

PrimeRail User Guide

Version M-2017.06, June 2017

SYNOPSYS®

Copyright Notice and Proprietary Information

©2017 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.

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

About This Guide	xiv
Customer Support.	xvii
1. Introduction to PrimeRail	
Features and Benefits.	1-2
Using PrimeRail in the Design Flow	1-4
In-Design Rail Analysis in IC Compiler	1-5
In-Design Rail Analysis in IC Compiler II	1-7
PrimeRail Analysis Flows	1-8
2. Working With PrimeRail	
Running PrimeRail	2-2
Starting PrimeRail	2-2
Command Log Files	2-3
Setup Files	2-4
Entering pr_shell Commands	2-4
Using PrimeRail Command-line Interface	2-5
Using Regular Expressions	2-6
Using Aliases	2-6
Listing and Rerunning Previously Entered Commands	2-7
Redirecting and Appending Command Output.	2-7
Getting Help While Running PrimeRail	2-8
Command-Line Help.	2-8
PrimeRail Man Pages.	2-8

Displaying the List of Keyboard Shortcuts	2-9
PrimeRail Working Directories	2-9
Exiting PrimeRail	2-11
Working With the GUI	2-11
Opening the GUI	2-13
Using GUI Windows.	2-13
Menu Bar	2-14
Toolbars	2-14
Layout Panel.	2-14
Status Bar	2-15
Panels.	2-15
Choosing Menu Commands in GUI Windows	2-15
Previewing pr_shell Commands in Dialog Boxes	2-17
Displaying the List of Keyboard Shortcuts	2-17
Working With the Console	2-17
Viewing the Session Log	2-18
Viewing the History View	2-20
Viewing the Physical Layout in the Layout Viewer.	2-21
Setting GUI Preferences	2-22
Recording and Replaying a GUI Session	2-23
Closing the GUI	2-24
Working With Licenses	2-24
Getting, Listing, and Releasing the Licenses in Use	2-24
Enabling License Queuing.	2-25
Enabling Multicore Processing	2-26
Configuring Multithreading.	2-26

3. Preparing Input Data

Creating Taps for Ideal Voltage Sources.	3-3
Manually Specifying Tap Locations	3-5
Treating Top-Level Pins as Taps	3-6
Using Existing Instances as Taps	3-6
Importing a Tap File	3-7
Importing Taps From the Context	3-8
Taps From Result Contexts	3-9
Creating Taps With Packages	3-10

Creating Taps With Package SPICE Models	3-12
SPICE Files	3-14
Reporting Taps	3-15
Validating Taps	3-17
Finding Invalid Taps	3-19
Creating Taps Using Layout-Based Interactive Tap Tool	3-20
Setting TLUPlus Models	3-22
Specifying Voltage Settings	3-22
Setting Voltages	3-23
Setting Rail-Specific Voltages	3-24
Reporting Supply Voltages	3-24
Setting Operating Conditions	3-25
Reading SDC Files	3-25
Loading Signal Parasitics Information	3-26
Loading Timing Constraints	3-26
Reading Switching Activity Information	3-27
Annotating Switching Activity	3-28
Reporting Switching Activity	3-31
Reading Electromigration Rules	3-32
Reporting Electromigration Rules	3-33
Creating Rail Setup Data in the IC Compiler Tool	3-34
4. Preparing and Checking Design Data	
Setting Up the Technology File	4-2
Checking CCS Power Libraries	4-6
Settings for Generating Macro Cell Waveforms	4-7
Converting LEF/DEF Cell Data	4-8
Converting LEF Files into the Milkyway Format	4-9
Converting DEF Files into the Milkyway Format	4-10
Importing Multiple DEF Files for Hierarchical Designs	4-11
Setting Up the Design	4-14
Before Opening a Design	4-15

Opening Milkyway Designs	4-17
Opening IC Compiler II Designs	4-19
Checking Design Readiness	4-19
Reporting Design Checking Results	4-20
5. Analyzing IC Compiler II Designs	
Rail Analysis Flow With IC Compiler II Designs	5-2
Input Data for Analyzing IC Compiler II Designs	5-5
Displaying Maps in the IC Compiler II GUI	5-5
Sample Scripts for Analyzing IC Compiler II Design Libraries	5-9
6. Integrity Checking	
Rail Integrity Checking Flow	6-2
Rail Integrity Checking	6-3
Integrity Checking Types	6-7
Floating Shapes	6-8
Floating Pin Shapes	6-9
Dangling Pin Shapes	6-11
Discontinuous Connection Checking	6-12
Wires	6-13
Paths	6-14
Pin Shapes	6-14
Rectangles	6-15
Dangling Vias	6-15
Missing Vias	6-16
Checking for Fully or Partially Enclosed Vias	6-17
Modeling Via Cuts by Bounding Boxes	6-17
Checking for Stacked Vias	6-18
Ignoring Via Enclosure Metals as Endpoints	6-21
Allowing Endpoints Outside the Overlapping Areas	6-21
Merging Same Layer Shapes	6-22
Cutting a Merged Shape into Rectangles	6-23
Specifying a Maximum Distance Between Vias	6-24
Allowing Small Areas	6-24
Area-Based Filtering	6-24

Toggling Between Integrity Results	6-25
Reporting Integrity Results	6-26
Deleting Integrity Results	6-27
Viewing Integrity Errors in the Error Browser	6-28
7. Extracting Supply Net Parasitics	
Setting Extraction Options	7-2
Extracting Supply Net Parasitics	7-4
Handling Via Arrays	7-6
Hierarchical Extraction	7-7
Determining PG Shape Overlaps	7-10
Multicore and Parallelism	7-10
Extraction and Power Analysis in Parallel	7-11
Reporting Power and Ground Net Hierarchy	7-13
Viewing Parasitic Maps	7-13
Reusing Power and Extraction Data	7-14
8. Analyzing Power and Updating the Currents	
Introduction to Power and Current Waveform Generation	8-2
Required Input Data and Data Flow for Current Calculation	8-3
Analyzing Power and Generating Current Waveforms	8-4
Static Power and Current Generation	8-5
Dynamic Event-Based Power and Current Calculation	8-6
Dynamic Vectorfree Power and Current Calculation	8-7
Power Analysis With Assigned Power	8-7
Assigning Power to PG Pins	8-8
Assigning Power to Signal PG Nets or Pins	8-9
Assigning Internal and Leakage Power	8-9
Importing a Power File	8-10
Power Scaling	8-10
Waveform Calculation With Assigned Current	8-12
Debugging Power and Current Information	8-13

Displaying Power and Current Maps	8-15
Saving and Reusing Power Data	8-17
9. Static Rail Analysis	
Static Rail Analysis Flow	9-2
Setting Ideal Voltage Sources	9-3
Reading Supply Parasitics	9-3
Performing Static Voltage Drop Analysis	9-4
Displaying Static Rail Analysis Results	9-6
Displaying Voltage Drop Map	9-7
Writing Rail Data	9-8
Debugging Rail Results	9-11
Toggling Between Rail Results	9-12
Performing Static Electromigration Analysis	9-13
Required Input Data for Electromigration Analysis	9-14
Setting Electromigration Options	9-14
Performing Electromigration Analysis	9-16
Displaying Static Electromigration Analysis Results	9-17
Viewing the Electromigration Violation Report	9-17
Displaying Electromigration Maps	9-20
Checking Threshold Information	9-21
Opening Error Data in the Error Browser	9-23
Considering Body-Bias Effects	9-25
10. Dynamic Rail Analysis	
About Dynamic Rail Analysis	10-2
Dynamic Rail Analysis Flow	10-3
Required Input Data for Dynamic Analysis	10-5
Library Characterization for Dynamic Analysis	10-6
Preparing Data for Library Characterization	10-6
Editing the configure.tcl File	10-7
Running Library Characterization	10-9
Troubleshooting Library Characterization	10-11

Dynamic Voltage Drop Analysis	10-13
Dynamic Rail Analysis With Multiple Time Windows	10-14
Dynamic Rail Analysis Without Transient Noises	10-15
Dynamic Rail Analysis Results	10-16
Dynamic Electromigration Analysis	10-17
Dynamic Electromigration Analysis Results	10-19
11. Rail Macro Models for Milkyway Designs	
What Is a Rail Macro Model?	11-2
Using a Rail Macro Model.	11-2
Connectivity Views.	11-4
Current Source Locations	11-6
Creating Rail Macro Models	11-7
Creating Macro Models With Milkyway Data	11-7
Creating Macro Models With GDSII Data	11-8
Layer Mapping Files	11-9
Layer Text Files.	11-10
Layer Translation Table Files	11-10
Creating Macro Models With Synopsys Embed-It! Integrator Data	11-11
Creating Macro Models With On-Resistances	11-13
Creating Simple Rail Macro Models	11-15
Updating Macro Models	11-18
Updating Rail Macro Models With SPICE Data	11-18
Updating Rail Macro Models With Simulation Data	11-21
Setting Mode Sequences for Macro Models.	11-28
Checking the Macro Model Content	11-28
Reporting Unmodeled Macro Cells.	11-31
Displaying Current Source Information.	11-31
Viewing Macro Models in the GUI.	11-32
12. Power Management Cells	
Analyzing Power Management Cells	12-2

Rail Analysis With Multithreshold-CMOS Switch Cells	12-3
Setting On-State Resistances	12-5
Annotating Switch-Cell Instances	12-8
Deleting Annotations.	12-10
Checking Supply Networks	12-10
Rail Analysis With Hard Macro Power Management Cells	12-12
Analysis Flow When Analyzing With Hard Macro PM Cells	12-12
Power Management Location Files	12-14
Creating CONN Views and PML Files	12-14
Manually Inserting On-Resistances	12-15
Performing Rail Analysis	12-16
What-If Analysis With Switch Cells	12-17
Inrush Current Analysis	12-18
Calculating Rush Current By Cell States	12-20
Calculating Ramp-Up Current By Net States.	12-22
Power-Up Sequence Information.	12-25
Modifying Wake-Up Time	12-25
Running the calculate_rush_currents Command.	12-26
Dynamic Rail Analysis With Inrush Currents	12-28
13. Other Analysis Topics	
Top-Level-Only Analysis	13-2
Analyzing Top-Level Designs.	13-3
Calculating Effective Resistances.	13-4
Reporting Effective Voltage Drops	13-6
Reporting Switching-Window-Based Effective Voltage Drops	13-7
Minimum Path Resistance Analysis	13-9
Using the calculate_minimum_path_resistances Command	13-10
Calculating Minimum Path Resistance During Voltage Drop Analysis.	13-10
Debugging With Minimum Path Resistance Maps.	13-11
Rail Analysis With Decoupling Capacitors	13-12
Virtual Decap Insertion Flow	13-13
Setting Assigned Leakage Current or Capacitance	13-14
Inserting Virtual Decap Cells	13-16

Voltage Drop Analysis With Regulators	13-17
Using Voltage Regulators in Dynamic Rail Analysis	13-18
Voltage Regulator Analysis Flow	13-19
Advanced Electromigration Analysis	13-22
What-If Analysis	13-27
Adding External Resistors or Non-Ideal Voltage Sources	13-28
 Appendix A. SiliconSmart Script Examples	
SiliconSmart configure.tcl Script Example	A-2
SiliconSmart Run Script Example	A-7
 Glossary	

Preface

This preface includes the following sections:

- [About This Guide](#)
- [Customer Support](#)

About This Guide

PrimeRail provides gate-level static and dynamic voltage drop and electromigration analysis capabilities that help detect and eliminate possible voltage drop and electromigration violations for system-on-chip (SoC) designs. The rail integrity checking engine can analyze designs and pin point design issues that are found in the design, and provides fixing guidance whenever possible.

