2 \
        -submit_command /lsf/bin/bsub
start_hosts
```

2. Partition the simulation time and perform peak power on each time window.

   This step divides the total simulation time for the peak power analysis into smaller time windows.

   To run peak power analysis on each time window, create a scenario for each partition using the `create_scenario` command. Use the `-common_data` and `-common_variables` options to specify the data and variables that are common across the scenarios. Use the `-specific_data` option to specify the scenario-specific data.

   The following steps are common to all the scenarios before reading the VCD file:

   a. Read and link the design.

   b. Read and apply the constraints and parasitics.

   c. Update the timing constraints.

   The steps performed to read different time windows of the VCD file are specific to each scenario.

   The following script example creates two simulation scenarios and performs peak power analysis for the two scenarios:

```
set test_dir .
# The common data is used by all the scenarios. Since all the
# scenarios must read the design data and update timing.
create_scenario -name run_0 -specific_data run_0.tcl \
    -common_data common.tcl -common_variables {test_dir}
create_scenario -name run_1 -specific_data run_1.tcl \
```

```
    -common_data common.tcl -common_variables {test_dir}

set_app_var power_enable_analysis true
set_app_var power_analysis_mode time_based
set_app_var search_path "."
set_app_var link_library    " * link_library.db"
read_verilog   mac.vg
current_design mac
link
read_sdc $test_dir/mac.sdc
set_disable_timing [get_lib_pins ssc_core_typ/*/G]
read_parasitics $test_dir/mac.spef.gz
update_timing

# The simulation time from 0 to 10000 is divided into two blocks, 0 to
# 5000 and 5001 to 10000.
# The script run_0.tcl, specific to the run_0 scenario
read_vcd -time {0 5000} $test_dir/vcd.dump.gz -strip_path "tb/macinst"

# The script run_1.tcl, specific to the run_1 scenario
read_vcd -time {5001 10000} $test_dir/vcd.dump.gz -strip_path "tb/
macinst"
```

3. Launch the jobs.

   When you setup the distributed environment and created the scenarios, use the `current_session` command to select a set of scenarios for analysis. Use the `remote_execute` command to execute one or more commands on the remote machines.

   The `remote_execute` command builds a buffer of commands and triggers the execution of the commands in the buffer on the remote machines. The commands in the buffer are executed in the order in which they are listed. You can run this command only after you have selected the scenarios using the `current_session` command.

   The following script example shows how to launch your jobs to run peak power analysis in a distributed environment.

   ```
   current_session -all
   remote_execute { \
      report_power
   }
   ```

   For more information about the actual launching of the jobs and selecting the physical machines to run the jobs, see *PrimeTime User Guide*.

4. Generate merged reports. You can:

   - Run the `report_power` command to generate a combined report which records all the power data from separate slave processes, and to identify the maximum peak power and its corresponding peak time from all the scenarios.

     ```
     pwr_shell> report_power

     ****************************************
     Report : Merged Power Report
     Design : top
     Version: <version>
     Date   : <time>
     ****************************************
     Scenario Name   Switching Power     Internal Power   Leakage Power
     Total Power     X-Transition Power  Glitch Power
     Peak Power      Peak Time           Rail Names
     ------------------------------------------------------------------
     S1              1.3297e-06          2.9673e-05       1.17158e-08
     3.10144e-05     0                   1.2605e-07
     0.00544426      0.101               VDD1_ADD

     S2              0                   0                1.17158e-08
     1.17158e-08     0                   0
     2.34316e-08     1                   VDD1_ADD

     S3              0                   0                1.17158e-08
     1.17158e-08     0                   0
     2.34316e-08     2                   VDD1_ADD
     ```

   - Generate a merged report from all the slave sessions using a subset of the `report_power` and `report_clock_gate_savings` commands.

     The tool first calls the following commands with the respective options in the child processes and then merges the power reports from each child process into a single power report for the entire analysis interval.

     These options for the `report_power` and `report_clock_gate_savings` commands are supported only in the DMSA flow for report merging.

     ```
     report_power -include_boundary_nets
     report_power -include_boundary_nets -cell_power -leaf
     report_clock_gate_savings -by_clock_gate -sequential
     report_power -include_boundary_nets -cell_power \
                  -net_power -groups clock_network -leaf
     ```

5. Verify the switching activity for the entire simulation time.

   Use the `get_switching_activity` command to view the toggle rate and static probability for the selected design objects.

Note:

Tcl collections are not supported within the PrimeTime DMSA infrastructure; you cannot specify lists of design objects with the `get_nets`, `get_cells`, and `get_pins` commands.

See Also

- Generating Reports
- Reporting Annotated Power

Scenario-Based and Core-Based Licensing Models

By default, PrimePower performs distributed power analysis using the scenario-based licensing model, wherein analyzing each activity window requires its own license while it is running. For example, analyzing four simulation activity windows requires one PrimePower license per window.

You can use the core-based licensing model in which one license is required for each 32 core processes running for an analysis, irrespective of the number of activity file windows being analyzed.

To select core-based licensing, change the `multi_scenario_license_mode` variable setting from `scenario` (the default) to `core` in the DMSA master process. For example,

```
pwr_shell> set_app_var multi_scenario_license_mode core
```

After you set the variable, running any number of simulation window scenarios requires just one license set per 32 cores used. These 32 cores can be divided among multiple scenarios in any combination. For example, one license set can handle any of the following:

1 scenario x 32 cores per scenario

2 scenarios x 16 cores per scenario

4 scenarios x 8 cores per scenario

16 scenarios x 2 cores per scenario

If you need to run more than 32 cores, each additional 32 (or fraction thereof) requires another license set:

1 to 32 cores: 1 license set

33 to 64 cores: 2 license sets

65 to 96 cores: 3 license sets

97 to 128 cores: 4 license sets

That how many cores can be used is limited to what is available on the server. For example, in a dual core server, threads are launched to use only the two available cores.

Use the `-max_cores` option of the `set_host_options` command to specify the maximum number of CPU cores used per process. The largest allowed maximum is 32. For example,

```
pwr_shell> set_host_options -max_cores 32 -num_processes 4
Information: Checked out license 'PrimePower' (PT-019)
```

Setting the maximum number of cores to a value in the range from 17 to 32 checks out one PrimePower license. The tool retains the license even if the maximum number of cores is later reduced to 16 or less.

Calculating Power During PrimeTime Distributed Timing Analysis

PrimeTime provides distributed timing analysis to analyze timing on large hierarchical designs with hardware resource constraints. You can calculate averaged power values on each partition and the master of hierarchical designs when running distributed timing analysis.

Note:

PrimePower supports only averaged power analysis in the PrimeTime distributed timing analysis flow.

To analyze power in the PrimeTime distributed timing analysis flow:

1. Annotate switching activities and perform the light weight timer update on the whole design before partitioning it into blocks. The tool then propagates the annotated activity with zero-delay simulation to determine the activity on all other nets in the design.
2. Run the `get_switching_activity` commands to report the averaged switching activity data for each partition in the design, using the distributed timing analysis infrastructure.
3. Run the `update_power` command on each of the partitions as separate child processes to calculate power values and update activity on the partition nets.
4. Run the `report_power` command to generate power reports at the master process using the distributed analysis infrastructure. This allows the distributed analysis master to have complete power and activity data on the instances and nets in each partition of the entire design. All power and activity reporting and query commands work in the master thereafter.

   Using the `report_power_calculation` command to calculate power values is also supported.

11

Cycle-Accurate Peak Power Analysis

This chapter describes how to specify the RTL switching activity files, and how PrimePower uses the activity to calculate power in the cycle-accurate peak power analysis mode. In this mode, the tool calculates power per event accurately, and generates power waveforms at the clock-cycle resolution.

This chapter contains the following topics:

Cycle-Accurate Peak Power Analysis Flow

Using the switching activity from the RTL activity file, PrimePower performs name mapping to match the RTL objects to the gate-level objects. If a name mapping file is available from the Design Compiler tool, you must source it before the `read_vcd` or `read_fsdb` command. If the name mapping file is not available, the tool performs name matching using a built-in algorithm. For more details, see Name Mapping.

PrimePower checks for the appropriate clock edge within the time stamp before propagating activity through edge-triggered elements. This ensures that propagation occurs only when the correct clock edge (either rise or fall) has occurred and the appropriate data value is passed to the Q output pin, thus preventing excess activity when edge-triggered elements are part of a feedback loop.

When you run the `update_power` command, PrimePower processes the events in the activity file, and propagates the activity to the non-annotated portions of the design. The tool sums up the energy per clock cycle for all the events, and averages the power over the cycle.

PrimePower identifies generic integrated clock gating (GICG) cells through the `clock_gating_integrated_cell:"generic"` attribute and the latch model description in the .lib file; it propagates switching activities through GICG cells and integrated clock gating (ICG) cells with latch constructs in the cycle-accurate peak power mode. The propagation is dependent on the clock or the test enable signal of the cell. When power analysis is complete, you can query accurate activity data on GICG pins using the `get_switching_activity` command and the power of the GICG cells using the `report_power` command. In the `report_power` report, the power of GICG cells is included in the `clock_network` group.

For more information about how to use the `clock_gating_integrated_cell:"generic"` attribute, see the related chapter in the *Library Compiler User Guide*.

Sometimes, event propagation through edge-triggered sequential elements might lead to excess activity and over-estimation of power in the cycle-accurate time-based analysis mode. In this case, disable event propagation by setting the `power_capp_edge_triggered_propagation` variable to `false` before the `update_power` command. The default is `true`.

To calculate power in the cycle-accurate peak power analysis mode,

1. Set the `power_analysis_mode` variable to `time_based`.
2. Run the `read_vcd -rtl` command to read the switching activity data from the specified RTL activity file. When the `-rtl` option is specified, PrimePower enables name mapping and event propagation and enables the cycle-accurate peak power analysis mode.

   Unannotated nets in the cycle-accurate peak power mode are assigned with uninitialized activity (that is, TR=-1), an `activity_source` of UNINITIALIZED, and a static probability

equal to the set_case_analysis value found on the nets if no value is propagated to the nets during update_timing.

You can use the set_case_analysis command to specify a constant logic value to a port or pin when the activity file does not have activity data for primary inputs.

For more information, see Reading Switching Activity Information in Time-Based Mode.

3. Specify additional options by using the set_power_analysis_options command.
   - -waveform_format: Specify the format of the output waveform file. Valid values are fsdb (default), out and none.
   - -waveform_output: Specify the prefix of the output waveform file. The default is primetime_px.
   - -cycle_accurate_clock: Specify the reference clock to be applied in the analysis. The power consumed by the design is considered constant over one or more cycles of the specified clock. The default is the fastest clock in the design.
   - -cycle_accurate_cycle_count: Specify the number of clock cycle over which the power consumed by the design is considered constant when VCD is RTL or zero delay. The default is 1.

   Specify other options as needed.
4. Run the update_power command to calculate power of the design in cycle-accurate peak power analysis mode.
5. Run the report_power command to generate power reports.

The following example performs cycle-accurate peak power analysis to generate a cycle-accurate waveform by using an RTL VCD file as input:

```
pwr_shell> set power_analysis_mode time_based
pwr_shell> read_vcd myRTL.vcd -rtl
pwr_shell> set_power_analysis_options -waveform_output capp_waveform \
                      -waveform_format out
pwr_shell> update_power
pwr_shell> report_power
```

Specifying Activity With the set_case_analysis Command

If the activity file does not contain switching activity for primary inputs, use the set_case_analysis command to assign a constant value to a port. This is useful for applying activity to test enable signals, which exist in the gate-level netlist, not in the RTL-based activity file.

Because the annotation in the activity file takes precedence over the set_case_analysis command, it can only be used to annotate signals, which are not present in the activity file.

Glitch Power During Cycle-Accurate Peak Power Analysis

In the cycle-accurate peak power mode, the tool considers glitches as extra transitions within a clock cycle before a signal settles to its steady state. When multiple inputs are in transition simultaneously, the tool treats them as separate events, resulting in multiple transitions in the output. The power consumption of these extra transitions is scaled by 0.5.

```
glitch power = 0.5 * dynamic power for extra transitions
```

To report the estimated glitch power, run the `report_power` command. The estimated glitch power is included in the total power value reported.

```
pwr_shell> report_power
Net Switching Power = 2.252e-03 (44.82%)
Cell Internal Power = 2.773e-03 (55.18%)
Cell Leakage Power  = 2.594e-07 ( 0.01%)
---------
Total Power = 5.025e-03 (100.00%) #Including the estimated glitch power
X Transition Power = 2.171e-07
Estimated Glitching Power = 7.569e-04
Peak Power = 0.0968
Peak Time = 1956.00
```

With no glitches, the total power is

```
Total power = 5.025e-3 - 7.569e-4 = 4.268e-3
```

If all glitch toggles occur as full transitions, the total power value is