This document describes how to use the PrimeRail analysis capabilities. This guide is intended to be used in conjunction with PrimeRail man pages in which you can find detailed descriptions of all PrimeRail commands.

Audience

This manual is intended for integrated circuit designers who are concerned about voltage drop and electromigration problems during the physical design process and who have knowledge of UNIX and high-level design techniques.

Before using PrimeRail, you should be familiar with the following topics:

- Physical design implementation principles
- Power analysis principles
- Timing analysis principles

Related Publications

For additional information about PrimeRail, 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:

- HSPICE[®]
- IC Compiler[™]
- IC Compiler II[™]
- Embed-It![®] Integrator
- Milkyway Environment[™]
- PrimeTime[®] Suite

- SiliconSmart®

Release Notes

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

To see the *PrimeRail Release Notes*,

1. Go to the SolvNet Download Center located at the following address:
<https://solvnet.synopsys.com/DownloadCenter>
2. Select PrimeRail, 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 <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>slow medium high</code>
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 <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>
- Telephone your local support center.
 - Call (800) 245-8005 from within North America.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>

1

Introduction to PrimeRail

The Synopsys PrimeRail tool is a power network analysis tool to analyze voltage (IR) drop, electromigration, and power network integrity problems of the design. PrimeRail enables you to locate potential voltage drop and electromigration violations for system-on-chip (SoC) designs.

PrimeRail provides accurate power network analysis with signoff quality for all kinds of designs, including full-chip designs, block-level designs, top-level designs, multivoltage designs, and so on.

The following sections offer an overview of the PrimeRail analysis capabilities:

- [Features and Benefits](#)
- [Using PrimeRail in the Design Flow](#)
- [In-Design Rail Analysis in IC Compiler](#)
- [In-Design Rail Analysis in IC Compiler II](#)
- [PrimeRail Analysis Flows](#)

Features and Benefits

PrimeRail offers gate-level static and dynamic voltage drop, electromigration, and power network integrity analyses with full-chip sign-off with package parasitics. If you have nonstandard cells, such as power management cells, PrimeRail also considers switched power domains during full-chip rail analysis.

When rail analysis is complete, PrimeRail provides maps with color highlights to show the problem areas in an easy-to-use graphical user interface. The what-if analysis feature enables you to experiment with different elements, such as user-defined taps, virtual decoupling capacitors, virtual vias and so on, to see the effects on IR drop and electromigration. Early rail analysis can also be done before the design is complete—for example, when only the power routing is done or right after the placement stage in a power network analysis.

The key features that PrimeRail provides are:

- Power network integrity checking

The integrity checking capability finds physical connectivity issues in the design's power network, such as missing vias, floating shapes, floating pins, or dangling shapes. The tool reports design issues that are found in the design and provides guidance to fix them.

- Voltage drop and electromigration analysis

The voltage drop and electromigration analyses allow you to find voltage drop and electromigration violations of the specified power and ground nets.

- Macro modeling support

The macro modeling capability allows you to model a macro into a rail macro model which is an abstract representation of the internal physical model (including PG structure and currents) and the transistor-level electrical model of a hard macro cell. You can capture the analog effects in a rail hard macro during full-chip analysis with reusability.

- Close integration with the IC Compiler and IC Compiler II tools

You can invoke PrimeRail In-Design rail analysis within the IC Compiler tool to perform integrity checking, rail analysis and inrush current analysis. You can display maps and browse error views without leaving an IC Compiler session. This allows you to find and fix potential power network problems at an early place and route stage without leaving the tool environment.

You can also invoke PrimeRail In-Design Rail analysis within the IC Compiler II tool to perform integrity checking on the IC Compiler II designs.

You can generate the rail setup information in the IC Compiler tool and use the generated script in the `pr_shell` executable. This facilitates the process of the supply net extraction, power and current waveform generation, and rail analysis.

- Minimum Path Resistance Analysis

The minimum path resistance analysis feature helps you to easily detect power and ground network weaknesses in the design. This allows you to detect design defects at the chip level by simply viewing the minimum path resistance map on the power network. You can calculate the minimum path resistance from each point of the power and ground network.

- Top-level analysis

The top-level analysis capability allows you to analyze only the top-level routing of the design in combination with the results generated from the In-Design rail analysis in the IC Compiler tool. This capability is useful when you want to check the robustness of the overall power network if no sub-block is present in the design.

- Special cell support

Special cells, such as power management switch cells, on-chip regulator cells or hard macros, can have significant effects on voltage drop and electromigration. The power consumption of these cells or the power and ground routing that exists inside them can affect the voltage and current supplied to other regions of the chip. Therefore, you should consider the internal power and ground network and power consumption of the cells during rail analysis.

- Reusing power and extraction analysis data

You can reuse the power and extraction data from previous analysis runs. This allows you to generate two or more sets of results and perform data comparison, such as what-if analysis.

- Managing results

Each time when you perform rail analysis or integrity checking, PrimeRail saves the generated results with unique names. You can load and set the previously generated rail analysis or integrity checking results as the one to be used in the current session. This allows you to compare data from two or more result sets for what-if analysis.

- Multiple database support

PrimeRail reads both Milkyway and IC Compiler II design libraries.

PrimeRail supports designs in the Library Exchange Format/Design Exchange Format (LEF/DEF). You can translate the LEF/DEF database into the Milkyway format using the PrimeRail database converter.

- Map display features

You can view analysis results graphically either in the minimum path resistance, voltage drop or electromigration map. The visibility control over layout layers and design levels enables you to easily investigate possible violation problems and improves the map drawing speed.

- Graphical user interface

The PrimeRail graphical user interface is similar to the IC Compiler graphical user interface. The PrimeRail GUI provides a flexible working environment with multiple windows for performing different tasks. You can open or close the GUI at any time during the session.

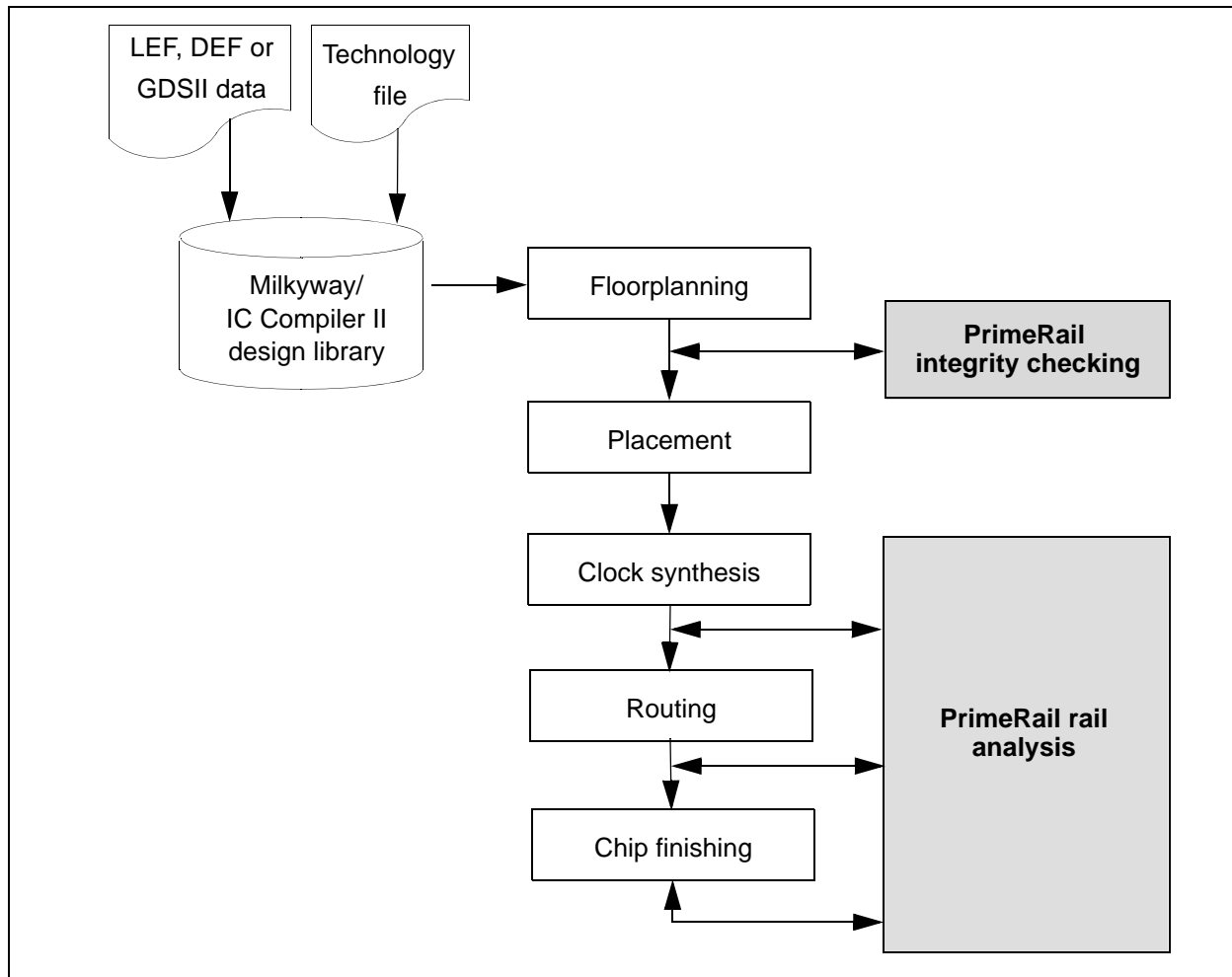
Using PrimeRail in the Design Flow

You can use PrimeRail at different stages in the design flow when power planning and initial placement are completed. When global routing is complete, you can verify the power and ground network robustness, calculate power consumption, and check for voltage drop and current density violations. These capabilities enable you to detect potential power network issues before you perform detail routing and thus reduce turnaround time of the design cycle.

In addition, you can use the PrimeRail analysis capabilities at other stages in the design flow, such as after detail routing or chip finishing.

[Figure 1-1](#) shows when to use PrimeRail in a typical design flow.

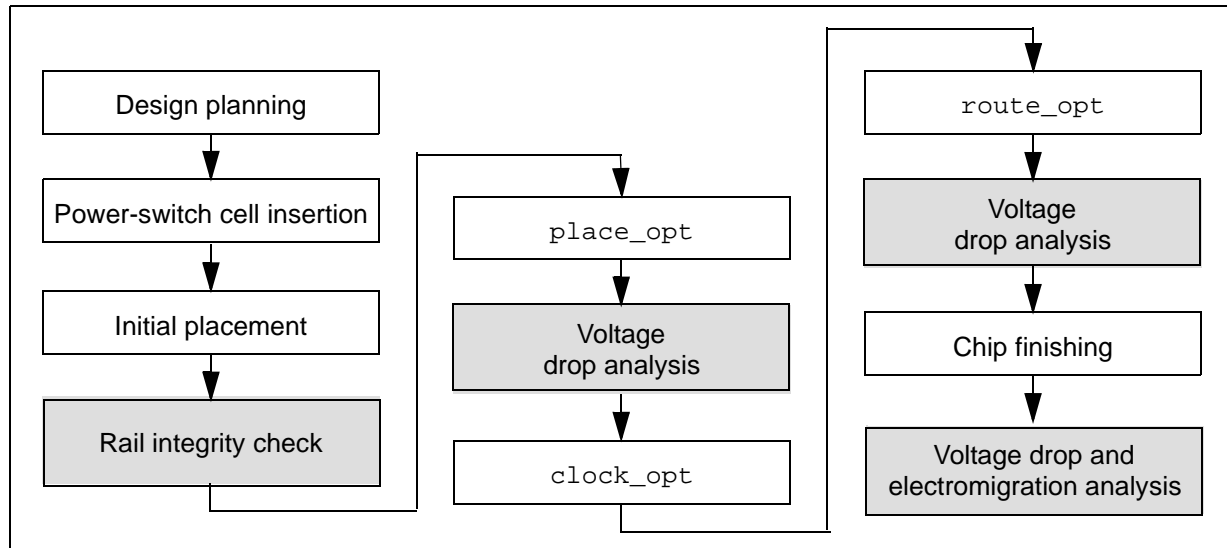
Figure 1-1 Using PrimeRail in the Design Flow



In-Design Rail Analysis in IC Compiler

PrimeRail is tightly integrated with the IC Compiler tool; you can run rail integrity checking, voltage drop analysis, and electromigration analysis directly within the IC Compiler environment. When the checking and analysis processes are finished, you can display the generated maps and error data to find and fix the problematic areas without leaving the current session. Error data is also available for display through the error browser.

Figure 1-2 The In-Design Rail Analysis Flow in IC Compiler



Within the IC Compiler tool, you can invoke PrimeRail to:

- Check rail integrity

Execute the `verify_rail_integrity` command to perform rail integrity checking with strategies that are defined using the `set_rail_integrity_layout_check_strategy` command. The rail integrity checking capability works in conjunction with the template-based power network synthesis capability inside the IC Compiler tool in a fast design creation and verification usage mode. Potential missing vias and connections are reported in the error views and report files. You can fix the problems based on these generated error views and reports.

- Perform voltage drop analysis

Execute the `analyze_rail -voltage_drop` command to perform voltage drop analysis on the design and generate voltage drop maps. When analysis is complete, you can display the generated map from the IC Compiler graphical user interface, and examine the highlighted violations in the map.

- Perform electromigration analysis

Execute the `analyze_rail -electromigration` command to perform electromigration analysis on the design and generate electromigration maps. Electromigration violations, along with the fixing recommendations, are reported in a separate text report. When analysis is complete, you can display the generated maps and error views from the IC Compiler graphical user interface, and check and fix the highlighted violations.

For a detailed description about the In-Design rail analysis capability in the IC Compiler tool, see the In-Design rail analysis chapter in *IC Compiler Implementation User Guide*.

In-Design Rail Analysis in IC Compiler II

PrimeRail is tightly integrated with the IC Compiler II tool. You can run rail integrity checking within the IC Compiler II environment after completing the power structure. You can display the generated error data to find the problematic areas and perform implementation correlation without leaving the current IC Compiler II session.

To check rail integrity in the IC Compiler II tool, execute the `verify_rail_integrity` command to perform rail integrity checking with strategies that are defined using the `set_rail_integrity_strategy` command.

Here is a script example:

```
### Open NDM design library ###
open_lib $lib_dir/myLib
open_block myBlock
create_taps -import ./inputs/taps.rpt

## Set TLUPlus files
read_parasitic_tech \
-tlup ./inputs/test.tluplus \
-layermap ./inputs/tech.map \
-name sd32nm \
save_block
save_lib

# Define a floating pin shape checking strategy:
set_rail_integrity_layout_check_strategy Float_Pin_Shape_1_VDD \
  -supply_nets { VDD } \
  -error_view Float_Pin_Shape_1_VDD \
  -pg_pin_floating_connection_check maximal \
  -ignore_eeq_for_physical_connectivity_tracing \
  -area_edge_condition touching \
  -voltage_areas { 1 }

# Run integrity check
verify_rail_integrity \
  -integrity_layout_strategies \
    [list \
      Float_Pin_Shape_1_VDD \
    ]

# Open the error browser in IC Compiler II GUI
```

For a detailed description about performing rail integrity checking in the IC Compiler II tool, see the PrimeRail In-Design topic in the *IC Compiler II Implementation User Guide*.

PrimeRail Analysis Flows

PrimeRail provides timing and power analysis capabilities to generate current waveforms. A built-in power and ground net extraction engine extracts the parasitic information for the power and ground network. With the generated current waveforms and extracted parasitic data, PrimeRail analyzes the voltage drop and electromigration violations of the design in the static and dynamic rail analysis flows.

PrimeRail provides the following capabilities for top-level, block-level or full-chip designs in the rail analysis flow:

- Physical connectivity checking, such as missing vias and floating pins
- Static voltage drop and electromigration analysis
- Dynamic voltage drop, electromigration, and vectorfree analysis
- Other analysis supports, including
 - Advanced electromigration analysis
 - Inrush current analysis
 - Multithreshold-CMOS design analysis
 - Reduced core model generation analysis
 - Minimum path resistance analysis
 - Voltage analysis with regulators
 - Virtual decoupling capacitor analysis

[Figure 1-3](#) and [Figure 1-4](#) illustrate the basic static and dynamic rail analysis flow for analyzing Milkyway designs at the top level. See [Figure 5-1](#) for the basic analysis flow for analyzing IC Compiler II designs.

Figure 1-3 PrimeRail Analysis Flow (Part I)