```
Total power = 5.025e-3 + 7.569e-4 = 5.7819e-3
```

Delay-Aware Peak Power Analysis

When running either cycle-accurate or zero-delay gate-level peak power analysis with RTL-level or gate-level VCD or FSDB files, PrimePower generates optimistic power numbers. This is because in RTL-level or zero-delay gate-level VCD or FSDB files, transitions on all the signals occur at a given clock edge, and events are always aligned with certain clock edges. The leakage and dynamic energy associated with any cell event is distributed over the reference clock period, which leads to the low instantaneous peak power.

To calculate accurate peak power numbers, use the delay-aware peak power analysis capability to shift all the events to be at a given time stamp by a delay factor associated with the corresponding pin or net. The cell delay and net delay values are computed by PrimeTime.

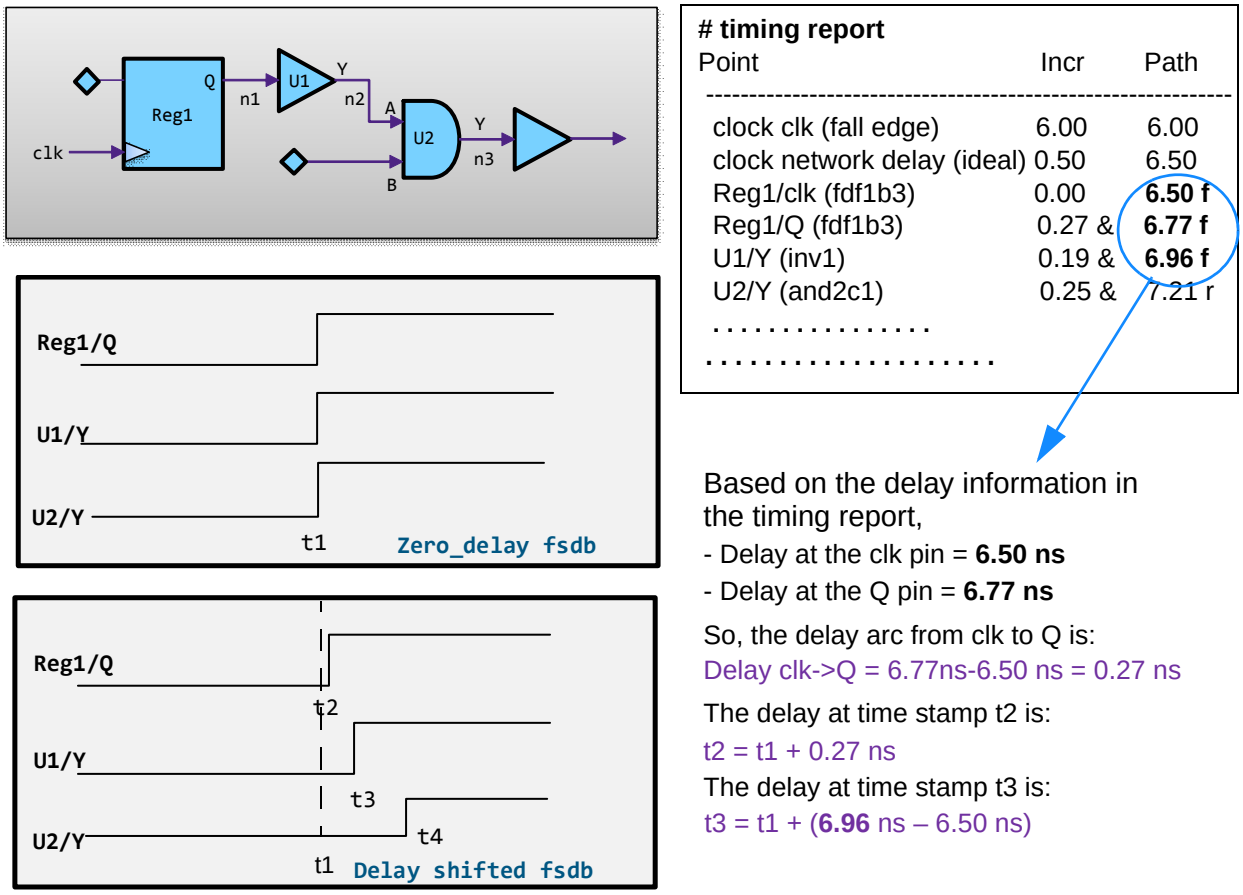
Note:

Delay-aware activity shifting is supported in cycle-accurate peak power analysis. Both propagated RTL activity and zero-delay gate-level activity in cycle-accurate peak power analysis can be delay shifted.

Activity on high fanout nets declared as an ideal network are not delay shifted.

Figure 11-1 and Figure 11-2 show how PrimePower uses the delay information from the timer to shift activity to a different time stamp in low and high effort mode.

Figure 11-1 An Example of Delay Shifting



 # timing report

Point	Incr	Path
clock clk (fall edge)	6.00	6.00
clock network delay (ideal)	0.50	6.50
Reg1/clk (fdf1b3)	0.00	**6.50 f**
Reg1/Q (fdf1b3)	0.27 &	**6.77 f**
U1/Y (inv1)	0.19 &	**6.96 f**
U2/Y (and2c1)	0.25 &	7.21 r
. . . . .		
. . . . .		

Based on the delay information in the timing report,

- Delay at the clk pin = **6.50 ns**
- Delay at the Q pin = **6.77 ns**

So, the delay arc from clk to Q is:

Delay clk->Q = 6.77ns-6.50 ns = 0.27 ns

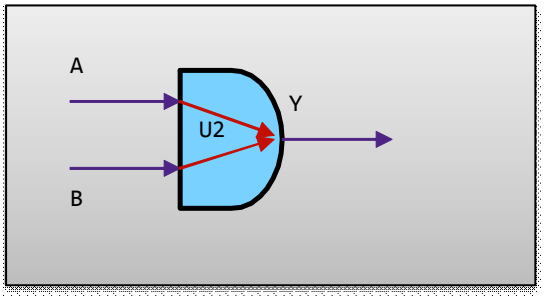
The delay at time stamp t2 is:

t2 = t1 + 0.27 ns

The delay at time stamp t3 is:

t3 = t1 + (**6.96** ns – 6.50 ns)

Figure 11-2 Shifting Delays in an AND Gate Cell



In this multi-input AND gate, for the given slew and load condition:

Delay at input A = 0.23 ns
Delay at input B = 0.34 ns

Cell arc from A to Y (A->Y) = 0.25 ns
Cell arc from B to Y (B->Y) = 0.41 ns

When shifting delays with the `-effort_level high` option, PrimePower chooses the cell arc based on the I/O PATH. In this case, the cell arc A->Y is chosen.
A->Y = 0.25 ns t4 = t1 + delay at A + arc delay = t1 + 0.23 ns + 0.25 ns

When shifting delays with the `-effort_level low` option, PrimePower chooses the cell arc with the worst delay. In this case, the arc B->Y for U2 is chosen.
B->Y =0.41 ns t4 = t1 + delay at B + arc delay = t1 + 0.34 ns + 0.41 ns

Delay-Aware Peak Power Analysis Flow

By using the delay information from the timer, you can run the peak power analysis with RTL-level and zero-delay gate-level event files (either in VCD and FSDB format) to calculate accurate peak power numbers.

To perform peak power analysis with delay-shifted annotation,

1. Enable the delay-aware peak power analysis by setting the `power_enable_delay_shifted_event_analysis` variable to `true`.
2. Define the options for delay shifting by running the `set_power_delay_shifted_event_analysis_options` command. Specify the following options as necessary:

Option	Description
`-effort_level`	Defines the effort level for delay shifting to be high or low. The default is low.
`-write_activity_file`	Writes out the shifted activity information to a file (in the FSDB format) during power update for debugging purposes.

Option	Description
`-include_ext_del`	Shifts the activity at the primary input ports by the delay values defined by the `set_input_delay` command. When not specified, PrimePower shifts the activity at both clocks (in a post clock tree synthesis design) and data paths, not the activity at the primary port activity.

Note:

X -> 1 or Z-> 1 is considered equivalent to 0 -> 1 transition for delay shifting purposes. Glitch power calculation is not supported when delay-aware activity shifting is enabled.

To report the delay-aware activity shifting options set by `set_power_delay_shifted_event_analysis_options`, run the `report_power_delay_shifted_event_analysis_options` command.

To reset the delay-aware activity shifting options to default, run the `reset_power_delay_shifted_event_analysis_options` command.

3. Run the `update_power` command to perform peak power analysis.

Sample Script

```
#PrimePower script (ASCII flow) for enabling delay-aware activity
#shifting in cycle-accurate peak power/gate-level zero-delay
#time_based mode

set power_analysis_mode time_based
set enable_timing_analysis true
set power_enable_delay_shifted_event_analysis true
...
set_power_delay_shifted_event_analysis_options \
 -effort_level high -write_activity_file shifted.fsdb
set_power_analysis_options . . .
...
read_fsdb –rtl ./file.fsdb –strip_path 'tb/dut'
...
update_power
report_power
report_power_delay_shifted_event_analysis_options
```

Example

Figure 11-3 shows the power reports generated by the `report_power` command when running power analysis with and without delay shifting. Table 11-1 compares total average power with peak power in various power analysis flows. Note that delay-aware activity shifting does not have an impact on the total average power whereas peak power becomes more realistic when the feature is enabled.

Figure 11-3 Power Reports With and Without Delay Shifting

```
##Sample capp report_power report
                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
-------------------------------------------------------------------------------
clock_network           7.134e-04    0.0000    0.0000 7.134e-04 (24.64%)  i
register                4.228e-04 1.095e-04 6.660e-08 5.324e-04 (18.39%)
combinational           5.822e-04 7.939e-04 1.836e-07 1.376e-03 (47.54%)
sequential              5.290e-05 2.200e-04 9.163e-09 2.729e-04 ( 9.43%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)
  Net Switching Power          = 1.123e-03   (38.81%)
  Cell Internal Power          = 1.771e-03   (61.18%)
  Cell Leakage Power           = 2.594e-07   ( 0.01%)
                                 ---------
Total Power                    = 2.895e-03  (100.00%)

X Transition Power             =    0.0000
CAPP Estimated Glitching Power =    0.0000

Peak Power                     = 5.263e-03
Peak Time                      =      9000
```

```
-------------------------------------------------------------------------------
##Sample delay shifted capp report_power report
                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs

clock_network           6.849e-04    0.0000    0.0000 6.849e-04 (24.13%)  i
register                4.225e-04 1.095e-04 6.660e-08 5.320e-04 (18.74%)
combinational           5.554e-04 7.934e-04 1.836e-07 1.349e-03 (47.53%)
sequential              5.272e-05 2.199e-04 9.163e-09 2.726e-04 ( 9.60%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power          = 1.123e-03   (39.55%)
  Cell Internal Power          = 1.715e-03   (60.44%)
  Cell Leakage Power           = 2.594e-07   ( 0.01%)
                                 ---------
Total Power                    = 2.839e-03  (100.00%)

X Transition Power             =    0.0000
CAPP Estimated Glitching Power =    0.0000

Peak Power                     =    0.0906
Peak Time                      =  2724.026
```

Table 11-1 Comparing Power Numbers

Design Name	CAPP total AVG power	CAPP peak power	CAPP delay shifted AVG power	CAPP delay shifted peak power	SDF annotated Gate-Level total AVG power	SDF annotated Gate-Level peak power
Design 1	2.89e-03	5.26e-03	2.83e-03	0.0906	3.670e-03	0.0970

12

Cell Electromigration Analysis

Use the cell electromigration analysis features to query maximum capacitance derived from the current toggle rate on a pin and to identify cells whose pin toggle rates exceed the maximum toggle rate based on the current types (either average, rms or peak) listed in the electromigration table. The cell electromigration analysis feature allows you to identify and resolve possible electromigration issues early in the flow. When the layout data is available, you can perform more accurate electromigration analysis in the IC Compiler II tool, which takes physical considerations into account.

Cell electromigration analysis is supported in both averaged and time-based power analysis modes.

By default, PrimePower extrapolates one extra grid point outside the electromigration table and reports an error message when the maximum toggle rate value is negative. To stop the extrapolation, set the `power_em_disable_extrapolation` variable to `true`.

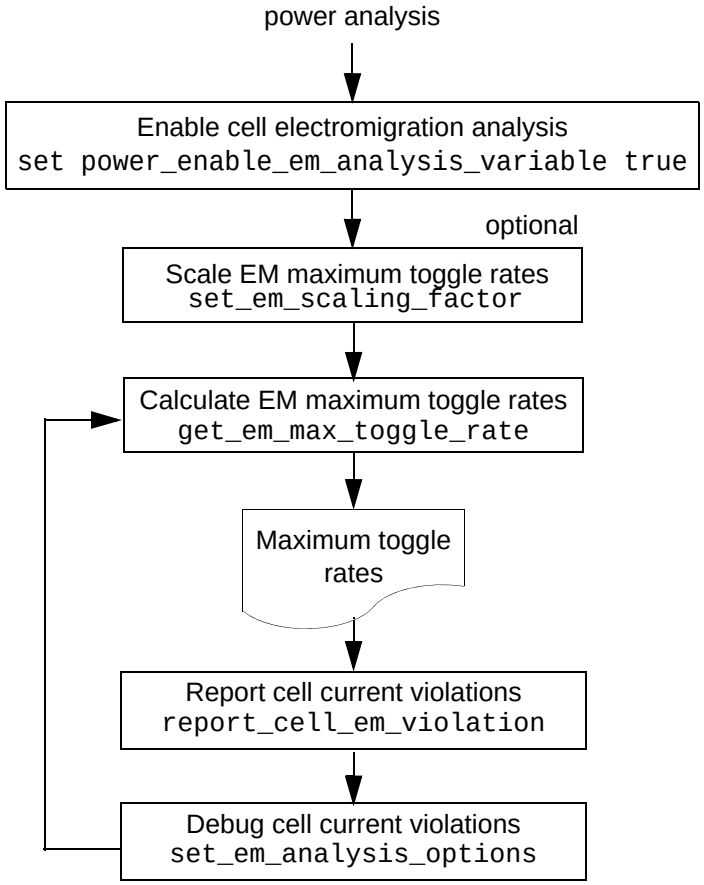
This chapter contains the following topics:

Analysis Flow and Required Input Data

As illustrated in Figure 12-1, PrimePower calculates the electromigration maximum toggle rate based on input transition times, output capacitance values, and switching activities during cell electromigration analysis based on the cell electromigration table in the technology library. PrimePower also allows you to query electromigration maximum capacitance on a pin.

For a detailed description about querying electromigration maximum toggle rates and reporting cell current violations, see Reporting Electromigration Maximum Toggle Rates and Reporting Cell Electromigration Violations.

Figure 12-1 Analysis Flow for Checking Cell Current Violations



Required Input Data

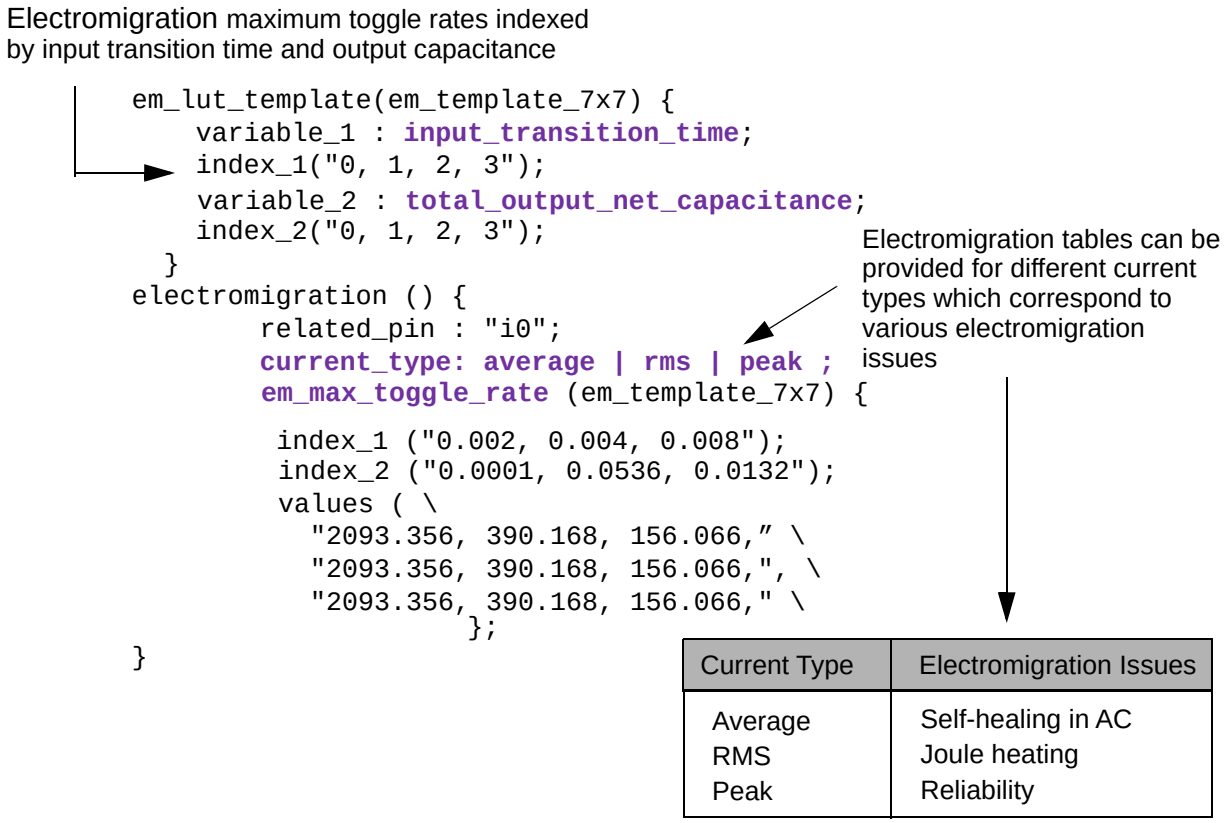
The following data is required to read the library data and to calculate electromigration maximum toggle rates per pin based on the logic library.

- Electromigration library model

  An electromigration table lists the toggle rates indexed by input transition times and output capacitances for different current violation types: average, root mean square (rms), and peak. See Figure 12-2 for an example.

- Transition time
- Capacitance
- Switching activity

Figure 12-2 An Example of the Electromigration Table in the Library Model



Scaling Electromigration Maximum Toggle Rates

When the existing electromigration table is not characterized for the expected PVT conditions, use the `set_em_scaling_factor` command to set a scaling factor on the cell electromigration toggle limit rate per library cell, arc, and current type using the available electromigration library of the cell. The scaled electromigration maximum toggle rate is then used for cell electromigration analysis.

Set the scaling factor value to be a value between 0 and 1. Use the `-current_type` option to specify the current type for toggle rate scaling. The default is RMS.

Example

```
pwr_shell> set_em_scaling_factor mylib/buf1 \
            -input_pin I -output_pin Z -factor 0.1 -current_type rms
```

Reporting Electromigration Maximum Toggle Rates

Run the `get_em_max_toggle_rate` command to calculate and report the electromigration maximum toggle rate for an instance pin, a library cell pin, or an arc. If you specify an I/O pin, PrimePower considers it an input or output, based on the electromigration data in the library relative to the input transition time and output capacitance load. If the transition time and capacitance load data is unavailable, PrimePower treats the pin as an output.

Example

The following example reports the maximum toggle rate for the specified arc without specifying transition time and load.

```
pwr_shell> get_em_max_toggle_rate U1/z -related_pin U1/a
```

The following example reports the maximum toggle rate for a specified arc with the slew rate and lumped load from timing analysis.

```
pwr_shell> get_em_max_toggle_rate myDesign/U1/o \
             –related_pin myDesign/U1/i -verbose
```

The following example reports the maximum toggle rate for the specified arc with user-specified slew and load with the `–verbose` option.