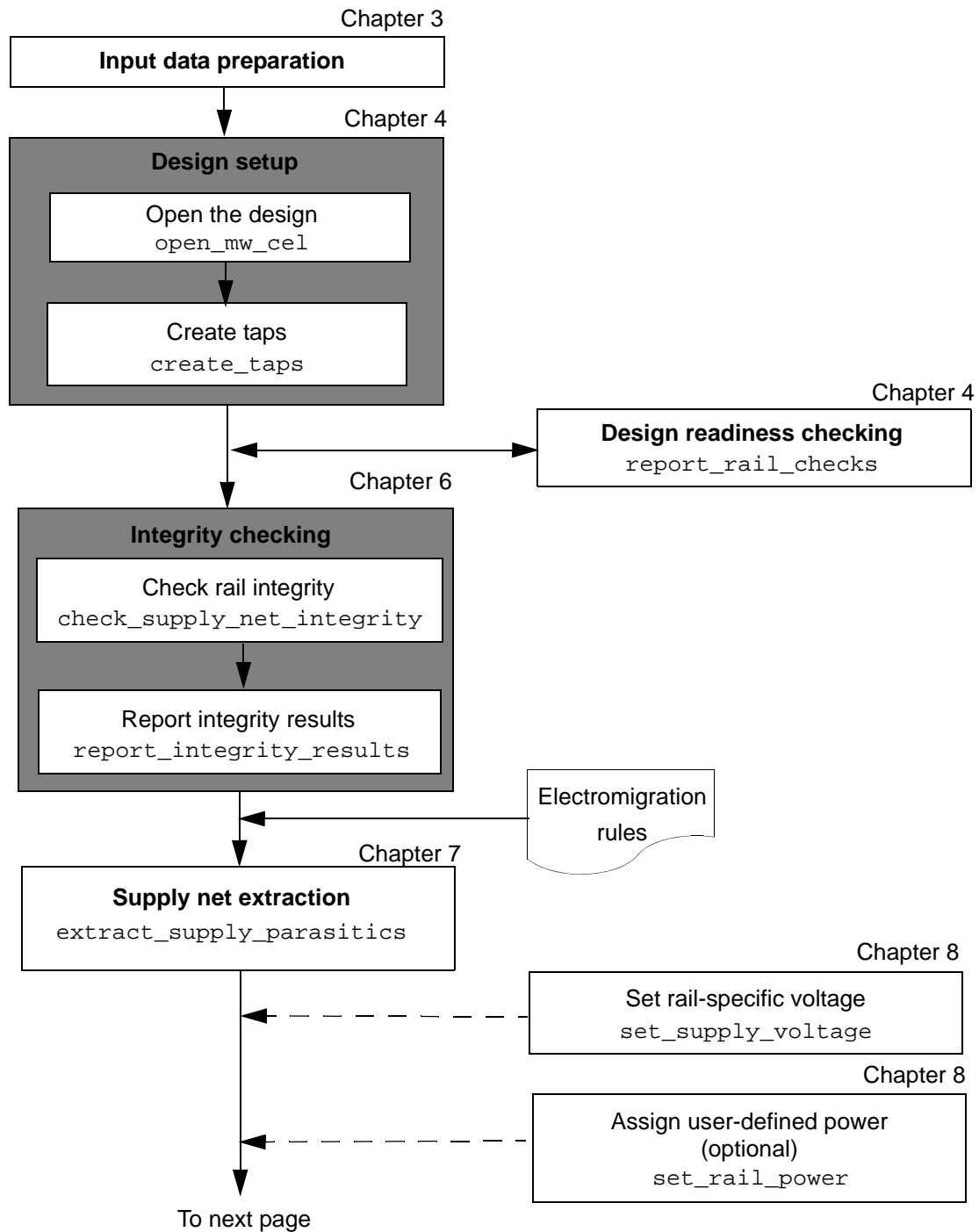
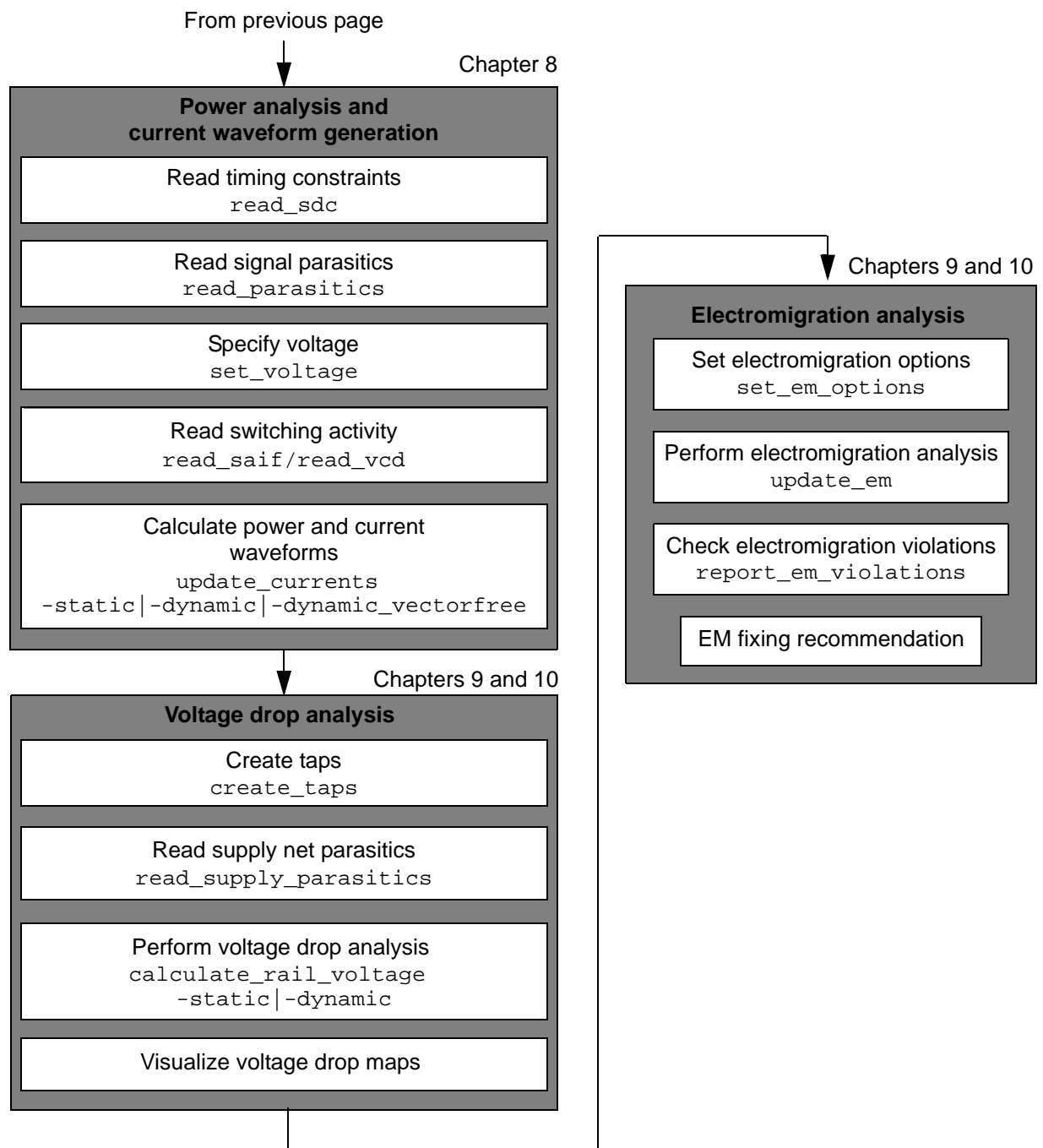


Figure 1-4 PrimeRail Analysis Flow (Part II)



2

Working With PrimeRail

PrimeRail provides a flexible working environment with a shell command-line interface and a graphical user interface (GUI). The `pr_shell` command-line interface is a text-only environment in which you enter commands at the command-line prompt. The PrimeRail GUI provides both menu commands and a command-line interface. The GUI also provides tools you can use to visualize design data and analyze calculation results.

This chapter describes the standard tasks involved when working with PrimeRail, as described in the following sections:

- [Running PrimeRail](#)
- [Working With the GUI](#)
- [Working With Licenses](#)
- [Enabling Multicore Processing](#)

Running PrimeRail

The following sections contain information about running PrimeRail using `pr_shell`:

- [Starting PrimeRail](#)
- [Entering `pr_shell` Commands](#)
- [Using PrimeRail Command-line Interface](#)
- [Getting Help While Running PrimeRail](#)
- [PrimeRail Working Directories](#)
- [Exiting PrimeRail](#)

Starting PrimeRail

PrimeRail operates in the X windows environment on UNIX or Linux. Before starting the tool, make sure the path to the bin directory is included in your `$PATH` variable.

To start the tool in the command-line interface (`pr_shell`), enter the `pr_shell` command in a UNIX or Linux prompt:

```
% pr_shell
```

To start the PrimeRail graphical user interface, enter the `pr_shell -gui` command in a UNIX or Linux shell:

```
% pr_shell -gui
```

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

- `-f script_file_name` to execute a script
- `-x` to execute a PrimeRail 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, `pr_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 `pr_shell>` prompt in the shell in which you started `pr_shell`.

The program header lists all PrimeRail packages for which your site is licensed.

To see the command options PrimeRail provides, enter the following command at the UNIX prompt:

```
%pr_shell -help
```

The following information is displayed:

```
pr_shell
  [-64bit]          (Start the 64 bit executable)
Usage: /amd64/syn/bin/pr_shell_exec
  -root_path path_name (Synopsys root path)
  [-gui]              (Start GUI immediately)
  [-display display_env_value] (Set DISPLAY environment variable)
  [-file file_name]   (Script file to exec after setup)
  [-no_init]          (Don't load any setup files)
  [-x command]        (Execute a PrimeRail shell command after setup)
  [-version]          (Show product version and exit immediately)
  [-optimize_mode]    (Run in the optimize mode)
  [-output_log_file file_name] (Output log file name)
```

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.

Command Log Files

The command log file records all `pr_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 `pr_shell_command.log`, in the directory from which you invoked `pr_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.

Note:

You must make any changes to this variable before you start the tool. If your setup file does not contain this variable, the tool automatically creates the `pr_shell_command.log` file.

Setup Files

PrimeRail runs the tool's default setup file, named `.synopsys_pr.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

This order of precedence allows you to define your general personal settings in your home directory and the design-related settings in each working directory.

The following is an example of the `.synopsys_pr.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.

See Also

- [Exiting PrimeRail](#)

Entering pr_shell Commands

Based on the tool command language (Tcl), the PrimeRail command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the `pr_shell>` prompt in `pr_shell`
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl command scripts, which are text files containing `pr_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, PrimeRail lists the qualifying names, from which you can select by using the arrow keys and the Enter key.

See Also

- [Running PrimeRail](#)

Using PrimeRail Command-line Interface

The PrimeRail command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. The syntax for a `pr_shell` command is

```
command_name [-option [argument]] [command_argument]
```

`command_name`

The name of the command.

`-option`

Modifies the command and passes information to the tool. Options specify how the command should run. A hyphen (-) precedes option names. Options that do not take an argument are called switches.

`command_argument`

An argument to the command.

Every `pr_shell` command returns a value, either a status code or design-specific information. The command status codes in `pr_shell` are

1 for successful completion

0 or { } (null list) for unsuccessful execution

The PrimeRail command language also supports wildcard characters and aliases. In addition, you can rerun commands and redirect or append command output. For details, see the following sections.

- [Using Regular Expressions](#)
- [Using Aliases](#)
- [Listing and Rerunning Previously Entered Commands](#)
- [Redirecting and Appending Command Output](#)

Using Regular Expressions

PrimeRail supports using regular expression pattern matching for names of multiple objects. You can build patterns from the following expressions:

- An ordinary character, which matches itself.
- A dot (`.`), which matches any single character.
- An asterisk (`*`), which matches occurrences of the character or expression immediately preceding it. For example, `n*` matches occurrences of the character `n`, and `.*` matches occurrences of any character.
- A backslash (`\`) followed by a special character, which matches the character. This allows you to match a character that would otherwise be considered a special character. For example, `\.` matches a dot (`.`) and `*` matches an asterisk (`*`).
- A set of characters enclosed by square brackets (`[]`), which matches any character inside the brackets. To include a right square bracket (`]`) in the set, place it at the beginning of the set.
- A caret (`^`) and a set of characters enclosed by square brackets (`[]`), which matches any character not included in the set. For example, `[^469]` matches any character except 4, 6, and 9. A caret in any other position is interpreted as an ordinary character. To include a right square bracket (`]`) in the set, place it at the beginning of the set, immediately following the caret—for example, `[^]cdrx]`.
- A minus sign (`-`) between two characters, which indicates a range of consecutive characters and matches any character in the range. For example, `[2-5]` matches 2, 3, 4, or 5 and `[b-d]` matches b, c, or d. To include a minus sign in a set, place it at the beginning or end of the set. For example, `[-cdrx]` or `[cdrx-]`.

Using Aliases

You can use aliases to create short forms for the commands you commonly use. When you use aliases, follow the guidelines below:

- The `pr_shell` interface recognizes aliases only when they are the first word of a command.
- An alias definition takes effect immediately but lasts only until you exit the PrimeRail session. To save commonly used alias definitions, store them in the `.synopsys_pr.setup` file.
- You cannot use an existing command name as an alias name; however, aliases can refer to other aliases.

The following example shows how you can use the `alias` command to create a shortcut for the `calculate_rail_voltage` command:

```
pr_shell> alias railVDD100 {calculate_rail_voltage {VDD}}
```

Use the `unalias` command to remove alias definitions created with the `alias` command. For example, to remove all aliases beginning with `f*` and the `railVDD100` alias, enter

```
pr_shell> unalias f* railVDD100
```

Listing and Rerunning Previously Entered Commands

You can use the `history` command to list the commands used in a `pr_shell` session. It prints an ordered list of previously executed commands. You can control the number of commands printed. The default number printed is 20. The `history` command is complex and can generate various forms of output. For detailed information about this command, see the command man page.

You can rerun and recall previously entered commands by using the exclamation point (!) operator.

Redirecting and Appending Command Output

If you run `pr_shell` scripts to analyze a design, you cannot see warnings or error messages echoed to the command window while your scripts are running. You can direct the output of a command, procedure, or script to a specified file in two ways:

- By using the traditional UNIX redirection operators (`>` and `>>`)
- By using the `redirect` command

You can use the UNIX redirection operators (`>` and `>>`) in the following ways:

- Divert command output to a file by using the redirection operator (`>`).
- Append command output to a file by using the append operator (`>>`).

Note:

Unlike the UNIX `redirect` operator, PrimeRail treats the (`>` and `>>`) operators as arguments to a command. Therefore, you must use a white space to separate these arguments from the command and the redirected file name. The pipe character (`|`) has no meaning in the `pr_shell` interface.

The `redirect` command performs the same function as the traditional UNIX redirection operators (`>` and `>>`), but more flexible.

The result of a redirected command that does not generate a Tcl error is an empty string. Screen output from a redirected command occurs only when there is an error. You can redirect multiple commands or an entire script.

Although the result of a successful `redirect` command is an empty string, you can get and use the result of the command you redirected. You do this by constructing a `set` command in which you set a variable to the result of your command, and then redirecting

the `set` command. The variable holds the result of your command. You can then use that variable in a conditional expression.

You can direct command output to the standard output device as well as a redirect target by using the `-tee` option.

See Also

- [Running PrimeRail](#)

Getting Help While Running PrimeRail

PrimeRail provides a variety of user-assistance tools. You can view command-line Help and man pages for `pr_shell` commands and variables, find equivalent GUI menu commands and dialog boxes for `pr_shell` commands, and display a list of keyboard shortcuts that you can use in the GUI.

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

- [Command-Line Help](#)
- [PrimeRail Man Pages](#)
- [Displaying the List of Keyboard Shortcuts](#)

Command-Line Help

The following online information resources are available while you are using `pr_shell`:

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

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

```
pr_shell> command_name -help
```

PrimeRail 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 `pr_shell` and in the console log view in the GUI,

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

```
pr_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. PrimeRail 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**.

PrimeRail displays the keyboard shortcut report in the report view. The report consists of three columns:

- Hot Key lists the shortcut keys or key combinations.
- Type indicates whether the key is a shortcut for a menu command or a Tcl command.
- Function lists the commands that the shortcut keys launch.

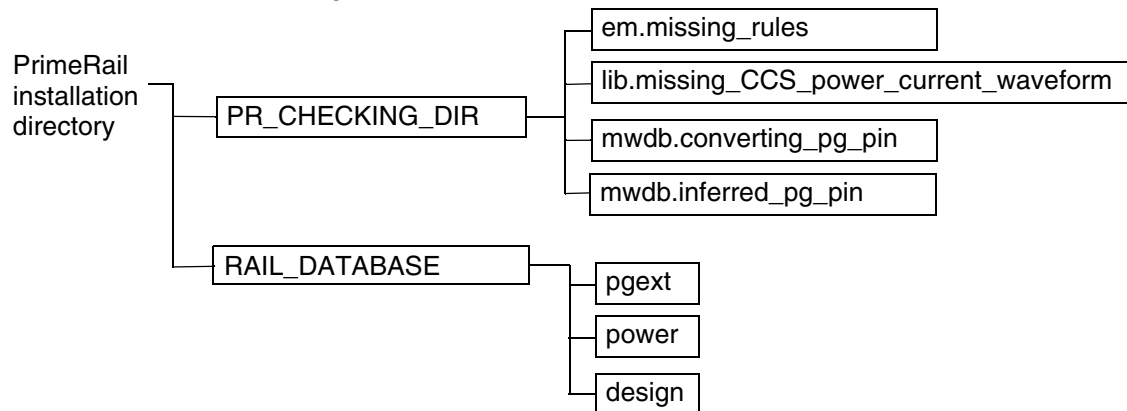
See Also

- [Running PrimeRail](#)

PrimeRail Working Directories

PrimeRail creates the following directories to save the results generated during analysis. [Figure 2-1](#) is a graphical representation of the data structures.

Figure 2-1 PrimeRail Working Directories



- **RAIL_DATABASE:** The directory where PrimeRail saves and retrieves results and log files that are generated during supply net extraction, power calculation, and rail analysis.

This directory contains the following subdirectories:

- **pgext:** The directory where PrimeRail writes and retrieves supply net extraction results.
- **power:** The directory where PrimeRail writes and retrieves power analysis results.
- **design_name:** The directory where PrimeRail writes and retrieves integrity and rail analysis results.
 - integrity
 - rail

- **PR_CHECKING_DIR:** The directory where PrimeRail saves log files, warnings, and error messages during various analysis stages.

This directory contains the following files:

- **em.missing_rules:** The file contains information for the layers that do not have electromigration rules defined.
- **lib.missing_CCS_power_current_waveform:** The file contains the names of the library cells that do not have CCS power current waveforms.
- **mwdb.converting_pg_pin:** The file contains power and ground pins that are inferred from the reference library but not in the logic library.
- **mwdb.inferred_pg_pin:** The file contains cells with power and ground pins that are derived from the reference library but not in the logic library.

See Also

- [Running PrimeRail](#)

Exiting PrimeRail

You can end the session and exit the tool at any time. To exit the tool, use the `exit` or `quit` command at the `pr_shell` prompt. To exit the tool from the GUI, enter the `exit` or `quit` command on the console command line or choose `File > Exit` in the main window or layout window.

PrimeRail does not save view settings when you exit the GUI. To save view preference before existing, choose `Options > Preferences > Save to Preferences` in the View Settings panel.

See Also

- [Starting PrimeRail](#)

Working With the GUI

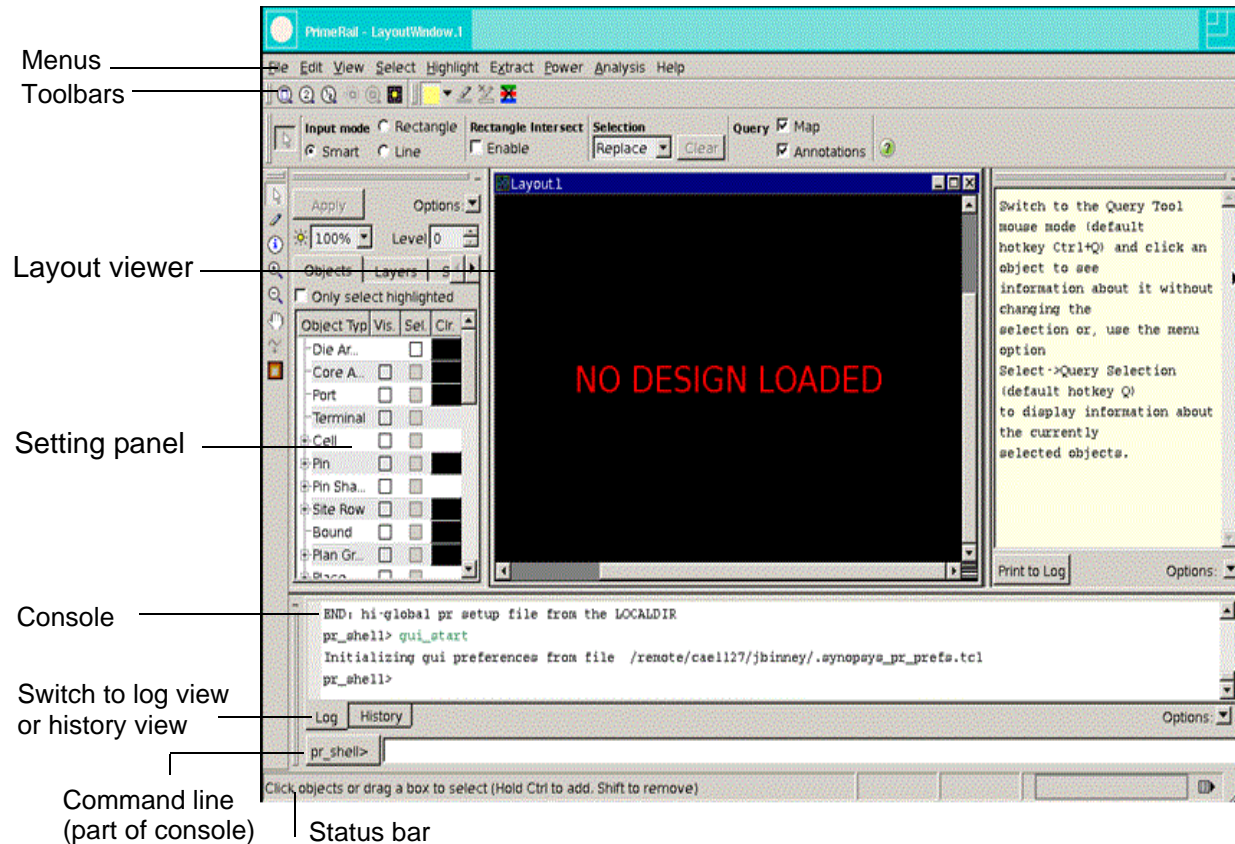
The PrimeRail GUI provides a flexible working environment with multiple windows for performing different tasks. The GUI windows operate independently in your X windows environment, but they all share the design and global selection data in the tool.

You can open or close the GUI at any time during the session. For example, if you need to conserve system resources, you can close the GUI and continue the session in `pr_shell`. When you are ready to use the GUI again, you can open it again.

When you start PrimeRail and open the GUI, the PrimeRail layout window appears.

[Figure 2-2](#) identifies the major features of the main window. In the main window, the console is docked above the status bar by default.

Figure 2-2 PrimeRail Main Window Features



For information about how to get started working with the PrimeRail GUI, see the following sections:

- [Opening the GUI](#)
- [Using GUI Windows](#)
- [Choosing Menu Commands in GUI Windows](#)
- [Previewing pr_shell Commands in Dialog Boxes](#)
- [Displaying the List of Keyboard Shortcuts](#)
- [Working With the Console](#)
- [Viewing the Physical Layout in the Layout Viewer](#)

- [Setting GUI Preferences](#)
- [Recording and Replaying a GUI Session](#)
- [Closing the GUI](#)

Opening the GUI

Before opening the GUI, make sure your `$DISPLAY` environment variable is set to the name of your UNIX or Linux system display.

To open the GUI when you start `pr_shell`, use the `-gui` option when you start the tool.