```
pwr_shell> get_em_max_toggle_rate myDesign/U2/i \
             –related_pin myDesign/U2/o \
             –verbose –transition_time 0.1 \
             –output_load 0.2
```

Here are the commonly used options for calculating maximum toggle rates using the `get_em_max_toggle_rate` command:

- (Optional) Use the `-related_pin` option to specify the related pin using the instance pin or pin of the library cell. The pin must be an input pin or I/O pin.
- (Optional) Use the `-output_load` option to specify the output load. The tool uses this data to calculate the maximum toggle rate from the library. Use this option only when you specify the pin as output or I/O pin. You must use this option when you specify the pin of the library cell. The tool uses the total load on the net connected to the driver pin. If the pin is a dangling pin, effective capacitance on the pin is used as load.
- (Optional) Use the `-transition_time` option to specify the transition time which is used to calculate the maximum toggle rate. You must use this option when you specify the input pin of the library cell. If you do not specify the instance pin and transition time, the tool calculates the maximum transition time by averaging the maximum fall and rise transition times at the input pin.
- Use the `-current_type` option to specify the current type from the electromigration table for toggle rate calculation. Different current types correspond to various electromigration issues. For example, use the maximum toggle rate of peak current to identify the reliability issue. Valid values for current types are `avg`, `rms` and `peak`. The default is `rms`.
- Use the `-verbose` option to report the slew rate and capacitance values for the maximum toggle rate calculation. The following error is issued if any electromigration data is missing.

  ```
  Error: No library EM data found for the given pin and2:z. (PEM-018)
  ```

Reporting Electromigration Maximum Capacitance

Use the `get_em_max_capacitance` command to query the electromigration maximum capacitance value on a pin, based on the current toggle rate and the input slew from the output load dependent electromigration table on the pin. Use this query report to fix electromigration violations through PrimeTime engineering change orders (ECOs).

Note:

Before running the `get_em_max_capacitance` command, you need to set the `power_enable_em_analysis` variable to `true` and perform power analysis to make sure the switching activity information on the net objects is up-to-date.

Example

The following example reports the RMS maximum capacitance value from the electromigration table.

```
pwr_shell> get_em_max_capacitance -current_type rms
```

Reporting Cell Electromigration Violations

Use the `report_cell_em_violation` command to check the violations of the specified pins or library cells whose toggle rates exceed the maximum toggle rate.

By default, the command uses both arc-based and non-arc-based data from the electromigration table for calculating toggle rate violations. To avoid using arc-based data for calculation, use the `-exclude_arc` option.

To check the violations by different current types, use the `-current_type` option and specify the current type as average, RMS or peak. The default is RMS.

To report the input slew rate, the load type, the output load value, the capacitance value, the EM scaling factor, and the cell reference name for the selected arc, use the `-verbose` option.

Example

The following example reports the pins with toggle rate violations in the design and the related pins with their associated slew information and capacitance loads. Arc-based tables are ignored during calculation.

```
pwr_shell> report_cell_em_violation -pin [get_pins] -exclude_arc -verbose

Pin Name    Max TR     Max TR    Max TR    Current TR  Diff      Violation  Max. Cap.
            (average)  (rms)     (peak)
-------------------------------------------------------------------------------------
U2/Z        4.338390   1.206270  1.870910  2.000000    -0.793730     V      0.103354
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
Cell Reference Pin   Related  Current Transition Time Load  Load  Load  Slew Scaling
Name           Name  Pin Name Type    Time                 Type  Value Type Factor
-------------------------------------------------------------------------------------
buf1           U1/y  U1/a     AVE     0.000000   0.070000  TOT   MIN   MIN  1.000000
buf1           U1/y  U1/a     RMS     0.000000   0.012000  EFF   MIN   MIN  1.000000
buf1           U1/y  U1/a     PEAK    0.000000   0.070000  TOT   MAX   MAX  0.200000
-------------------------------------------------------------------------------------
```

Printed when the `-verbose` option is specified.

Debugging Cell Electromigration Violations

To set slew values and capacitance load values and types per current type for cell electromigration analysis, use the `set_em_analysis_options` command. Use this feature when you want to rerun cell electromigration analysis with the user-defined slew value or capacitance load value or type for debugging purposes. You can specify the current type as average, RMS or peak. The default is RMS.

Here is the syntax for the `set_em_analysis_options` command:

```
set_em_analysis_options \
 -cload_type [eff/tot] \
 -cload_val [min/max/avg] \
 -in_slew [min/max/avg] \
 -current_type [ave/rms/peak]
```

13

RedHawk Analysis Fusion in IC Compiler II

The RedHawk power integrity solution is integrated with the IC Compiler II implementation flow through the RedHawk Analysis Fusion interface. To perform voltage drop analysis on the power and ground network, you need to provide accurate timing and power consumption data for calculating current and voltage violations.

This chapter describes how to generate the timing window file and the instance power file for RedHawk rail analysis in PrimePower. It includes the following topics:

Generating Timing Window Files

The RedHawk tool requires the static timing analysis information to perform voltage drop or clock jitter analysis. To generate a timing window file (*.twf) in PrimePower, run the `write_rh_file` command when the timing update is complete. The output timing window file is a compressed ASCII file.

Note:

You must set the `power_enable_timing_analysis` variable to `true` before the `write_rh_file` command in the ASCII input flow.

Script Example

The following example runs timing updates and generates a timing window file, called design.twf, in PrimePower.

```
set power_enable_timing_analysis true
set search_path "../dbs/"
set link_path "lib.db"
read_verilog design.v
current_design design
read_parasitics design.spef
read_sdc design.sdc
update_timing
write_rh_file -significant_digits 6 -filetype irdrop -output design.twf
```

Enable timing analysis

Generate a timing window file

Generating Instance Power Files

To generate an instance power file for running RedHawk rail analysis in the IC Compiler II environment, run the `write_rh_power_file` command.

Note:

Your library must contain PG pin definitions.

Before generating an instance power file, you must enable multirail analysis be setting the `power_enable_multi_rail_analysis` to `true`.

The output instance power file contains the following information for power pins:

- Cell instance name
- Pin names
- Total power in watts

To generate an instance power file for RedHawk rail analysis,

1. Enable multirail analysis by setting the `power_enable_multi_rail_analysis` to `true`.
2. Read the library data with the PG pin definitions for the cells.

   For more information, see Reading the Design Data and Logic Library.
3. Specify switching activity information.

   For more information, see Annotating Switching Activity.
4. Calculate power by running the `update_power` command.

   For more information, see Averaged Power Analysis and Time-Based Power Analysis.
5. Write out the instance power file by running the `write_rh_power_file` command.

An Instance Power File Example

The following output file example includes the instance power data per PG pin in the following RedHawk-specific format:

```
#<instance_name> <pin_power_in_Watts> <VDD_pin_name>
U721             5.998e-10            VDD
U721             0                    VBB
U721             2.51e-11             VPP
U457             5.998e-10            VDD
U457             0                    VBB
U457             2.51e-11             VPP
```

14

Generating Reports

PrimePower can generate a wide range of reports that provide information about the power consumption, as described in the following topics:

- Generating Power Reports
- Generating Reports on Threshold Voltage Groups
- Generating Reports on Clock Gating Efficiency
- Generating Reports on Power Derating Factors
- Generating Custom Reports with Attributes
- Querying Internal Power on Pins
- Generating Power Calculation Reports

The `report_power` command is the primary command for generating power dissipation reports. After you have successfully performed power analysis with the `update_power` command, use the `report_power` command to view the results.

Generating Power Reports

By default, the `report_power` command reports the top-level power consumption. You can generate four types of reports:

- Group-based (the default) when the `-cell_power`, `-net_power`, and `-hierarchy` options are not specified (see Example 14-1).

  Power reports for the power groups created by the `create_power_group` command (see Creating Power Groups).

- Cell-based, with the `-cell_power` option.

  Use the `-sort_by` option to sort the report by name, cell internal power, cell leakage power, or dynamic power (see Example 14-4).

- Net-based, with the `-net_power` option.

  Use the `-sort_by` option to sort the report by name, net static probability, net switching power, net toggle rate, relative toggle rate, or total net load (see Example 14-5).

- Hierarchy-based, with the `-hierarchy` option (see Example 14-6).

- Clock-tree-based, with the `-clocks` option (see Example 14-7).

- Power-rail-based, with the `-rails` option (see Example 14-7).

- PST-based, with the `-pst` option.

The `-groups`, `-rails`, and `-clocks` options are filters that apply to all report types. Using these in combination with the `current_instance` command allows you to generate reports per clock domain, power rail, power group, and hierarchy.

Power Group Reports

When none of the `-cell_power`, `-net_power`, or `-hierarchy` option is specified, the `report_power` command classifies the power consumption into categories, referred to as power groups, based on the cell type. There are seven predefined power groups, based on the rules shown in Table 14-1:

Table 14-1 Predefined Power Groups

Power group	Cell type	Identification
`io_pad`	I/O pad cells	attribute: `is_pad_cell`: `true`
`memory`	Memory cells, which are identified in multiple ways	attribute: `is_memory_cell`: `true`

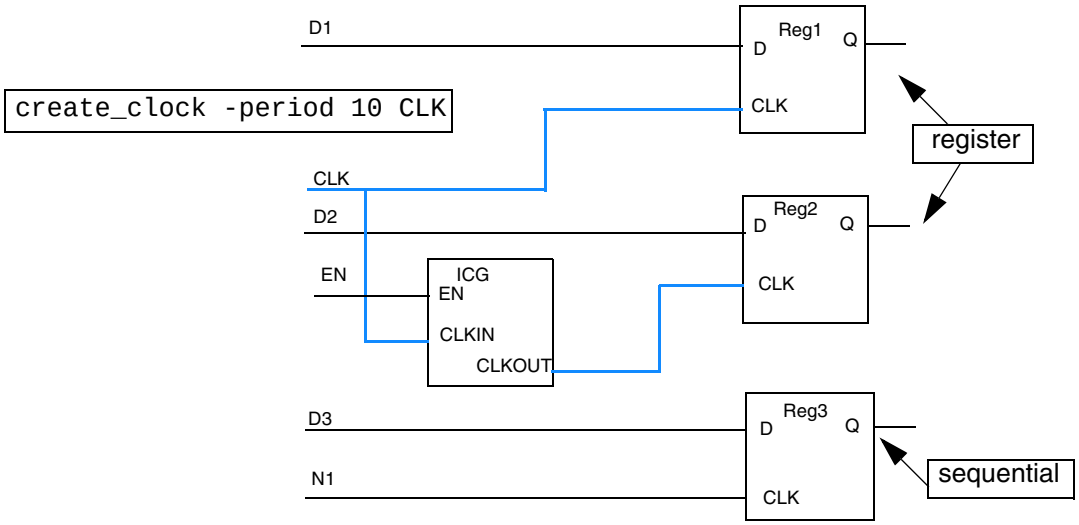
Table 14-1 Predefined Power Groups (Continued)

Power group	Cell type	Identification
`black_box`	Cells with no functionality, excluding `io_pad` and memory cells	attribute: `is_black_box`: `true`
`clock_network`	Cells in the clock tree network, excluding I/O pad cells	The `create_clock` command is used to identify the clock. Cells in the clock network are buffers, inverters and clock gating logic through which the clock is traced.
`register`	Latches and flip-flops, excluding clock gating logic, driven by a clock	Register clock pin power is placed in the `clock_network` group, or the register group, based on the value of the `power_clock_network_include_register_clock_pin_power` variable. The default is `true`. The register clock pin power is reported as `clock_network` power, while the remaining register power is reported as register power. When set to `false`, the clock pin power is included in the register group power.
`sequential`	Latches and flip-flops excluding clock gating logic and not driven by a clock	The clock pin of the sequential cell is not connected to the clock network.
`combinational`	Combinational logic	Any remaining cells not identified as belonging to the other power groups.

When you specify a `group_list` value with the `-groups` option of the `report_power` command, the tool generates a power report for the cells in the specified `power_group`. By default, all the instances are placed in only one power group. Therefore, the summation of the power of the predefined groups equals the total power consumption of the design.

Figure 14-1 highlights the difference between register and sequential power groups. Reg1 and Reg2 are in the register power group because their clock pins are tied to nets which are part of the clock network as defined by the `create_clock` command. Reg3 is in the sequential power group because its clock pin is not part of the clock network.

Figure 14-1 Difference Between Registers and Sequential Power Groups



Example 14-1 shows the power group report without the `-cell_power`, `-net_power`, and `-hierarchy` options specified:

Example 14-1 Power Group Report Without Options Specified

```
pwr_shell> report_power
****************************************
Report: Averaged Power
Design: mac
...
****************************************
  Attributes
      i  -  Including register clock pin internal power
      u  -  User-defined power group
Power           Internal   Switching   Leakage    Total
Group           Power      Power       Power      Power   (     %)  Attrs
--------------------------------------------------------------------------
clock_network   8.529e-04 1.599e-05  4.398e-04   8.689e-04(40.56%)   i
register        1.882e-04 3.482e-05  6.666e-08   2.230e-04(10.41%)
combinational   4.068e-04 5.646e-04  1.836e-07   9.716e-04(45.36%)
sequential      2.327e-05 5.522e-05  9.163e-09   7.851e-05( 3.67%)
memory          0.0000    0.0000     0.0000      0.0000(0.00%)
io_pad          0.0000    0.0000     0.0000      0.0000(0.00%)
black_box       0.0000    0.0000     0.0000      0.0000(0.00%)
--------------------------------------------------------------------------
Net Switching Power  = 6.707e-04 (31.31%)
Cell Internal Power  = 1.471e-03 (68.68%)
Cell Leakage Power   = 2.594e-07 ( 0.01%)
----------------------------------------
Total Power          = 2.142e-03 (100.00%)
----------------------------------------
```

You can specify the `-include_estimated_clock_network` option only when generating the power group reports and provided you have already specified the `estimate_clock_network_power` command.

For more information about how to estimate the additional power incurred by a clock network, see Estimating Clock Network Power Consumption.

Creating Power Groups

You can create your own power groups to classify cells into groups using the `create_power_group` command. Cells in each group need not be mutually exclusive. When defining your own power group, use a unique name.

To remove predefined or user-specified power groups, use the `remove_power_groups` command.

To return a list of cells contained within specified predefined or user-defined power groups, use the `get_power_group_objects` command.

To list the predefined and user-defined power groups, use the `report_power_groups` command.

To report the power information for objects in power groups, use the `report_power -groups` *`power_group_list`* command. This command generates a report that contains power data for objects in the predefined or user-defined power groups.

For more information about these commands, see the man pages.

Overriding the Default Power Group Classification

To override the default power group classification, use the `power_cell_type` attribute to classify cells into different default power groups. You can use the `power_cell_type` attribute in both the averaged and time-based power analysis modes.

You can set the value of this attribute to one of the following default power groups: `clock_network`, `register`, `combinational`, `sequential`, `memory`, `io_pad`, and `black_box`. You cannot use the `power_cell_type` attribute to assign a cell to a user-defined power group.

The `power_cell_type` attribute is a cell-level and a library-cell level attribute. To define this attribute, use the `set_user_attribute` command. For example,

- To assign a cell to the register power group,

  ```
  set_user_attribute [get_cell U0/U1] power_cell_type register
  ```

- To assign a cell instance from a library to the register power group,

  ```
  set_user_attribute [get_lib_cells lib1/REF1] power_cell_type register
  ```

The cell-level attribute definition overrides the library-cell level attribute definition.

The `power_cell_type` attribute value takes precedence over the default group classification. For example, a cell in the black_box power group, can be reclassified as a register, by setting the `power_cell_type` attribute to register. As part of the register group, PrimePower associates any clock pin power of the cell to the clock_network power group.

The resulting power report includes the cell with the `power_cell_type` attribute set to register, as part of the register power instead of black box power.

The following example shows the results of the `report_power` command using the default power group classification, followed by the results of the `report_power` command after the `power_cell_type` attribute is used to move the cells from the black box power group to the register power group.

Example 14-2 shows a power report generated by the `report_power` command when the `power_cell_type` attribute is not set.

Example 14-2 Power Report Before Setting the power_cell_type Attribute

```
pwr_shell> report_power

****************************************************************************
Report: Averaged Power
...
****************************************************************************
Attributes:
-------------------------------------------------
     i - Including register clock pin internal power
     u - User-defined power group

Power Group     Internal   Switching   Leakage    Total
Attrs
                Power      Power       Power      Power(  %)
------------------------------------------------------------------------------
----
clock_network   6.144e-06  1.750e-07   8.774e-08  6.406e-06(70.32%)
register        4.295e-07  4.614e-07   1.152e-06  2.043e-06(22.42%)
combinational   0.0000     0.0000      0.0000     0.0000
sequential      0.0000     0.0000      0.0000     0.0000
memory          0.0000     0.0000      0.0000     0.0000
io_pad          0.0000     0.0000      0.0000     0.0000
black_box       4.017e-07  2.326e-08   2.370e-07  6.620e-07( 7.27%)
------------------------------------------------------------------------------
     Net Switching Power = 6.597e-07 ( 7.24%)
     Cell Internal Power = 6.975e-06 (76.56%)
     Cell Leakage Power  = 1.476e-06 (16.20%)
-------------------------------------------------
     Total Power         = 9.111e-06 (100.00%)
-------------------------------------------------
```

After you have set the attribute, generate a power report, as shown in Example 14-3:

Example 14-3 Power Report After Setting the power_cell_type Attribute