```
% pr_shell -gui
```

If you start `pr_shell` without the GUI, you can open the GUI by entering the `gui_start` command at the `pr_shell>` prompt.

```
pr_shell> gui_start
```

When you open the GUI, it automatically executes the commands in the GUI setup files named `.synopsys_pr_gui.tcl`. For information about PrimeRail setup files, see [Setup Files](#).

Note:

In addition to running the setup files when you open the GUI, the tool loads application preferences from a file named `.synopsys_pr_prefs.tcl`. For details, see [Setting GUI Preferences](#).

You can execute run scripts to run by using the `-f` option of the `pr_shell` command or the `-file` option of the `gui_start` command. For example,

```
pr_shell> gui_start -file my_gui_script.tcl
```

If you open a design in `pr_shell` and then open the GUI, the tool automatically displays the layout view of the design in the layout window.

Using GUI Windows

The GUI window displays design information in the layout panel and provides menus and dialog boxes (with keyboard shortcuts), toolbars and panels, interactive mouse button tools, a command console (with a command-line interface), and a status bar.

The title bar shows the product name (PrimeRail), the window name (LayoutWindow.1), and the name of the design.

You can move, resize, minimize, or maximize a GUI window by using the window management tools on your UNIX or Linux desktop.

For more information about the top-level GUI windows, see the following sections:

- [Menu Bar](#)
- [Toolbars](#)
- [Layout Panel](#)
- [Status Bar](#)
- [Panels](#)

Menu Bar

The menu bar at the top of a GUI window contains menus with the commands you need to work in the window. You can display a brief message in the status bar about the action that a menu command performs by holding the pointer over the command. For menu commands that can also be used by clicking a toolbar button or pressing a keyboard shortcut, the menus show representations of those alternatives.

Toolbars

The PrimeRail GUI window provides toolbars with buttons you can use to quickly perform frequently used operations or tasks. To determine the function of a toolbar button, hold the pointer over the button. A ToolTip displays the name of the button, and the status bar displays a brief description of its function. You cannot disable these messages.

You can move a toolbar to another location by dragging the double bars on one of its sides. You can also disable a toolbar and hide it.

To display or hide a toolbar or panel,

- Choose View > Toolbars > *toolbar_name*.

A check mark beside the name of a toolbar on the Toolbars menu indicates that the toolbar is visible.

Layout Panel

PrimeRail displays the graphic or textual design information in the layout window.

You can resize a GUI window or move it to another location inside the workspace area of its parent window.

In most views, you can use navigation keys (the arrow keys, Page Up, Page Down, Home, and End) to scroll through and navigate the view.

Status Bar

The GUI window displays a status bar at the bottom of the window. The status bar displays the following information:

- Status information such as the number and type of selected objects
- information about the action performed by the toolbar button, menu command, or workspace tab under the pointer

If you hold the pointer over a menu command, a toolbar button, or a tab in the workspace, the status bar displays a brief message about the action that the command, button, or tab performs.

- The relative coordinates of the pointer position over an active layout view (layout windows only)

To quickly display the list of selected objects in the Selection List dialog box, click the button at the right end of the status bar.

Panels

Panels are enhanced toolbars that open within the workspace area of the top-level window to provide tools or views for performing particular tasks. Some panels contain tabs that you can click to access different tools or views. The first time that you open a panel during a session, it displays the tools or views for its default tab.

You can move a panel to another location on or off the parent window by dragging the double bars on one of its sides. You can also dock a panel (attach it to a window edge) or let it float where you leave it.

You can also disable a panel, hiding it from view. (A panel that is associated with a particular view is automatically hidden when you close the view window.)

To display or hide a toolbar or panel,

- Choose View > Toolbar > *panel_name*.

A check mark beside the name of a panel on the Toolbars menu indicates that the panel is visible.

Choosing Menu Commands in GUI Windows

The PrimeRail GUI provides menu commands and dialog boxes for most graphic features, such as viewing, selecting, and highlighting objects in layout views. In addition, the GUI provides menu and dialog box equivalents for many `pr_shell` commands. Menu commands are grouped by function on the menus in each GUI window. You can add commands to these menus and even create new menus.

To choose a command on a menu bar menu,

- Click the menu name to open the menu, and click the command name on the menu.

A menu command can perform an immediate operation, display a submenu, or display a dialog box.

- Menu commands that display a submenu are followed by a right-pointing arrow.
- Menu commands that open dialog boxes are followed by an ellipsis (...). These commands open dialog boxes to prompt you for the information before performing any operations.
- Menu commands without an arrow or ellipsis either perform an immediate operation or open a dialog box that performs immediate operations.

When you choose a command or select a dialog box option that performs an immediate action, or when you click OK or Apply in a dialog box that requires a response, the tool displays command output (including processing messages and any warnings or error messages) in both `pr_shell` and the console log view.

This manual identifies commands with their menus in the following formats:

- *Menu > Command*
- *Menu > Submenu > Command*

where

- *Menu* represents a menu title on the menu bar.
- *Submenu* represents a menu command that displays a submenu. Some submenus contain commands that open other submenus.
- *Command* represents a command that performs an operation or displays a dialog box.

Each menu command can also be activated by a shortcut key, which is indicated on the menu by an underscore (`_`) below a letter in the command and. If needed, use the name of the modifier key (Shift or Ctrl) to the right of the command name. You can view a list of shortcut keys by choosing Help > Report Hotkey Bindings. For details, see [Displaying the List of Keyboard Shortcuts](#).

Previewing pr_shell Commands in Dialog Boxes

Many PrimeRail dialog boxes allow a preview mechanism that you can use to view the commands without running them. You can view the command equivalents for various dialog box options, or combinations of options, or generate a pr_shell command that you want to copy into a script. When you enable the preview mechanism, it remains on for all dialog boxes that support the feature until you disable it.

When you click OK or Apply in a dialog box that supports the preview mechanism while the mechanism is enabled, the pr_shell command equivalents appear in the console history log preceded by a pound sign (#). The tool does not execute the commands and they do not appear in the console log view or the shell transcript.

To enable or disable the dialog box command preview mechanism,

- Click Menu in the bottom right corner of a dialog box, and choose “Preview command only”.

A check mark next to "Preview command only" on the menu indicates that the preview mechanism is enabled.

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. The tool displays the keyboard shortcut report in a report view.

To display the report of keyboard shortcuts for the active window,

- Choose Help > Report Hotkey Bindings.

The report consists of three columns:

- Hot Key lists the shortcut keys or key combinations.
- Type indicates whether the key is a shortcut for a menu command or a Tcl command.
- Function lists the commands that the shortcut keys launch.

Working With the Console

The console provides a command-line interface and two views, a log view that displays the session transcript (the default) and a history view that displays the command list.

You can use the console to

- Enter PrimeRail Tcl commands on the console command line

- View either the session transcript (log view) or the command history list (history view) by clicking the tabs above the command line
- Display an error message man page in the man page viewer by clicking the message number in the console log view
- Copy and edit or reuse commands
- Search for, select, and save commands or messages in the log view or the history view

You can enter any PrimeRail shell command on the console command line. When you enter a command on the console command line, make sure the console has the mouse focus.

To enter a command on the console command line,

1. Type the command.
2. Click the `pr_shell>` prompt button or press Return.

You can open (or close) one console in each top-level window. When the console is open, you can dock it to the bottom or top of its parent window, or move it over or away from the window.

Viewing the Session Log

The console log view displays a transcript of the current session that includes the commands you entered and all the messages generated by the tool commands. You can use this information to check on tool status after performing functions, to troubleshoot problems you encounter, and to look up information about past functions. You can copy and paste commands from the log view to the `pr_shell` command line.

To view the session transcript,

- Click the Log tab in the console. The console changes to the log view, which is displayed by default whenever the console is opened.

You can use commands on the Options menu (on the right side of the console above the command line) to

- Find text in the transcript
- Select and copy text in the transcript
- Search the transcript for commands or messages
- Save text from the transcript in a text file

To find text in the transcript,

1. Choose Options > Find.

2. Type the text you want to find in the Find box.

You must type the text exactly as it appears in the transcript, including uppercase and lowercase letters and blank spaces.

3. Search the transcript forward, by clicking Next, or backward, by clicking Previous.

To copy a used command from the log view to the command line,

1. Select the command or portion of the command that you want to copy in the log view.
2. Middle-click the console command line.

If you already have text on the command line, middle-click where you want to paste the selected information.

You can copy text in the transcript that you want to paste somewhere else, such as in a file.

To copy text in the transcript,

1. Drag the pointer over the text to select it.

You can select all of the text in the transcript by choosing Options > Select All.

2. Choose Options > Copy.

You can search the transcript for commands or messages, and you can set the types of text that you want the search mechanism to find.

To search the transcript,

1. (Optional) Set up the search criteria to include or exclude certain types of information by choosing commands on the Options menu.

You can include or exclude commands, error messages, warnings, or information messages. Check marks appear on the Options menu next to the commands for the types of text included in the search.

2. Search forward or backward by choosing Options > Find Next or Options > Find Previous.

The console highlights the next or previous instance of any text type you included in the search.

You can save all the text in the transcript, save selected text, or save just the error and warning messages.

To save text from the transcript in a text file,

1. Do one of the following:
 - To save the transcript, choose Options > Save Contents As.
 - To save selected text, drag the pointer over the text to select it, and choose Options > Save Selection As.
 - To save error and warning messages, choose Options > Save Errors/Warnings As.
2. In the dialog box that appears, navigate to the directory where you want to save the file.
3. Enter the file name in the “File name” box.
4. Click Save.

Viewing the History View

The console history view lists shell, menu, and stroke commands you have used in the current session. You can do the following with this list:

- See which commands you have used
- Find and reuse commands already used
- Copy commands in the list
- Save the list for later use

To view the command history for the current session,

- Click the History tab in the command console.

The console displays the list of commands, each with a number showing the order in which you used it.

You can use the scroll bars on the history view to scroll through the list of commands, or you can enter the `history` command on the command line to see the list of commands in the log view.

To reuse a command listed in the history view, you can do any of the following:

- Double-click the command in the history view.
- Select the command in the view and click either Execute (to immediately rerun the command) or Edit (to paste the command on the command line where you can edit it).
- Enter an exclamation point character followed by the number of the command on the `pr_shell` command line. For example, to reuse the fifth command, enter the following:

```
pr_shell> !5
```


You can copy commands in the list that you want to paste somewhere else, such as in a file.

To copy commands in the history list,

1. Drag the pointer over the text to select it.

You can select all of the text in the transcript by choosing Options > Select All.

2. Choose Options > Copy.

To save the command history list (or a portion of the list) in a text file,

1. Perform the following steps:

- If you are saving the entire list, click the Save Contents As button or choose Options > Save Contents As.

The Save Contents As dialog box appears.

- If you are saving just a portion of the list, choose Options > Save Selection As.

The Save Selection As dialog box appears.

2. If necessary, navigate to the directory where you want to save the file.

3. Enter the file name in the “File name” box.

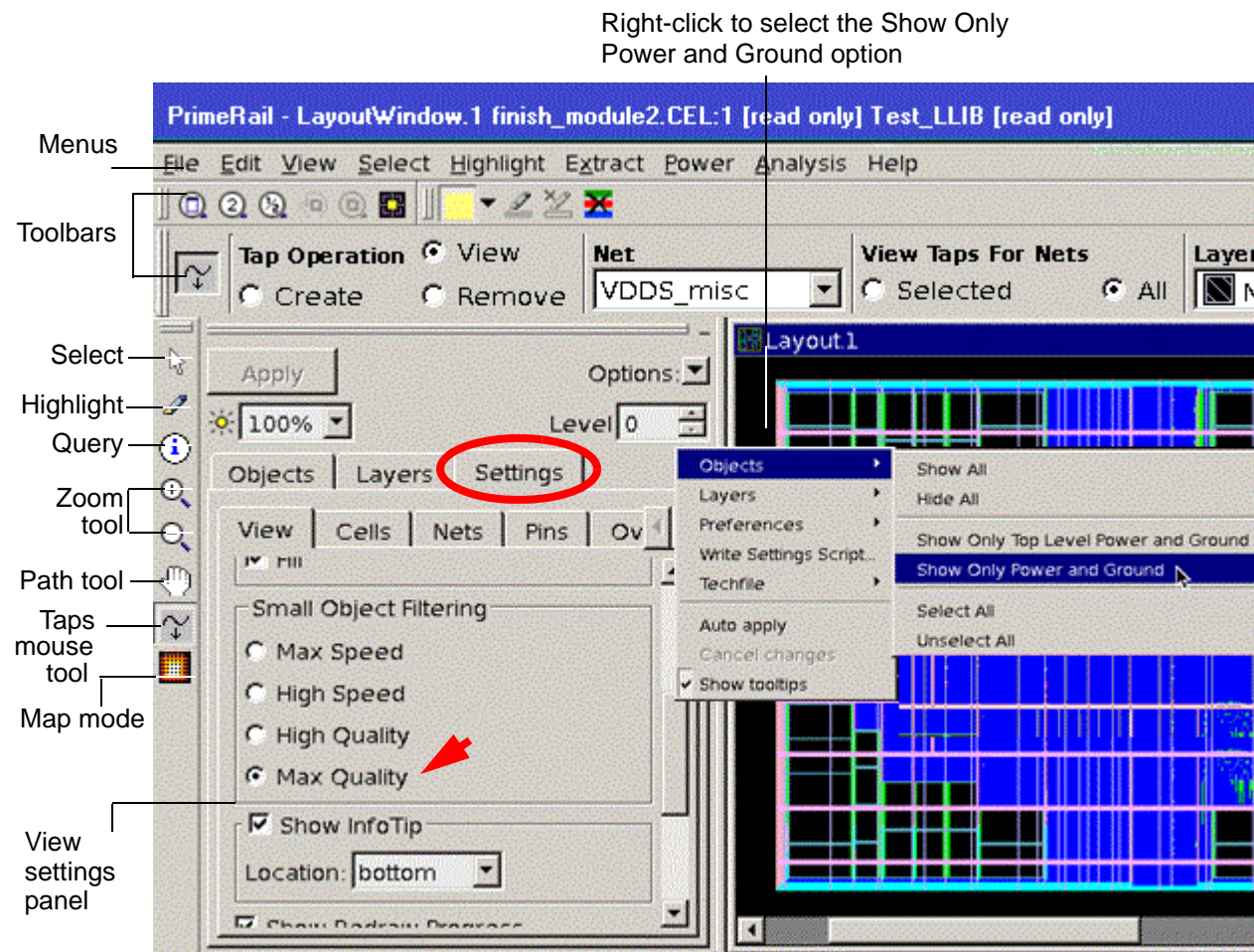
4. Click Save.

You can set options in the Application Preferences dialog box to control which types of GUI commands are included in the history list. For details, see [Setting GUI Preferences](#).

Viewing the Physical Layout in the Layout Viewer

When you read in a design, the layout window displays the design. [Figure 2-3](#) identifies the major features in the layout window.

Figure 2-3 PrimeRail Layout Window Features



The layout window provides the physical design working environment where you can view, analyze, and edit the physical design.

When the layout is loaded, right-click and select the Objects > Show Only Power and Ground option to display only the power and ground nets in the design. In the View setting panel, click the Settings tab and choose Max Quality for filtering small objects in the design.

Setting GUI Preferences

At the beginning of a GUI session, the tool loads GUI preferences and layout view settings from the preferences file, .synopsys_pr_prefs.tcl, in your home directory. You can change global preferences for the GUI and the window environment by setting options in the Application Preferences dialog box. When you change preference settings during a GUI session, the new settings are automatically saved in your preferences file.

You can set the following options in the Application Preferences dialog box:

- Font options
- GUI command logging options
- Global tool default options
- Global layout view options
- Global schematic view options

Recording and Replaying a GUI Session

You can use the command log file as a replay script to reproduce a GUI session in the tool. The command log file records the `pr_shell` commands processed by the tool, including setup file commands and variable assignments. Before you record a GUI session, make sure that the preferences for logging selection operations and interactive GUI operations are enabled in the Application Preferences dialog box.

Important:

You must replay the entire command log file as is, with no modifications, to reproduce the exact state of the design in the GUI.

To set GUI command logging preferences,

1. Choose View > Preferences.

The Application Preferences dialog box appears.

2. Select Global Settings in the Categories tree.
3. Select the optional output logging options as needed.
 - To enable logging of interactive selection operations, select the “Selection commands” option.
 - To enable logging of other interactive GUI operations, select the “GUI commands” option.
4. Click OK or Apply.

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

Closing the GUI

You can open or close the GUI at any time during the session. For example, if you need to conserve system resources, you can close the GUI and continue the session in `pr_shell`.

To close the GUI, do one of the following:

- Choose File > Close GUI in the main window or layout window.
- Choose Window > Close All Windows in any GUI window.
- Enter the `gui_stop` command.

```
pr_shell> gui_stop
```

Working With Licenses

Running PrimeRail requires one of the following license packages:

- PrimeRail
 - In-Design-Rail-Integrity
 - In-Design-Static-Rail
- PrimeRail-static

For more information about licenses and their usage, see the following topics:

- [Getting, Listing, and Releasing the Licenses in Use](#)
- [Enabling License Queuing](#)

Getting, Listing, and Releasing the Licenses in Use

When you invoke PrimeRail, 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 PrimeRail.

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

```
pr_shell> list_licenses

Licenses in use:
    PrimeRail (1)
    In-Design-Rail-Integrity (1)
    In-Design-Static-Rail (1)
1
```

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

```
pr_shell> remove_license PrimeRail
```

See Also

- [Working With Licenses](#)

Enabling License Queuing

PrimeRail 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_QUEUE TIME` 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 `pr_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 'PrimeRail'
```

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

You have already started the tool and are running a command that requires a PrimeRail 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 'PrimeRail'.
```

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

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

See Also

- [Working With Licenses](#)

Enabling Multicore Processing

Several PrimeRail 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 PrimeRail. PrimeRail supports up to 16 cores per license.

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.

Configuring Multithreading

Multithreading performs tasks in parallel by using multiple cores on the same machine, using a single process memory image. When using multithreading, each parallel task is called a thread. For the best performance during multithreading, you should limit the number of threads to the number of available cores, which is the number of CPUs in the machine times the number of cores per CPU.

To enable multithreading for those commands that support it, set the `set_host_options -max_cores` option to a value greater than one and less than or equal to the number of cores available on your machine.

When you enable multithreading, multithreaded commands create the number of threads specified, even if the number is more than the number of available cores. You must set an appropriate number of threads, so that the command does not use more resources than your machine has. Overthreading can reduce performance because the extra threads compete for resources. For best performance, do not run more than one thread per available core.

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

```
pr_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.

3

Preparing Input Data

To analyze voltage drop and current density violations in a design, PrimeRail requires a certain set of data to be available in the database, such as ideal voltage drop sources, timing constraints, signal parasitics, switching activity, and so on. Or, you can use the `analyze_rail -script_only` command in the IC Compiler tool to generate setup information and a run script for running rail analysis in PrimeRail.

PrimeRail also supports the wire and via electromigration rules through an electromigration rule file in either Advanced Library Format (ALF) or TLUPlus format. Usually the advanced library file can be obtained from the foundry. If you need an application note on preparing ALF files based on the electromigration rules provided by the foundry, contact your local Synopsys application consultant.