```
pwr_shell> set_user_attribute [get_lib_cells lib1/REF1] power_cell_type register
pwr_shell> report_power
****************************************************************************
Report: Averaged Power
...
****************************************************************************
Attributes
---------------------------------------------------
         i - Including register clock pin internal power
         u - User-defined power group
---------------------------------------------------
Power Group     Internal   Switching   Leakage    Total            Attrs
                Power      Power       Power      Power(  %)
---------------------------------------------------------------------------
clock_network   6.144e-06  1.750e-07   8.774e-08  6.406e-06(70.32%)
register        8.312e-07  4.847e-07   1.389e-06  2.704e-06(29.68%)
combinational   0.0000     0.0000      0.0000     0.0000  (0.00%)
sequential      0.0000     0.0000      0.0000     0.0000  (0.00%)
memory          0.0000     0.0000      0.0000     0.0000  (0.00%)
io_pad          0.0000     0.0000      0.0000     0.0000  (0.00%)
black_box       0.0000     0.0000      0.0000     0.0000  (0.00%)
---------------------------------------------------------------------------

         Net Switching Power = 6.597e-07 ( 7.24%)
         Cell Internal Power = 6.975e-06 (76.56%)
         Cell Leakage Power  = 1.476e-06 (16.20%)
---------------------------------------------------
         Total Power         = 9.111e-06 (100.00%)
---------------------------------------------------
```

The highlighted text in the report example shows the power values shifted from the `black_box` power group to the register power group after setting the `power_cell_type` attribute.

Cell-Based Reports

To generate a cell-based report, specify the `-cell_power` option with the `report_power` command. The cell-based report lists the internal, switching, leakage, and total power per cell. By default, only the cells (including hierarchical cells) in the current hierarchy are displayed.

You can specify a cell list, or the `report_power` filter to report only cells in the specified power group, clock domain, or power rail.

Example 14-4 shows a cell-based power report for all of the cells in the clock network power group:

Example 14-4 An Example of the Cell-Based Power Report for the Predefined Clock Network Power Group

```
pwr_shell> report_power -cell_power -groups clock_network
****************************************
Report : Averaged Power
        -cell_power
        -sort_by cell_internal_power
        -power_greater_than       0
        -groups clock_network
Design : test
...
****************************************
  Attributes
  ----------
      a  -  Annotated internal & leakage power
      b  -  Black box (unresolved) cell
      c  -  Clock pin internal power only
      d  -  Does not include clock pin internal power
      h  -  Hierarchical cell
                Internal  Switching  Leakage    Total
Cell            Power     Power      Power      Power   (     %)  Attrs
------------------------------------------------------------------
sub             3.626e-05 8.397e-06 8.384e-11 4.466e-05 (20.00%)  h
clk_out1_reg    1.228e-05 8.397e-06 8.384e-11 2.068e-05 ( 9.26%)  h
clk_temp1_reg   1.228e-05 8.397e-06 8.384e-11 2.068e-05 ( 9.26%)  h
temp1_reg_3_    5.995e-06    0.0000    0.0000 5.995e-06 ( 2.68%)  c
temp1_reg_0_    5.995e-06    0.0000    0.0000 5.995e-06 ( 2.68%)  c
------------------------------------------------------------------
Totals          1.813e-04 4.199e-05 4.192e-10 2.233e-04 (100.0%)
```

By default, the cells are sorted based on their internal power values. Use the `-sort_by` option to sort cells based on the leakage, switching, or total power.

Net-Based Reports

To generate a net-based report, specify the `-net_power` option with the `report_power` command. The net-based report lists the power rail, total net load, static probability, toggle rate, and switching power per net. By default, only nets in the current hierarchy are displayed, sorted by the net switching power.

Example 14-5 shows a net-based report generated using the `-net_power`, `-leaf`, and `-nworst` options of the `report_power` command. The generated report is sorted by net switching power and filtered to display only the five nets with the highest switching power.

Example 14-5 An Example of the Net-Based Power Report

```
pwr_shell> report_power -net_power -leaf -nworst 5
****************************************
Report : Averaged Power
        -net_power
        -nworst 5
        -leaf
        -sort_by net_switching_power
        -power_greater_than       0
Design : test
...
****************************************
Attributes
----------
      a  -  Switching activity information annotated on net
      p  -  Propagated switching activity information on net
      d  -  Default switching activity used on net
      u  -  Net switching activity uninitialized
      m  -  Net is driven by multiple pins
                          Total      Static   Toggle   Switching
Net                 Vdd   Net Load   Prob.    Rate     Power      Attrs
------------------------------------------------------------
net36               1.80    0.026    0.248    0.1985   8.397e-06  p
net42               1.80    0.026    0.248    0.1985   8.397e-06  p
net48               1.80    0.026    0.248    0.1985   8.397e-06  p
net54               1.80    0.026    0.248    0.1985   8.397e-06  p
sub/net20           1.80    0.026    0.248    0.1985   8.397e-06  p
------------------------------------------------------------
Total (5 nets)                                          4.199e-05 Watt
```

Specifying the `-leaf` option includes leaf instance power dissipation in the report. Reporting to the leaf level is useful specifically when the leaf cells are scattered throughout the hierarchy or when you want to perform detailed analysis.

Hierarchy-Based Reports

To generate a hierarchical report that displays the power consumption per hierarchy, specify the `-hierarchy` option of the `report_power` command. By default, only the first level is shown. Additional levels can be displayed with the `-levels` option.

Example 14-6 shows a hierarchy-based power report for an instance named `mac`:

Example 14-6 Hierarchy-Based Power Report

```
pwr_shell> report_power -hierarchy -levels 2
****************************************
Report : Averaged Power
        -hierarchy
        -levels 2
Design : mac
...
****************************************
                       Switch   Int      Leak      Total
Hierarchy              Power    Power    Power     Power    %
------------------------------------------------------------------
mac                    1.55e-03 2.23e-03 2.59e-07  3.78e-03 100.0
 mult_21               7.28e-04 5.60e-04 1.49e-07  1.29e-03  34.1
   U1/U9720            2.12e-04 1.31e-04 1.60e-08  3.43e-04   9.1
 add_23                3.31e-04 2.54e-04 2.36e-08  5.85e-04  15.5
```

Power Rail Reports

The following example shows power reports for rail VDD1 and VDD2, VDD2 and VDD3, and for all rails.

Example 14-7 Power-Rail-Based Reports

```
pwr_shell> report_power -rails {"VDD1 VDD2" "VDD2 VDD3" "all"}
****************************************
Report : Averaged Power
 ...
****************************************
Current Power Rail: all
Power Report For Rails: VDD1 VDD2
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User-defined power group

               Internal  Switching  Leakage  Total
Power Group    Power     Power      Power    Power    (     %)  Attrs
------------------------------------------------------------------------
io_pad         0.0000    0.0000     0.0000   0.0000   ( 0.00%)
memory         0.0000    0.0000     0.0000   0.0000   ( 0.00%)
black_box      0.0000    0.0000     0.0000   0.0000   ( 0.00%)
```

```
clock_network  0.0000     0.0000     0.0000     0.0000     ( 0.00%)  i
register       0.0000     0.0000     0.0000     0.0000     ( 0.00%)
combinational  4.061e-06  1.800e-07  8.415e-11  4.241e-06  (100.00%)
sequential     0.0000     0.0000     0.0000     0.0000     ( 0.00%)
--------------------------------------------------------------------------
  Net Switching Power  = 1.800e-07   ( 4.24%)
  Cell Internal Power  = 4.061e-06   (95.75%)
  Cell Leakage Power   = 8.415e-11   ( 0.00%)
-----------------------------------------------------------
Total Power            = 4.241e-06  (100.00%)
-----------------------------------------------------------
```

Generating Reports on Threshold Voltage Groups

In PrimePower, you can report information about threshold voltage groups using the `report_threshold_voltage_group` command and the `report_power -threshold_voltage_group` commands.

The `report_threshold_voltage_group` command reports the number and percentage of the cells in each threshold voltage group. When you specify the `-threshold_voltage_group` option with the `report_power` command, the tool reports the leakage power per threshold voltage group. Use the reports to compare the leakage power of your design for various threshold distributions.

Identifying the Threshold Voltage Group

By default, the `report_threshold_voltage_group` command identifies the threshold voltage groups to which the cells can be categorized using the following attributes at the library and cell level:

- Library-level attribute: `default_threshold_voltage_group`
- Cell-level attribute: `threshold_voltage_group`

When PrimePower does not find these attributes, you can set the attributes in the library using the `set_user_attribute` command.

The following example shows the standard and low threshold voltage attribute set on a library.

```
pwr_shell> set_user_attribute [get_libs agl90g_od_svt_ss] \
               default_threshold_voltage_group SVT

pwr_shell> set_user_attribute [get_libs agl90g_od_lvt_ss] \
               default_threshold_voltage_group LVT
```

The following example shows the standard and low threshold voltage attribute set on a cell:

```
pwr_shell> set_user_attribute [get_lib_cells -of_object [get_cells \
    data0_reg[3]]] threshold_voltage_group HVT

pwr_shell> set_user_attribute  \
   [get_lib_cells -of_object [get_cells U25]] threshold_voltage_group SVT
```

When you run the `report_threshold_voltage_group -pattern_priority` command, the threshold voltage groups are identified based on user-defined string patterns. This method is compatible with the PrimeTime tool for ECO cell swapping for leakage recovery.

For more information about ECO cell swapping, see the *PrimeTime User Guide*.

By default, user-defined pattern strings correspond to strings in the cell name. For example, a library cell might be named HVT_BUF1X, where HVT represents the voltage threshold group. When you specify the `-pattern_priority` option with the `-attribute` option, the user-defined string corresponds to a string in the attribute value.

Reporting Cell Count Based on Threshold Voltage Groups

The `report_threshold_voltage_group` command reports the number and percentage of the cells in each threshold voltage group and per leakage power group.

By default, the `report_threshold_voltage_group` command reports the cells per voltage threshold group, based on the cell or library threshold voltage group attributes. With the `-lvth_groups` option, you can specify the threshold voltage groups that are considered low-threshold.

Example 14-8 shows the voltage threshold distribution report generated by the `report_threshold_voltage_group` command with the `-lvth_groups` option to identify the low threshold voltage groups. The groups are assigned the attribute (L).

Example 14-8 Report Showing Cell Distribution Based on Threshold Voltage Groups

```
pwr_shell> report_threshold_voltage_group -lvth_groups {LVT SVT}
------------------------------------------------------------------------
Threshold Voltage Group Report
------------------------------------------------------------------------
Attributes:
-------------
           L: user-defined low Vth group
------------------------------------------------------------------------
Power Group         LVT(L)        SVT         HVT(L)
Name                cell(%)       cell(%)     cell(%)     Attribute
------------------------------------------------------------------------
io_pad              0(  0.00%)    0(0.00%)    0( 0.00%)
memory              0(  0.00%)    0(0.00%)    0( 0.00%)
clock_network       0(  0.00%)    0(0.00%)    0( 0.00%)
black_box           0(  0.00%)    0(0.00%)    0( 0.00%)
register            4(100.00%)    0(0.00%)    0( 0.00%)
combinational      12( 40.00%)   10(33.33%)   8(26.67%)
sequential          0(  0.00%)   4(100.00%)   0( 0.00%)
------------------------------------------------------------------------
Total              16(42.11%)   14(36.84%)    8(21.05%)
------------------------------------------------------------------------
SUMMARY:
          Low Vth cell (%) = 24(63.16%)
         Other Vth cell(%) = 14(36.84%)
     Undefined Vth cell(%) =  0( 0.00%)
     ----------------------------------
          Total Cell (%)   = 38(100.00%)
     ----------------------------------
------------------------------------------------------------------------
```

When you use the `-pattern_priority` option of the `report_threshold_voltage_group` command, specify the pattern strings in the reference cell name to indicate the threshold voltage group to which the cells belong. Additionally, the `-lvth_groups` option can be used to identify the low threshold voltage groups.

Example 14-9 shows the threshold voltage group distribution based on the reference name string pattern. The `-lvth_groups` option indicates the low threshold voltage groups.

Example 14-9 Report Showing Cell Distribution Based on Threshold Voltage Groups Identified by the Cell Name Pattern Priority

```
pwr_shell> report_threshold_voltage_group -pattern_priority {HVT LVT NVT SVT} \
         -lvth_groups {LVT NVT}
------------------------------------------------------------------------
              Threshold Voltage Group Report
------------------------------------------------------------------------
Attributes:
         u -> user-defined power group
         L -> user-defined low Vth group

Power Group        HVT        LVT(L)     NVT(L)     SVT
Name               cell(%)    cell(%)    cell(%)    cell(%)        Attribute
```

```
-------------------------------------------------------------------------------
io_pad             0 (0.00%)  0 (0.00%)  0 (0.00%)   0 (0.00%)
memory             0 (0.00%)  0 (0.00%)  0 (0.00%)   0 (0.00%)
clock_network      0 (0.00%)  0 (0.00%) 0 (0.00%)   0 (0.00%)
black_box          0 (0.00%)  0 (0.00%)  0 (0.00%)   0 (0.00%)
register           0 (0.00%)  0 (0.00%)  0 (0.00%)   3 (75.00%)
combinational      1 (3.33%) 13 (43.33%) 0 (0.00%)  16 (53.33%)
sequential         0 (0.00%)  4 (0.00%)  0 (0.00%)   0 (0.00%)
-------------------------------------------------------------------------------
Total              1 (2.63%) 17(44.74%)  1 (2.63%)  19 (50.00%)
-------------------------------------------------------------------------------
SUMMARY :
               Low Vth cell (%)        = 18 (47.37%)
               Other Vth cells (%)     = 20 (52.63%)
               Undefined Vth cells (%) =  0 (0.00%)
-----------------------------------------------------------------
               Total cells (%)         = 38 (100.00%)
-----------------------------------------------------------------
```

Reporting Leakage Power Based on Threshold Voltage Group

To report the leakage power per threshold voltage group, specify the `-threshold_voltage_group` option of the `report_power` command. The threshold voltage groups, by default, again are identified by the cell and library attributes. By using the `-pattern_priority` option, you can classify the threshold voltage groups based on the specified reference name strings.

You can filter the leakage power data using the `-clocks`, `-groups`, and `-rails` options. If you specify the `-lvth_groups` option, the tool reports separate data for low threshold voltage groups, as shown in Example 14-10.

Example 14-10 Report Generated by the report_power -threshold_voltage_group -lvth_groups Command

```
pwr_shell> report_power -threshold_voltage_group -lvth_groups {LVT SVT}
-------------------------------------------------------------------------------------------
Power Group   HVT              LVT(L)           SVT(L)              DEFAULT_VTH_GROUP
Name          leakage(%)       leakage(%)       leakage(%)          leakage(%)    Attr
-------------------------------------------------------------------------------------------
memory        0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)
io_pad        0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)
black_box     0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)
clock_network 0.0000(0.00%)     0.0000(0.00%)     4.398e-11(100.00%)0.0000(0.00%)
combinational 0.0000(0.00%)     1.084e-07(59.03%) 1.913e-09(1.04%)  7.333e-08(39.93%)
register      4.084e-08(61.32%) 0.0000(0.00%)     0.0000(0.00%)     2.576e-08(38.68%)
sequential    0.0000(0.00%)     0.0000(0.00%)     0.0000(0.00%)     9.163e-09(100.00%)
-------------------------------------------------------------------------------------------
Total         4.084e-08(61.32%) 1.084e-07(59.03%) 1.957e-09(0.75%)  1.083e-07(41.72%)
-------------------------------------------------------------------------------------------
SUMMARY:
          Low Vth leakage (%) = 1.104e-07(42.53%)
          Other Vth leakage(%)= 4.084e-08(15.74%)
     Undefined Vth leakage(%) = 1.083e-07(41.72%)
     ----------------------------------------
            Total Leakage (%) = 2.594e-07(100.00%)
     ----------------------------------------
```

Generating Reports on Clock Gating Efficiency

Use the `report_clock_gate_savings` command to generate reports that contain metrics for measuring clock gating efficiency (CGE) in the design. You can use these reports to locate inefficient clock gates and to identify sequential cells with further clock gating possibility. You can use this command for both averaged and time-based power analysis modes.

Annotate the switching activity to the design before using the `report_power` command, because the efficiency metrics are based on the toggle rates. In the time-based mode, you can apply switching activity using the `read_vcd` command followed by the `update_power` command. In the averaged mode, you can apply switching activity using the `read_saif`, `read_vcd`, or `set_switching_activity` command. If the design is not fully annotated, the `report_clock_gate_savings` command initiates power analysis to propagate switching activity.

The accuracy of the analysis report depends on the quality of the switching activity that you apply on the design. The switching activity must represent the typical behavior of the design, which covers all operation modes of the design. Simulation vectors are the source of quality switching activity that can be used for power analysis.

Generating Clock Gating Reports

Equation 14-1 is the basic metric for measuring clock gate toggle savings. It calculates the ratio of input clock toggles to the clock gates that are not propagated to the clock gate output, as shown in the following equation:

Equation 14-1 Metric for Calculating Clock Gate Toggle Saving

$$\text{ToggleSavings} = 1 - \left(\frac{\text{Tr}_{\text{clkout}}}{\text{Tr}_{\text{clkin}}}\right)$$

Clock gates with high toggle savings are those that disable a large percentage of input clock toggles from propagating to the output. The default report generated by the `report_clock_gate_savings` command provides an overview of the clock gating efficiency of the design, as shown in the following example:

```
pwr_shell> report_clock_gate_savings
****************************************
Report : Clock Gate Savings
   power_mode: Averaged
Design : test
Version: <Version>
****************************************
--------------------------------------------------------------------------------
Clock: sys_clock_i
  + Clock Toggle Rate: 0.2
  + Number of Registers: 1883
      + Number of Un-gated Registers: 67 (3.6 %)
      + Number of Gated Registers: 1816 (96.4 %)
  + Number of Clock Gates: 66
  + Max Number of Clock Gate Stage: 2
  + Average Clock Toggle Rate at Registers: 0.113322
  + Average Register Gating Efficiency (savings with respect to root
    clock): 43.3%

--------------------------------------------------------------------------
Toggle Savings                  Number of       % of
Distribution                    Registers       Registers
--------------------------------------------------------------------------
100%                               0             0.0%
80% - 100%                        63             3.3%
60% - 80%                        352            18.7%
40% - 60%                        941            50.0%
20% - 40%                          0             0.0%
0% - 20%                         460            24.4%
0%                                67             3.6%
--------------------------------------------------------------------------
```

The report includes a histogram that shows the average clock gate toggle savings per clock domain.

For each clock, the report also includes the percentage of gated versus ungated registers. If most registers are ungated, you can continue to perform clock gating and power savings. The maximum number of clock gate stages identifies multistage gating. The average register-gating efficiency is a further indicator of the clock gating efficiency.

Equation 14-2 is the metric to measure the ratio of the register clock toggle rate to the root clock, as shown in the following equation:

Equation 14-2 Metric for Calculating Clock Gate Toggle Saving

$$\text{RegisterEfficiency} = 1 - \left(\frac{\text{Tr}_{\text{regout}}}{\text{Tr}_{\text{regin}}}\right)$$

The lower the frequency of the register clock, the higher the register efficiency. The register efficiency takes into consideration the toggle savings throughout the multistage clock gating.

Generating the Clock Gating Report by Clock Gates

To view the contribution of each block to the total toggle savings, use the `-by_clock_gate` option with the `report_clock_gate_savings` command to specify the clock gating stage.

The clock gating stage for a register or clock stage represents the number of clock gates on the clock path, starting with the register clock pin. The rules to assign the clock stage for a clock gate or a register are:

1. If there is no clock gate driving the register, the clock stage is 0.
2. If there is a clock gate driving the register clock pin and the clock gate does not drive additional clock gates, the clock stage is 1.

Equation 14-3 is the clock gating efficiency metric for analyzing the toggle saving contribution per clock gate. This metric takes into consideration the number of registers driven by the clock gate, as well as the toggle savings, to rate the clock gate efficiency per clock domain.

Equation 14-3 Metric for Calculating Clock Gate Toggle Saving

$$\text{CGE}_G = (\text{N}_G/\text{N}) \times \text{TS}_G \times (\text{IN}_G/\text{ROOT}_G)$$

where

CGE_G = Clock gating efficiency of clock gate G

N = Total number of registers of the clock tree

N_G = Number of registers in fanout of clock gate G including transitive fanout

IN_G = Toggle rate at input clock of clock gate G

$ROOT_G$ = Toggle rate at the root clock of clock gate G

The clock gating efficiency for high-stage clock gates is expected to be greater than that of the lower stage clock gates, because their toggle savings contribute to reduced toggles on all the registers driven by all the clock gates they drive, as shown in Example 14-11:

Example 14-11 Report Generated by the report_clock_gate_savings -by_clock_gate Command

```
pwr_shell> report_clock_gate_savings -by_clock_gate
****************************************************************************
Report: report_clock_gate_savings
        -by_clock_gate
...
****************************************************************************
----------------------------------------------------------------------------
Clock Gate Structure Summary
----------------------------------------------------------------------------
Clock       Total       CG Stage   # of Clock    # of Gated  Clock Gating
            Registers               Gates         Cells      Efficiency
----------------------------------------------------------------------------
CLK         12           0          0              0         N/A
                         1          2              8         30.61%
                         2          1              4         54.52%
----------------------------------------------------------------------------

----------------------------------------------------------------------------
Clock Gate Structure Details
----------------------------------------------------------------------------
Clock Clock  Gating     # of   # of   # of     Toggle  Clock     Receiver
      Gate   Element    Fanout Clock Registers Savings Gating     Clock
      Stage                    Gates           (%)     Efficiency Gates
----------------------------------------------------------------------------
CLK   1     clk_gate_G1 4       0        4      81.8  27.3%
      1     clk_gate_G3 4       0        4      55.2   3.4%
      2     clk_gate_G2 5       1        4      81.8  54.5%   clk_gate_G3
----------------------------------------------------------------------------
```

In Example 14-11, the Clock Gate Structure Summary section identifies the clock gating stages, the number of clock gates per stage, the number of cells driven by these clock gates, and the average clock gating efficiency of each stage. The Clock Gate Structure Details section reports the fanout, toggle savings, clock stage, and clock gating efficiency for each clock-gate cell.

For more information about computation of clock gating efficiency, see Clock Gating Efficiency Computation.

Comparing Clock Gating Efficiency of Different Clock Domains

The power consumption of each clock domain is a function of the clock toggle rate in that domain. The clock gating efficiency is a normalized value with respect to the clock toggle rate and the number of registers in the domain. To compare the clock gating efficiency values between two clock domains, multiply both the clock frequency and the number of registers in that clock domain before comparison.

Generating the Sequential Cell Report

To identify registers that can be further gated, use the `-sequential` option with the `report_clock_gate_savings` command. The generated report includes the Q-to-clock ratio:

$\texttt{Q-to-clock ratio} = Tr_Q/Tr_{clk}$

Registers with a low Q-to-clock ratio might be the candidates for XOR self-gating. If the data rarely changes, there is no need to continually clock the register. For register banks with low Q-to-clock ratios, power savings might be possible by using the XOR of the data to enable the clock gate. The clock is enabled only when the data changes.

Additional Reporting Options

Use the `-sequential` option with the `-by_clock_gate` option to report the toggle savings on the register clusters in the design. A register cluster is a set of registers with clock pins driven by the same clock gating cell. This report can help you identify register clusters that can be repartitioned. For instance, if some registers in the cluster have different Q-to-clock ratios than the other registers in the cluster, you can partition the registers into different clusters with a different enable condition for clock gating.

The report generated by using the `report_clock_gate_savings` command with the `-by_clock_gate` and `-sequential` options includes the clock gating stages of the respective clock gates, as shown in Example 14-12.

Example 14-12 Report Generated by the report_clock_gate_savings -sequential -by_clock_gate Command

```
pwr_shell> report_clock_gate_savings –sequential –by_clock_gate
****************************************
Report : Clock Gate Savings
power_mode: Averaged
Design : test
****************************************
--------------------------------------------------------------------------
Clock Gate     Clock Gating    IN clk       OUT clk         Toggle
               Stage           Toggle Rate  Toggle Rate     Savings (%)
--------------------------------------------------------------------------
Register
Q              Q/Clk
Toggle         Toggle
```

```
Rate          Ratio
-------------------------------------------------------------------------
CG_CLK2_20/latch     1         0.1          0.02472          75.3%
   d0_r_reg[0]
0.00298462    12.1%
   d0_r_reg[1]
0.00282796    11.4%
CG_CLK1_10/latch     1         0.2          0.04883          75.6%
   d0_r_reg[2]
0.00385335    7.9%
   d0_r_reg[3]
0.00357056    7.3%
   d0_r_reg[4]
0.00380249    7.8%
   d0_r_reg[5]
0.00379639    7.8%
   d0_r_reg[6]
0.00373332    7.6%
   d0_r_reg[7]
0.00374349    7.7%
**Ungated**          0          NA           NA               NA
   d1_r_reg[1]
0.00302734    3.0%
   d1_r_reg[2]
0.00285645    2.9%
   d1_r_reg[0]
0.00367228    1.8%
-------------------------------------------------------------------------
```

To isolate specific parts in the design, use the `-hierarchical` option in combination with the `-by_clock_gate` or `-sequential` option.

To report the design-related information and control the format of the report, use the following options:

- Design-related options
  - `-clocks`: Reports the objects in the specified clock domains
  - *`object_list`*: Reports the hierarchical blocks, individual registers, or clock gating cells in the design
- Reporting format-related options
  - `-sort_by`: Sorts the order of objects in the report
  - `-nosplit`: Prevent splitting the report over multiple lines

For more information about the `report_clock_gate_savings` command, see the man page.

Generating Reports on Power Derating Factors

To report the power derating factors set on rails or PG pins, use the `report_power_derate` command.

If the derating value is applied to a rail, the command lists the rail as an object type to which the value is applied when set at the design level or on hierarchical cells. If the value is applied to a leaf cell, the object type is the PG pin to which the rail is connected.

Example 14-13 shows the power derating factor set on various objects and power components:

Example 14-13 Reports Generated by the report_power_derate Command

```
pwr_shell> set_power_derate -rails  {V}  2.0 [get_cell  A]
pwr_shell> report_power_derate

Report : power derate
Design :
...
************************************************************
Object Name   Object Type   Switching  Internal  Leakage
------------------------------------------------------------
A             cell(hier)        2.0       2.0      2.0
A/V1          Rail              2.0       2.0      2.0
A             cell(leaf)        2.0       2.0      2.0
A/V1          Rail              2.0       2.0      2.0
------------------------------------------------------------


pwr_shell> set_power_derate -pg_pin PG1 4.0 {A1 A2}
pwr_shell> report_power_derate

Report : power derate
Design :
...
************************************************************
Object Name   Object Type   Switching  Internal  Leakage
------------------------------------------------------------
A1/PG1        PgPin             4.0       4.0      4.0
A2/PG1        PgPin             4.0       4.0      4.0
------------------------------------------------------------
```

Generating Custom Reports with Attributes

PrimePower provides attributes that reference power information within the design and the libraries. Table 14-2 shows each attribute and its object type. These attributes are accessible only after you run the `update_power` command.

Table 14-2 Object Types for the Supported Attributes

Class	Attribute

net	`activity_source`
	`glitch_rate`
	`internal_power_derate_factor`
	`is_power_control_signal_net`
	`leakage_power_derate_factor`
	`power_base_clock`
	`static_probability`
	`switch_power_derate_factor`
	`switching_power`
	`toggle_rate`
	`toggle_count`

pin	`glitch_rate`
	`internal_power`
	`internal_power_derate_factor`
	`leakage_power_derate_factor`
	`power_base_clock (port also)`
	`static_probability`
	`switch_power_derate_factortoggle_rate`

lib_cell	`has_multi_power_rails`
	`has_multi_ground_rails`
	`has_rail_specific_power_tables`
	`is_memory_cell`
	`is_pad_cell`
	`is_black_box`

Table 14-2 Object Types for the Supported Attributes (Continued)

Class	Attribute
cell	dynamic_power
	glitch_power
	internal_power
	leakage_power
	peak_power
	switching_power
	total_power
	x_transition_power
	power_states
	peak_power_start_time
	peak_power_end_time
	has_multi_power_rails
	has_multi_ground_rails
	has_rail_specific_power_tables
	intrinsic_leakage_power
	gate_leakage_power
	internal_power_derate_factor
	switch_power_derate_factor
	leakage_power_derate_factor

Table 14-2 Object Types for the Supported Attributes (Continued)

Class	Attribute
design	dynamic_power
	glitch_power
	internal_power
	leakage_power
	peak_power
	switching_power
	total_power
	x_transition_power
	power_states
	peak_power_start_time
	peak_power_end_time
	power_simulation_time
	intrinsic_leakage_power
	gate_leakage_power
	internal_power_derate_factor
	switch_power_derate_factor
	leakage_power_derate_factor

You can write your own procedures and use the object collections to access the information.

The script example shown in Example 14-14 generates a custom report for power consumption of all registers in the design:

Example 14-14 Script for Generating Custom Report for Power Consumption of All Registers

```
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 total_power]
      set dynamic [get_attribute $cell dynamic_power]
      set leakage [get_attribute $cell leakage_power  ]
      set switching [get_attribute $cell switching_power]
      set internal [get_attribute $cell 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]
  }
}
```

Querying Internal Power on Pins

When power analysis is complete, use the `get_attribute` command to query the `internal_power` attribute on pin objects for debugging purposes.

PrimePower reports a valid power value for pins that have internal power arcs defined. In case of a cell pin being associated with different PG pins that have different internal power, querying the `internal_power` attribute shows the sum of the `internal_power` from each of the internal power arcs on the pins.

Note:

Before querying internal power on pins, you must enable the `power_enable_pin_internal_power_query` variable before the power update.

Script Example

```
set power_enable_analysis TRUE
set power_analysis_mode averaged
set_app_var power_enable_pin_internal_power_query true

set search_path         "../src/hdl/gate ../src/lib/snps . "
set link_library " * core_typ.db"

read_verilog mac.vg
current_designmac
link

check_timing
update_timing

read_saif "../sim/mac.saif" -strip_path "tb/macinst"

update_power
report_power

get_attribute [get_pin pin1] internal_power

# check if new attribute is listed
# list_attributes -application -class pin
```

Enable the feature to query internal power on pins

Use the `get_attribute` command to query internal power and set the `internal_power` application attribute on pin objects

Generating Power Calculation Reports

Before generating power calculation reports using the `report_power_calculation` command, you must read the logic library file into the system with the `library_features` attribute.

When power analysis results do not match the expectation, you run the `report_power_calculation` command for more details on the power calculation. The command is supported in both averaged and time-based analysis modes. For example, in the time-based mode, the command outputs per-event details and a summary report containing the toggles per arc and associated power.

To generate a power-net-based power report for multivoltage designs, use the `set_current_power_net` command to set the power net of interest before you run the `report_power_calculation` command. Only the power dissipated on the specified power nets is calculated and reported. By default, all the power nets are included in the power report.

To check the current power net setting, use the `get_current_power_net` command.

Averaged Power Analysis

In averaged power mode, use the `-state_condition`, `-rise`, `-fall` and `-path_sources` options to display the results of path-dependent the rise/fall internal power calculation. Use the output (see Example 14-15) to check for the estimated state-dependent and path-dependent toggle rates per power arc, and the capacitance and transition time values used to index the energy values power calculation.

Time-Based Power Analysis

In the time-based mode, use the `-time` option to display power calculation results for the specified time windows and cells. For example, use the output results to debug the power waveforms generated from annotation of zero-delay simulation activity files to examine the event order which caused the selection of specific state conditions.

Because all nets are annotated during gate-level power analysis, the `report_power_calculation` command re-analyzes only the selected cells to generate debugging data without overwriting the power data. You can access the original power data for additional debugging tasks when needed.

Use the per-event report to evaluate the event propagation performed by the cycle-accurate analysis.engine. The tool runs propagation on the full design but generates debug data and power computation only for the selected cells. The power data of the complete design from the original analysis is maintained for later retrieval.

Table 14-3 lists the commonly used options of the `report_power_calculation` command when generating power reports. For a detailed description about each option, see man pages.

Table 14-3 The Commonly Used Options of the report_power_calculation Command

Option	Averaged Mode	Time-Based Mode	Description
object_list	Yes	Yes	Specifies the objects for which the power calculation is reported.
`-fall`	Yes	No	Specifies that the internal power report should be limited to fall transition only.
`-state_condition`	Yes	No	Specifies that the report should be restricted to tables or polynomials with matching state condition.
`-rise`	Yes	No	Specifies that the internal power report should be limited to rise transition only.

Table 14-3 The Commonly Used Options of the report_power_calculation Command

Option	Averaged Mode	Time-Based Mode	Description
`-path_sources`	Yes	No	Specifies the related input pin that causes the output pin transition.
`-time`	No	Yes	Specifies the time windows for reporting the power calculation results. When not specified, the tool assumes the specified cells are to be monitored for the entire analysis window.
`-report_output`	Yes	Yes	Specifies the prefix of the report files generated. The default prefix is ptpx. The per-event debug output file name appends _debug.rpt and _summary.rpt to the specified prefix. By default, the per-event debug report file name generated is: ptpx_debug.rpt. and ptpx_summary.rpt.
`-report_type`	Yes	Yes	`-report_type detailed | summary | both` Specifies the type of report to be generated. By default both detailed and summary reports are generated (see Example 14-16).

Example 14-15 calculates the path-dependent internal power from the input pin B to the output pin Y. The contributions for the rise and fall of internal power are shown separately.

Example 14-15 An Example for Calculating Path-Dependent Internal Power

```
pwr_shell> report_power_calculation [get_pin U4/Y] \
             -state_condition default -path_sources B

****************************************
Report : power_calculation
 U4/Y
 -state_condition default
 -path_sources B
Design : abc
Version: <Version>
Date   : <Date>
****************************************
Library(s) Used:
example (File: /root/libraries/example.db)
Operating Conditions: typical
Library: example
Wire Load Model Mode: segmented

Design        Wire Load Model            Library
```

```
-----------------------------------------------
abc                    a                 example

Global Operating Voltage = 0.9
Library Power-specific unit information :
Voltage Unit : 1 V
Capacitance Unit : 1 pF
Time Unit : 1 ns
Dynamic Power Unit : 0.001 W    (derived from V,C,T units)
Leakage Power Unit : 1e-09 W
Note: Unless specified, the numbers reported are in library units.

====================================================================
Pin Internal Power Calculation
   cell: U4
   pin: Y
   path source: B

Path-Dependent Rise Pin Internal Power = 2.35214e-05 W
Pin Internal Power = Internal Energy * Pin Toggle Rate
Rise Internal Energy Per Transition = 0.143423

library model: NLPM
table variables:
    X = total_output_net_capacitance = 0.4
    Y = input_net_transition = 0.3
    relevant portion of lookup table:
                     (X)  0.1050      (X)  0.3550
    (Y)  0.0500      (Z)  0.1310      (Z)  0.1410
    (Y)  0.4510      (Z)  0.1320      (Z)  0.1420
     Z = A + B*X + C*Y + D*X*Y
     A =  0.1267           B =  0.0400
     C =  0.0025           D =  0.0000
     Z = 0.143423
(No scaling since the library has no internal power k-factors)

Path-Dependent Rise Pin Toggle Rate = 0.164
Path-Dependent Fall Pin Internal Power = 2.35214e-05 W
Pin Internal Power = Internal Energy * Pin Toggle Rate
Fall Internal Energy Per Transition = 0.143423
```

Example 14-16 A Summary Report Example

```
Summary Report
Instance: test_reg[32]
Reference cell: test/fdge
Cycle Duration: XX (only if running CAPP; in gate-level time_based mode,

the I/O path delay is reported per SDPD arc)
CLK (R) to Q (R) 0.286289
CLK (R) to Q (F) 0.197336
Pins:
CLK
```

```
Transition times:
(R) 0.003
(F) 0.002
Capacitance:
0.0013
...
Q ( R )
Path source: CLK
Rise Pin Toggle Rate: .05
Rise Pin Glitch Toggle Rate: 0
Fall Pin Internal Energy Value: .023
Fall Pin Internal Power: 4.5e-6

Q (F)
Path source: : CLK
Fall Pin Toggle Rate: .05
Fall Pin Glitch Toggle Rate: 0
Fall internal energy per transition: .54
Fall Pin Internal Power: 2.5e-6
Total Cell Internal Power: 9.43e-6
Switching Power:
Pins:

Q:
Pin Toggle Rate : 0.2
Pin Glitch Toggle Rate: 0
Total_output_net_capacitance: 0.0017
Switching Power: 8.43e-06

Leakage power
State condition : !D &!Q
Leakage Power Value: 2.3e-09
State Probability: .4
State condition: D & Q
Leakage power Value: 2.4e-09
State Probability : .6
Total Leakage Power : 4.7e-09
```

Example 14-17 A Debugging Report Example

```
Time : 6
Cell : z_reg[32]
Leakage :
State Condition : !D & !Q
Leakage Power : 2.52624e-09
Internal :
Pin : CLK Rise
State condition : !D & !Q + D&Q
Rising Internal Energy : 2.80279e-05
Transition  : 0.003

Time : 12
Cell : z_reg[32]
```

```
Internal :
Pin : CLK Fall
State condition : !D & !Q + D&Q
Falling Internal Energy : 3.2533e-05
Transition : 0.002

Time : 30.22
Cell : z_reg[32]
Internal :
Pin : Q Rise
Path source: CLK
State condition : default

Rising Internal Energy : 7.47514e-05
Arc delay : 0.286289
Switching :
Pin : Q Rise
Switching Energy : 3.41159e-05
….
Time : 366
Cell : z_reg[32]
Internal :
Pin : CLK Rise
State condition : !D & !Q + D&Q
Rising Internal Energy : 2.80279e-05
Arc delay :  …
Internal :
Pin : D Rise
State condition : default
Rising Internal Energy : 8.5566e-06
Transition : …
Leakage :
State Condition : D & Q
Leakage Power : 2.3334e-09
```

A

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. This chapter contains the following topics:

Opening and Closing the GUI

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

To start a PrimePower session and open the GUI, enter the following command:

```
% pwr_shell -gui
```

You can optionally set the DISPLAY environment variable when you invoke the PrimePower GUI. For example,

```
%> pwr_shell -gui -display 192.180.50.155:0.0
```

To open the GUI during a PrimePower session, enter the `gui_start` command at the pwr_shell prompt:

```
pwr_shell> gui_start
```

PrimePower automatically checks out a PrimePower license.

To start the tool in the pwr_shell command-line interface, enter the `pwr_shell` command in a UNIX or Linux prompt:

```
% pwr_shell
                                  PrimePower

Version O-2018.06 for linux64 - June 13, 2018
Copyright (c) 2018 Synopsys, Inc.
This software and the associated documentation are proprietary ...
% pwr_shell>
```

To close the GUI without exiting pwr_shell, choose File > Close GUI from the menu or enter the `gui_stop` command on the console. Your design remains loaded in the memory and the command-line prompt remains active in the shell.

To close the GUI and end the PrimePower session, choose File > Exit or enter `exit` on the console.

Analysis Flow With the GUI

The power analysis driver in the GUI helps 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 have read, linked, and calculated the power of the design.

Typically, an analysis session consists of the following tasks:

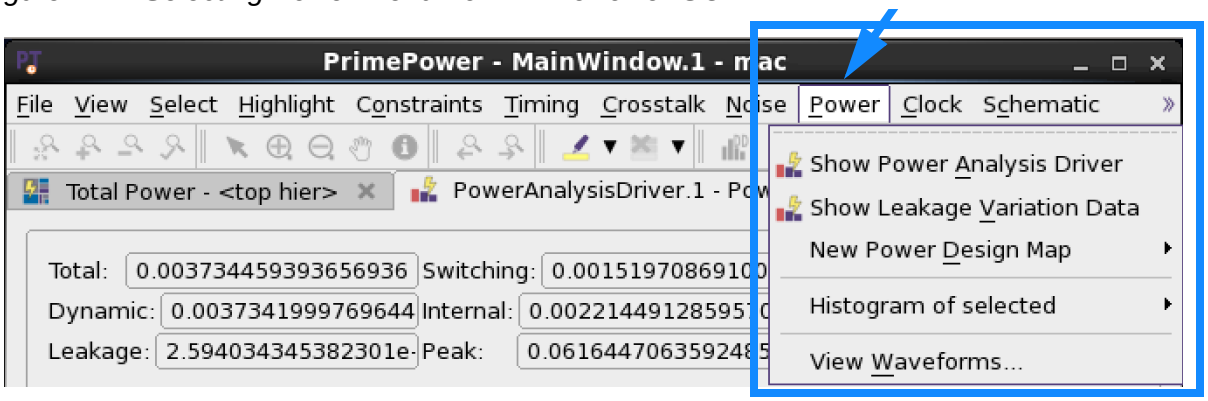
1. Read and calculate the power for the design.

2. Start the GUI.
3. For a high-level overview of the design's power consumption and to identify specific items that contribute to the highest power (such as high activity and capacitance), generate a power histogram.
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.

Using the GUI

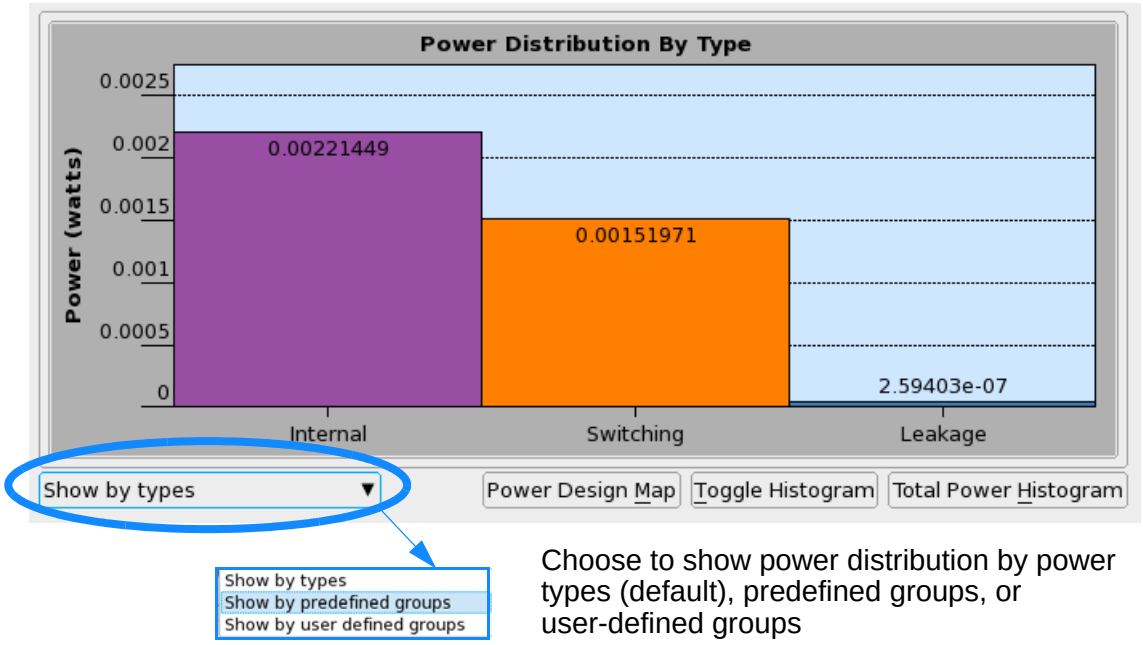
After you complete a power analysis, you can view and analyze the calculated power data in the GUI by choosing Power > Show Power Analysis Driver, which is as shown in Figure A-1.

Figure A-1 Selecting Power Menu from PrimePower GUI



Show Power Analysis Driver. The power analysis driver allows you to easily analyze dynamic power and static power. As shown in Figure A-2, the GUI opens with the analysis driver window on the right to a view of the total power consumption for the current design with its distribution among dynamic, switching, internal, and leakage consumption. The list box in the lower-left corner (circled) allows you to show the power distribution by power types (the default), by predefined groups, or by user-defined groups.

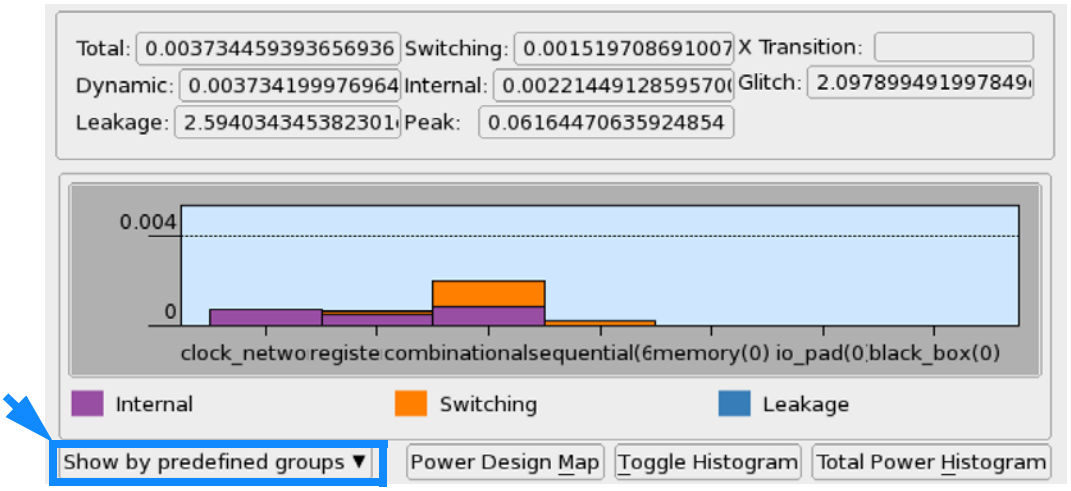
Figure A-2 Analysis Driver Window



Choose to show power distribution by power types (default), predefined groups, or user-defined groups

Figure A-3 shows the power distribution for predefined groups. The numbers inside parentheses indicate the number of cells in the associated groups.

Figure A-3 Power Distribution of Predefined Power Groups



Show Leakage Variation Data. Choose to view the leakage variation data.

New Power Design Map. View the total power density as a map of the design. The power density displays as a tree map, which is a visualization tool that can help you identify anomalies in large hierarchical data sets.

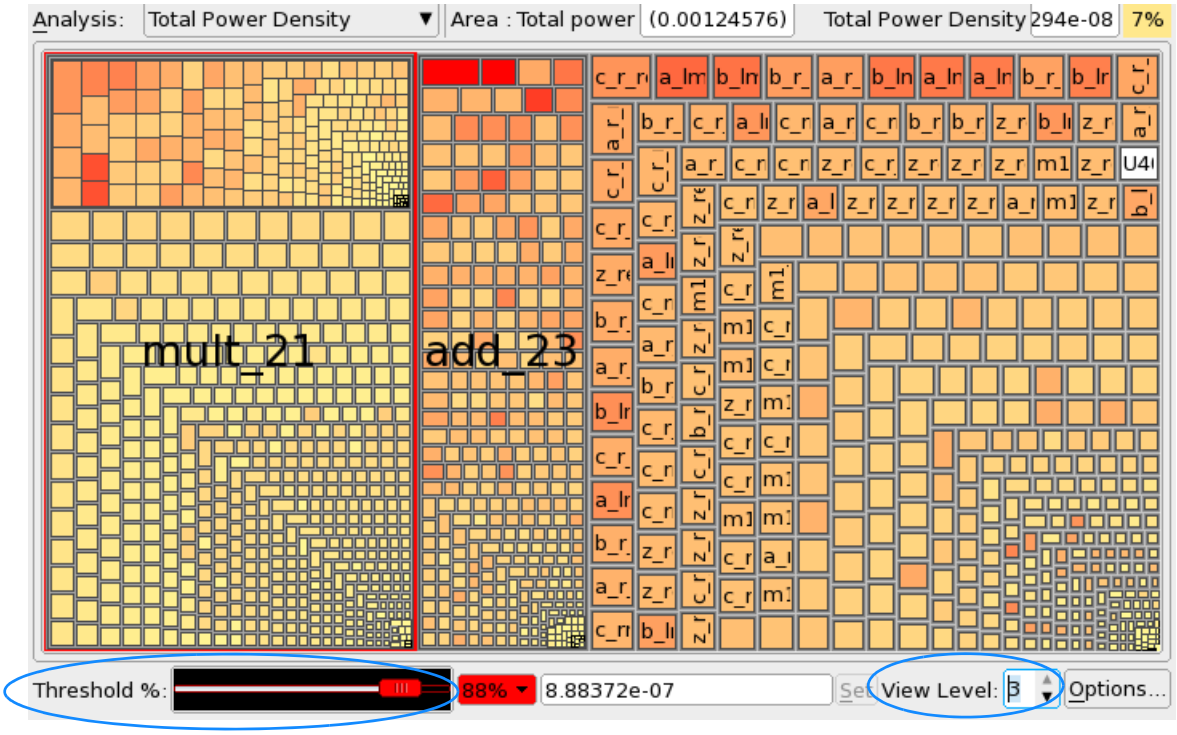
Tree maps convey hierarchical data with squares that represent cells in the hierarchy. The thickness of the lines between the squares tells you where you are in the hierarchy; that is, the thicker the line, the closer you are to the root of the hierarchy tree. Choose an analysis type to display. Your choices are shown in Table A-1.

Table A-1 Tree Map Analysis Types

Analysis types	Color indicator	Node size indicator
Total Power Density	Relative degree of total power consumption	Relative total power
Internal Power Density	Relative degree of internal power consumption	Relative internal power
Switching Power Density	Relative degree of switching power consumption	Relative switching power
Leakage Power Density	Relative degree of leakage power consumption	Relative leakage power
Toggle Coverage	Percentage of nets with one or more toggles	Relative net count
Toggle Average	Toggles per net count	Relative net count
Net Annotated Percentage	Percentage of annotated nets	Relative net count
Voltage Threshold Percentage	Relative percentage of threshold voltage cells	Relative leakage power
Total Power	Total power	Relative cell size within design
Internal Power	Internal power	Relative cell size within design
Switching Power	Switching power	Relative cell size within design
Leakage Power	Leakage power	Relative cell size within design

For example, Figure A-4 shows a tree map for total power density in which the area of each node represents its corresponding relative total power within the design. The color gradient progresses from yellow to red, with red indicating the highest power consumption.

Figure A-4 Tree Map



The Threshold % slider circled in blue adjusts the value at which a parameter is considered critical. In Figure A-4, sliding the control to the left causes more cells to appear red. A context-sensitive menu appears when you right-click the slider; you can customize the color selection and minimum and maximum scale colors.

The View Level option controls the number of hierarchical levels shown in the tree map.

Click the Options button to view the Power Treemap Options dialog box from which you can choose to open new tree maps in new windows and specify the toggle coverage threshold, as needed.

Histogram of selected. A histogram is a graph of the frequency distribution for a class of data. In PrimePower, the distribution can be plotted for power distribution or toggle rates.

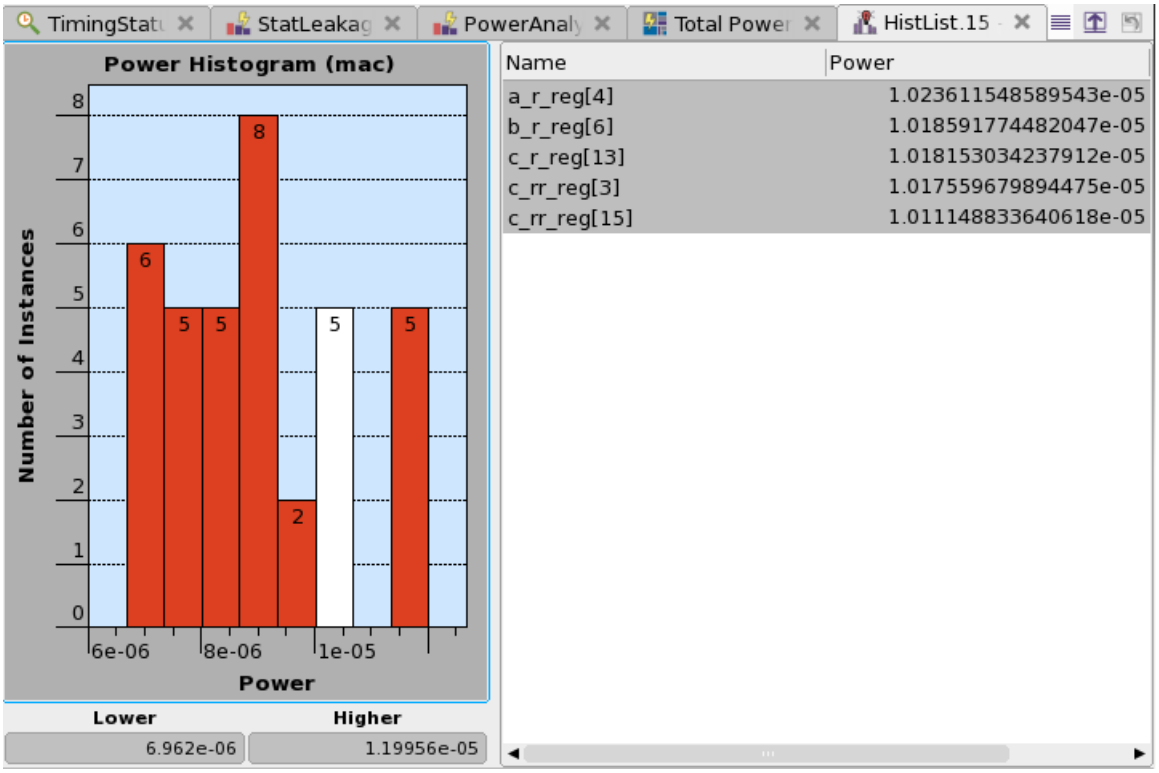
Click an instance on the map and then select Power > Histogram of Selected from the Power menu in the GUI (as shown in Figure A-1). Choose a power or toggle type to display the histogram data. The supported power or toggle types are:

- Total power

- Total rate
- Internal power
- Switching power
- Leakage power

Figure A-5 shows an example of a total power histogram for multiple cells.

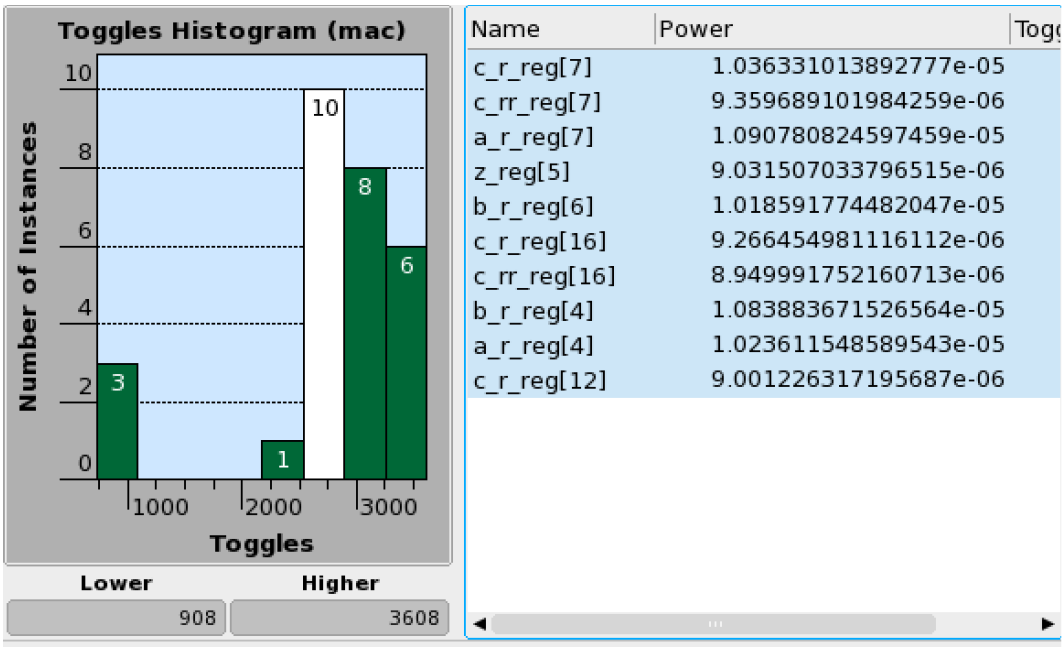
Figure A-5 Total Power Histogram



When you select a hierarchical item, the cells within the hierarchical cell are used to plot the next histogram using the hierarchical cell name in the title of the new histogram.

Figure A-6 shows an example of a total toggle rate histogram for multiple cells.

Figure A-6 Toggle Histogram



When you select a histogram bar, it turns yellow and its associated cells are displayed in the right pane. Using the specified multiple cells listed on the right (not shown), you can select any combination of them to view their histograms.

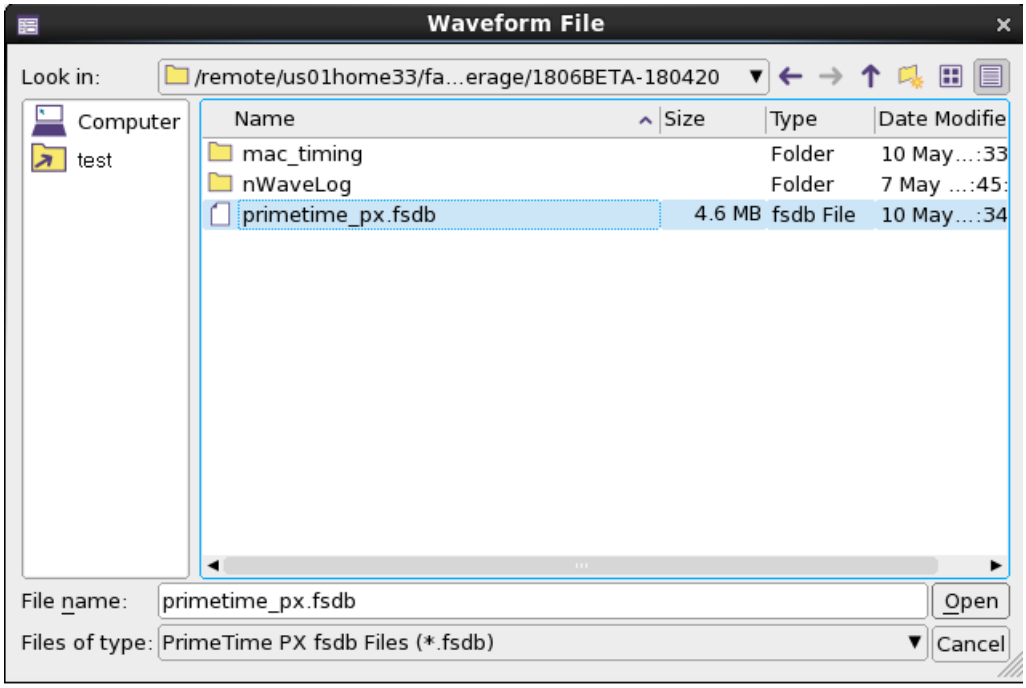
Viewing Power Waveforms

By default, PrimePower invokes the nWave tool provided with the PrimePower installation. To access the latest version of nWave waveform viewer, set the `$NOVAS_HOME` environment variable to the installation root directory of the Verdi release version. When this variable is specified, PrimePower invokes the waveform utilities from the $NOVAS_HOME/bin directory instead of the PrimePower installation directory. This ensures version compatibility when converting FSDB to VCD format during switching activity annotation.

To add the nWave waveform viewer to the Power menu in the GUI, see the steps documented in Integrating a Waveform Viewer into PrimePower.

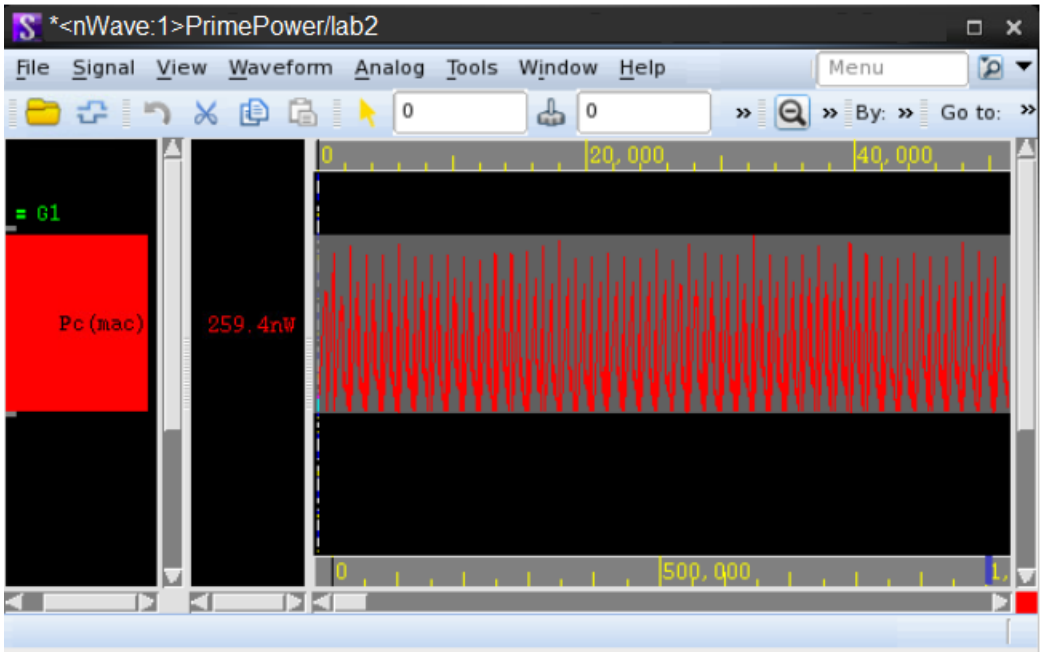
To view your switching activity in a waveform format, choose Power > View Waveforms to open the waveform viewer. In the Waveform File dialog box, select a waveform file generated from time-based power analysis.

Figure A-7 Selecting a Waveform File



PrimePower then invokes the nWave waveform viewer for display the power waveforms.

Figure A-8 Viewing Power Waveforms the nWave Waveform Viewer



You can select signals by choosing Signal > Get All Signals from the nWave menu. By default, nWave scales all the individual waveforms to a default height. To view the waveforms scaled to the total power consumption, take the following steps:

1. Select all the signals in the Signal column.
2. Choose Analog > Drawing Style > Piecewise Constant from the nWave menu.
3. Choose Analog > Zoom Value from the nWave menu.
4. In the dialog box that opens, select Full and then Close.

Integrating a Waveform Viewer into PrimePower

To display power waveforms by launching the nWave waveform viewer from the PrimePower GUI, you first need to add the waveform viewer to the Power menu selection by taking the following steps:

1. Edit and then source following script in the PrimePower GUI.

```
if {$in_gui_session == true} {
set wish_exec ""
set wish_exec [exec which wish]
if {$wish_exec==""} {
    echo "Sorry, but this interface requires wish"
    return
}
gui_add_menu -menu "&Power->View &Waveforms ..."  \
    -tcl_cmd    run_my_viewer  \
    -tooltip "View Waveforms with my_viewer " \
    -help_string "Launch my_viewer to view power waveforms" \
    -nolog

#    if {::pwrguiMenu::isPowerReady2Show}
proc run_spice_explorer {} {
   # {{FSDB Files}       {.fsdb}        }
    set wishfile [open wish.tcl w+]
    puts $wishfile "set filetypes {{{FSDB Files} {.fsdb}} {{All Files}
*
}  }"
    puts $wishfile "    set filename \[tk_getOpenFile -filetypes
\$filetypes \]"
    puts $wishfile "puts \"\$filename \""
    puts $wishfile "exit"
    close $wishfile
    set returnv [exec wish wish.tcl]
    file delete -force wish.tcl
    foreach {filename newfile} $returnv {}
    echo "\nLaunching my_viewer "
    exec my_executable my_open command $filename
```

In this script:

- Replace `my_viewer` with the name of the waveform viewer to be integrated into PrimePower.
- Replace `my_executable` with the path to the location of the executable of the waveform viewer.
- Replace `my_open_file` with the waveform command that the waveform viewer uses to open the waveform file.

2. Open the PrimePower GUI. The waveform viewer is now available under the Power menu in the PrimePower GUI.
3. Invoke the waveform viewer by choosing Power > View *my_viewer*.

   You must specify the waveform file name and select the signals for viewing before you invoke a waveform viewer.

B

Clock Gating Efficiency

This appendix includes an example of multistage clock gating per clock gate stage, and the computation of clock gating efficiency for this design, as described in the following topic:

- Clock Gating Efficiency Computation

Clock Gating Efficiency Computation

For the design in Figure B-1, PrimePower computes the clock gating efficiency of G3 as follows:

$$CGE_G = \left(\frac{4}{12}\right) * TS_{G3} * (1 - TS_{G2}) = 3.35\%$$

CGE represents the clock gating efficiency for a particular gating element only. To determine the CGE for the entire clock gate structure of a particular clock domain, the tool adds up all the CGEs of that clock domain, as shown in the following formula:

For all the clock gates G in the clock domain for CLK,

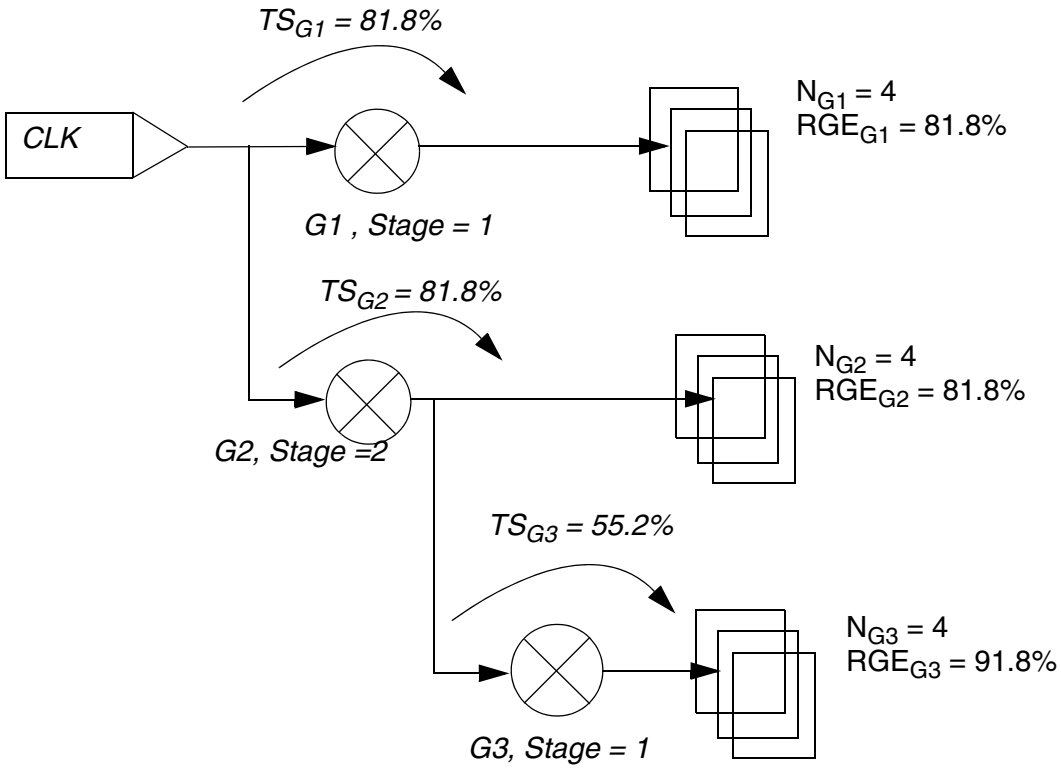
$$GE_{CLK} = \sum CGE_G$$

CGE for a clock gating stage S is the sum of all the CGEs of clock gates at stage S as follows:

$$CGE_S = \sum CGE_G$$

Figure B-1 shows an example of a multistage clock gating per clock gate stage.

Figure B-1 Clock Gating Example and How CGEs are Computed



An example computation for the clock-gate structure as shown in Figure B-1 is as follows:

$$\mathrm{CGE_{G1}} = \left(\frac{4}{12}\right) * 81.8\% = 27.3\%$$

$$\mathrm{CGE_{G2}} = \left(\frac{8}{12}\right) * 81.8\% = 54.5\%$$

$$\mathrm{CGE_{G3}} = \left(\frac{4}{12}\right) * 55.2\% * (1 - 81.8\%) = 3.35\%$$

C

Simulation Activity Formats

This appendix describes the types of simulation activity file formats. It contains the following topics:

Switching Activity Formats

Specify both the RTL and gate-level switching activity in any of the following formats:

- SAIF

  SAIF stands for Switching Activity Interface Format. The SAIF files capture signal transitions and the time spent at each logic level. The SAIF file contains the toggle counts and static probabilities for the nets in the design.

  SAIF is supported only in the averaged power analysis mode. Use the `read_saif` command to annotate the activity information in the SAIF format.

- VCD

  VCD stands for Verilog Change Dump format. It is an event-based format that contains every value change for the signals in the design and the time at which they occurred. Use the `read_vcd` command to specify the VCD file in both the averaged and time-based analysis modes.

  Use the `read_vcd` command to specify the activity files in other event-based formats within PrimePower. Internally, the tool converts these formats to VCD format using the appropriate utility.

  Table C-1 lists the formats, extensions, and conversion utilities supported by the tool:

Table C-1 Supported File Formats, Extension, and Conversion Utilities

File type	Extension	Utility
VCD	Default file type	None
VPD	.vpd	$VCS_HOME/bin/vpd2vcd
Compressed VCD	.Z	uncompress
Gzipped VCD	.gz	gunzip
FSDB	.fsdb	In PrimePower, use the `read_vcd` command. Otherwise, use the fsdb2vcd utility. $SYNOPSYS/bin/fsdb2vcd $NOVAS_HOME/bin (if $NOVAS_HOME environment variable is set to Verdi installation)
Bzip2	.bz2	bunzip2

Note:
If your file is of the unsupported format, the tool treats it as a VCD file and reads the data directly.

- FSDB

  FSDB is a binary event-based format.

  - When specified an FSDB file with the `read_vcd` command, the tool converts the FSDB data to the VCD format.
  - When specified an FSDB file with the `read_fsdb` command, the tool reads the FSDB file activity directly.

All the formats are supported in the averaged power analysis mode. In the time-based mode, specify the simulation activity in the VCD or FSDB format.

Note:
When you specify switching activity from multiple sources, the tool honors the latest one.

- Activity Files Generated from a SystemVerilog Simulation

  Use the activity files generated from a SystemVerilog simulation if the files are in VPD or .fsdb format. Specify the format as SystemVerilog using the `-format` option of the read_vcd or read_fsdb command.

SAIF File Format

For specifying switching activity data, two types of SAIF files are supported:

- RTL SAIF

  If the gate-level simulation data is not available, provide RTL SAIF files that contain the switching activity of the primary inputs, hierarchical ports, and synthesis-invariant elements, such as sequential elements.

  For the tool to perform correct switching activity annotation, you must specify the correct name mapping technique.

- Gate-Level SAIF

  If the gate-level SAIF file contains activity annotations for all the elements in the design, PrimePower can accurately calculate the average power consumption of the design.

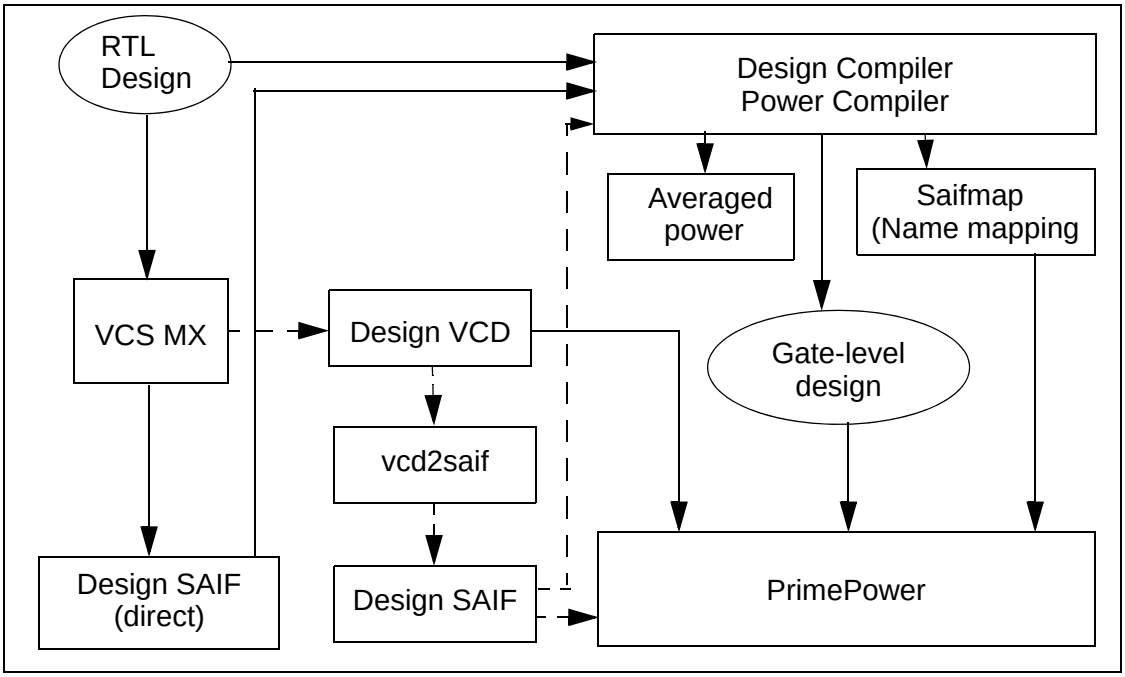
Use the `read_saif` command to specify the activity information in the SAIF file format generated from RTL or a gate-level netlist.

Generating SAIF Files

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

Figure C-1 shows two ways of generating a SAIF file. The solid lines indicate the suggested SAIF flow while the dotted lines indicate the alternative method of SAIF flow using various Synopsys tools.

Figure C-1 SAIF File Generation and Its Usage With Various Synopsys Tools



When specifying an RTL-based activity file, read the SAIF file into the Power Compiler tool to generate a name mapping file, which matches the RTL object names to the gate-level objects. Then read the name mapping file in PrimePower before annotating the RTL-SAIF or VCD files. For more details on mapping RTL-to-gates, see Chapter 3, “Name Mapping.”

The following sections describe the ways of generating SAIF files:

- Generating SAIF Files From Simulation
- Generating SAIF Files From VCD Output Files

Generating SAIF Files From Simulation

The VCS MX tool can generate a SAIF file directly from simulation. This direct SAIF file is smaller relative to the VCD files. Your input design for simulation can be an RTL or gate-level design. Also, the design can be in Verilog, SystemVerilog, VHDL, or mixed HDL formats. When your design is in Verilog or SystemVerilog format, you must specify system tasks to VCS MX using the SAIF generation procedures.

When generating the SAIF file during simulation, use the default monitoring method. This monitoring method captures the switching activity of only the synthesis-invariant objects, such as ports, tristate cells, black box cells, flip-flops, latches, retention registers, and hierarchical cells other than clock-gating cells. Integrated clock-gating cells and latch-based isolation cells are not synthesis invariant objects and therefore not captured.

If the library forward-SAIF file contains details of state and path dependencies, the backward SAIF file generated by the tool also contains these details.

For more information about generating SAIF files using these methods, see the *Power Compiler User Guide*.

When you run the `read_saif` command, the tool annotates the simulation activity in the SAIF file to the gate-level objects. Use the command to generate a report showing the percentage of nets annotated and the source of annotation.

Generating SAIF Files From VCD Output Files

Use the `vcd2saif` utility to convert the VCD or other event-based format activity files generated by VCS into SAIF files. To generate the SAIF file and annotate the switching activity, use the following procedure:

1. Run the simulation to generate a VCD output file.
2. Use the `vcd2saif` utility to convert the VCD output file to a SAIF file.
3. Annotate the switching activity within the SAIF file.

The `vcd2saif` utility converts the RTL or gate-level VCD file generated by the VCS tool into a SAIF file. The utility is provided with PrimePower installation and is located in the `install_dir/bin` path.

The `vcd2saif` utility supports the same event-based activity formats as the `read_vcd` command, shown in Table C-1 on page C-2.

The `vcd2saif` utility does not extract state-dependent and path-dependent switching activity when converting VCD to the SAIF format. For information about each option, use the `vcd2saif -help` command.

VCD File Format

During timed-based analysis, you must provide the event-based activity for the design using only the VCD, VPD, or .fsdb formats. PrimePower calculates power for every event for accurate results.

For specifying the activity data, two types of VCD files are supported by the tool:

- RTL VCD

  Use the `read_vcd -rtl` command to read the activity file. With the `-rtl` option, the tool performs name mapping and event propagation.

  Use the `read_vcd -zero_delay` command if the VCD file is generated from a zero-delay simulation.

  You can also use the `set_rtl_to_gate_name` command to map the RTL and gate-level object names. As the RTL object names can change after synthesis, the `read_vcd` command cannot map the names present in the RTL VCD file to the gate-level objects, causing inaccurate results. To avoid this problem, use the `set_rtl_to_gate_name` command.

  When you use the `-rtl` or `-zero_delay` option, the tool generates cycle-accurate waveforms. The clock cycle is used as the default waveform interval.

- Gate-level VCD

  When you read the VCD file without specifying the `-rtl` or `-zero_delay` option with the `read_vcd` command, the tool assumes that the file is a gate-level VCD file.

Generating VCD Files

Use the VCS or other HDL simulator to generate the event-based activity files. These simulators use standard PLI procedures.

For information about how to generate VCD, VPD, and FSDB activity files, refer to the VCS manual.

Reading VCD Files

In the time-based power analysis mode, PrimePower does not support reading activity information from multiple VCD files. If you specify multiple VCD files, the tool uses only the latest VCD file specified.

If you use the `read_vcd -pipe_exec` command, the tool defers reading the VCD file, including the header annotation.

FSDB File Format

Use the `read_fsdb` command to specify event-based switching activity information in the FSDB format. PrimePower uses the switching activity information to annotate activity and to calculate power for each event in the design.

When the `read_fsdb` command is executed, the tool first reads the header of the activity file to determine which portion of the design is annotated in the time-based power analysis flow. The remainder of the activity file is not read until power analysis is performed.

The following are the commonly used options of the `read_fsdb` command when specifying an event-based activity file in the FSDB format:

`-rtl`

Indicates if the input FSDB file is generated from RTL simulation. When specified, the `read_fsdb` command uses name mapping set by the `set_rtl_to_gate_name` command. The `update_power` command also uses zero delay simulation to create events on unannotated nets.

Using an FSDB file generated from RTL simulation is recommended, so the sequential cells, primary inputs, and black box outputs are annotated, while combinational nets are not. Events on these unannotated nets must be generated via simulation.

PrimePower supports annotation of activity generated from SystemVerilog VCS simulation with modports to define interfaces between blocks with appropriate name mapping files.

`-zero_delay`

Indicates if the input FSDB file is generated from the zero-delay gate-level simulation. When specified, power is averaged over each clock cycle. You need to specify the Synopsys Design Constraint (SDC) file, so that the clocks and transition time at primary inputs are defined. When not specified, most events happen at clock edges, which might lead to large unrealistic peak power.

`-time` *`{start_time end_time start_time end_time ...}`*

Specifies time windows in which the activity is counted for power calculation. Define the window list with an even number list in an increasing order. For example, `-time {10 20 50 70}` specifies time windows `[10ns, 20ns]` and `[50ns, 70ns]`. If you do not know when the simulation time ends, use a negative number, such as `-time {10 -1}` for an activity time window from 10 ns to the end of simulation time.

The default activity time window is the whole simulation time in the FSDB file. The specified time window is automatically adjusted by the tool based on the timescale in the FSDB file or the value specified with the `-waveform_interval` option of the `set_power_analysis_options` command. For example, if the `-waveform_interval` option is specified as `10`, the tool changes `-time {13 33}` to `-time {10 30}`.

This option applies to averaged power analysis, time-based power analysis, and statistical leakage power variation analysis.

`-when`

Specifies a Boolean condition to determine which simulation time from the FSDB file is considered for power analysis. When specified, the time selected is chosen automatically based on the logical value in the FSDB file.

Examples

The following example reads the FSDB file dump.fsdb and uses the first 100 ns of the events in the file.

```
pwr_shell_shell> read_fsdb dump.fsdb -time {0 100}
```

The following example reads the FSDB file dump.fsdb and includes time periods when the specified condition is true.

```
pwr_shell> read_fsdb dump.fsdb -when {net125 && net126}
```

Debugging Problems With the Activity Files

The following errors might occur when you use the activity files for power analysis:

1. The hierarchy in the activity file does not match that of the current design.

   In this case, the total number of annotated nets and cells in the summary report generated using the `read_vcd` command report is zero. The tool generates warning messages for all the nets in the design that do not exist in the activity file.

   This hierarchy mismatch generally occurs when you incorrectly set the `-strip_path` option of the `read_vcd` and `read_fsdb` commands. Therefore, the activity file does not contain all levels of design hierarchy.

2. Syntax errors occur when parsing the activity file because of

   - Unsupported file format. For example, PrimePower generates syntax error messages when you specify a VCD file in the unsupported format, such as an enhanced VCD (EVCD) file.
   - Unmatched format or version between conversion utilities and activity files. PrimePower invokes the appropriate utilities to convert the switching activity data to the VCD format as applicable. The utility must be available in the search path and must match the version of the data.

3. Name mapping of the RTL to gate-level design object is incorrect.

   Specify a name mapping file that contains a list of the `set_rtl_to_gate_name` commands to avoid the error.