This chapter provides information about preparing input data, as described in the following tasks:

- [Creating Taps for Ideal Voltage Sources](#)
- [Setting TLUPlus Models](#)
- [Specifying Voltage Settings](#)
- [Setting Operating Conditions](#)
- [Reading SDC Files](#)
- [Loading Signal Parasitics Information](#)
- [Loading Timing Constraints](#)

- [Reading Switching Activity Information](#)
- [Reading Electromigration Rules](#)
- [Creating Rail Setup Data in the IC Compiler Tool](#)

Creating Taps for Ideal Voltage Sources

Taps are used to model the external voltage source environment in which the device under analysis (DUA) operates; they are not part of the design itself, but can be thought of as virtual models of voltage sources. During rail analysis, PrimeRail combines power consumption results and resistive network results to solve for voltage drop values at each node in the resistive network. To achieve more accurate results, PrimeRail needs the location of the ideal voltage source and the ideal power supply in the design.

Use the `create_taps` command (or choose Rail > Manage Taps in the GUI) to create tap point objects in the current session. PrimeRail considers these created taps as part of the supply network in the subsequent rail analysis. If a tap is already present at a location at the time when the `create_taps` command is executed, PrimeRail replaces the existing tap point with the newly created tap, and discards the previously defined tap point name. When this occurs, a summary message is issued, reporting the number of the replaced tap points.

You can create two or more tap points close to one another, typically by running multiple `create_taps` commands to specify taps in different ways. Depending upon the situation, this might have an effect on the subsequent rail analyses. However, if you place taps too close to one another, they effectively act as a single tap point and the effective resistance between them is too small to have an impact on the analysis results. For example, if the taps are sufficiently separated at opposite ends of a relatively long segment of a net shape, there is nontrivial resistance between them, which alters the rail results more substantially.

In PrimeRail, you can create taps in one of the following ways:

- [Manually Specifying Tap Locations](#)
- [Treating Top-Level Pins as Taps](#)
- [Using Existing Instances as Taps](#)
- [Importing a Tap File](#)
- [Importing Taps From the Context](#)
- [Creating Taps With Packages](#)
- [Creating Taps With Package SPICE Models](#)
- [Creating Taps Using Layout-Based Interactive Tap Tool](#)

The tool also allows you to run validation checking during tap creation or verify the created taps. For more information, see [Validating Taps](#), [Finding Invalid Taps](#) and [Reporting Taps](#).

[Table 3-1](#) lists the commonly used options of the `create_taps` command.

Table 3-1 Commonly Used Options of the `create_taps` Command

Option	Description
<code>-name</code>	<p>Specifies the name for the tap to be created.</p> <p>All taps are uniquely named, either by using the <code>-name</code> option or by the default naming. If the <code>-name</code> option is not specified, the taps are named TAP_NNN, where NNN is a number that starts at 1 and is incremented with each created tap.</p> <p>See Manually Specifying Tap Locations for more information.</p>
<code>-point</code> and <code>-layer</code>	<p>Specifies the physical location for a tap object when the tap is virtually connected to the design. If there is no supply net (or supply pin) or via shape at the specified location, which means there is no conductive path to the supply network, the tap has no effect on rail analysis despite being present.</p> <p>See Manually Specifying Tap Locations for more information.</p>
<code>-supply_net</code>	<p>Specifies the supply net with which this tap point is associated.</p> <p>A tap point is always associated with a specific net either explicitly or implicitly. If the <code>create_taps</code> command is run without the <code>-supply_net</code> option (which is allowed for the <code>-top_pg</code> and the <code>-of_objects</code> options), the tap is implicitly associated to the supply (power or ground) net connected to the object from which the tap was defined.</p>
<code>-of_objects</code>	<p>Creates tap points on the specified objects. Each item in the argument list can be a cell or a library cell, or a collection of cells and library cells.</p> <p>When <code>lib_cell</code> is defined within the <code>object_list</code> parameter, the tap points are created on all instances of that <code>lib_cell</code> used in the design.</p>
<code>-nocheck</code>	<p>Chooses not to check the tap points against layout geometry to determine whether the tap points touch any layout geometry. By default, the command checks tap points against layout geometry if they are created based on the <code>-point</code>, <code>-layer</code>, and <code>-import</code> options.</p>
<code>-import</code>	<p>Specifies a file that defines tap xy locations, layer numbers and supply nets. See Importing a Tap File and Creating Taps With Packages for more information.</p>

Table 3-1 Commonly Used Options of the `create_taps` Command (Continued)

Option	Description
<code>-snap_distance distance</code>	Defines the snap distance in microns. When a tap does not touch any layout geometry, the tool searches the nearby layout geometries within the specified distance by expanding to the left, right, top and bottom corners. The tool creates a new tap point at the corner that is within the minimum distance to the original location specified by the <code>-point</code> option. This option must be used with the <code>-point</code> and <code>-layer</code> options.
<code>-honor_non_pg_pin</code>	Allows taps to be created on a non-PG pin that is connected to the supply net. The tap point is located at the center of the pin shape geometry. Use this option with the <code>-of</code> option to support bump cells when pins are unavailable or are of the signal pin type.

Manually Specifying Tap Locations

You can create a tap in the layout by specifying the `create_taps` command with the `-layer` and `-point {X-coord Y-coord}` options.

If a tap point is defined in a location where a tap already exists, PrimeRail issues a warning message and replaces the original point with the new one.

You can specify the supply net with which this tap point is associated using the `-supply_net supply_nets` option. You have to specify the `-supply_net` option when using together with the `-point` and `-layer` options.

Alternatively, you can specify a name for the created tap point with `-name tap_name` option. If the specified name is already present in the design, a warning message is issued, and the new tap replaces the original one. When multiple tap points are created by a single invocation of the `create_taps` command, all of the created taps are named in the form `tap_name_NNN`, where NNN is a unique integer identifier and `tap_name` is the string argument to the `-name` option.

Note:

When you use the `-point` option in conjunction with the `-of_objects` option, the xy location is relative to the origin of the specified object. Otherwise, it is relative to the origin of the current top-level cell.

You cannot specify the location with the `-point` option when using the `-top_pg` option.

Example

The following example shows how to create a tap using absolute coordinates. The command checks whether it touches any layout geometry.

```
pr_shell> create_taps -supply_net VDD -layer 3 -point {100.0 200.0}
Warning: Tap point (100.0, 200.0) on layer 3 for net VDD does not touch
any polygon (RAIL-305)
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Treating Top-Level Pins as Taps

Use the `-top_pg` option of the `create_taps` command to create tap points for all the top-level supply (power or ground) pins of design cells. You cannot use the `-top_pg` option in combination with the `-point` option. When the `-layer` option is used, PrimeRail creates the tap points at the specified layer instead.

Use the `-top_pg` option when analyzing a block-level design or when the designs do not have physical pad or bump information available.

Example

The following example creates taps for all the top-level supply pins.

```
pr_shell> create_taps -supply_net VDD -top_pg top
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Using Existing Instances as Taps

Run the `create_taps -of_objects object_list` command to create tap points on the specified objects. Each item in the object list can be a cell or a library cell, or a collection of cells and library cells. By default, PrimeRail creates the taps on the power or ground pins of the objects. If the `-point {X-coord Y-coord}` option is specified, PrimeRail creates the taps at the specified physical location relative to the origin of each object.

Use the existing instances for creating taps when analyzing a top-level design where pad cells are instantiated physically.

Example

The following example creates taps for each instance of the library cell `some_pad_cell`. One tap per instance is created on `metal1` at a location of {30.4 16.3} relative to the origin of the cell instance.

```
pr_shell> create_taps -of_objects [get_lib_cell */some_pad_cell] \
               -layer metal1 -point {30.4 16.3}
```

The following example creates taps on all power or ground pins of all cell instances matching the pattern `*power_pad_cell*`.

```
pr_shell> create_taps -of_objects [get_cells -hier *power_pad_cell*]
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Importing a Tap File

If you are working with a block or a design that does not yet have pins defined, or if the location or design of the pad cells is not finalized, you can define tap xy locations, layer numbers and supply nets in an ASCII file and then import it to the tool using the `create_taps -import` command. PrimeRail automatically detects if the file is in the format of a PrimeRail tap file or an IC Compiler virtual power pad file.

The supported PrimeRail tap file format is as follows:

```
net_name layer_number X-coord Y-coord R(R_value) L(L_value) C(C_value)
```

The PrimeRail tap file supports lumped resistance-inductance-capacitance (RLC) networks. The unit is ohm for resistance, henry for inductance, and farad for capacitance.

To create a tap file for packaging SPICE models, define the tap information in the following format:

```
net_name layer_number X-coord Y-coord PKG_PORT package_model_port
```

Use the `PKG_Port` keyword to specify a package port. Specify each `package_model_port` in uppercase.

The IC Compiler virtual power pad file does not support lumped RLC networks. The supported IC Compiler virtual power pad file format is as follows:

```
net_name
X-coord Y-coord layer_number
```

In either of the files, lines starting with semicolon (;) or pound (#) symbols are treated as comments. The x-coordinate and y-coordinate values are specified in microns.

Example

The following example shows how to import taps from a PrimeRail tap file without specifying lumped RLC:

```
pr_shell> exec cat tap_file
VDD 3 500.000 500.000
VSS 5 500.000 500.000
pr_shell> create_taps -import tap_file
```

The following example shows how to import taps from a PrimeRail tap file in which lumped RLC is defined:

```
pr_shell> exec cat tap_file_rlc
VDD 3 500.000 500.000 R(1.0) L(1.0e-9) C(2.0e-12)
VSS 5 500.000 500.000
pr_shell> create_taps -import tap_file_rlc
```

The following example shows how to import taps from an IC Compiler virtual power pad file:

```
pr_shell> exec cat pad_file
VDD
555.000 555.000 3
666.000 666.000 2
VSS
555.000 555.000 5
pr_shell> create_taps -import pad_file
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Importing Taps From the Context

To import taps from a current result context to the session context, use the `create_taps -result result_name` command, where *result_name* can be either of the keywords `rail` or `integrity`.

The `-result` option cannot be specified with the `-of_objects`, `-import`, `-top_pg`, `-point` or `-layer` option.

Example

The following example imports taps from the rail context into the session context, assuming the rail context consists of the taps rail_tap1 and rail_tap2.

```
pr_shell> remove_taps
pr_shell> create_taps -result rail
pr_shell> get_taps
```

```
{"rail_tap1", "rail_tap2"}
```

```
pr_shell> report_taps
```

Name	Type	Net	Point	Layer	Object Name	Context
rail_tap1	auto_pg	VDD	(100.000 200.000)	(3) M2	/MyTop/VDD	session
rail_tap2	auto_pg	VSS	(300.230 302.180)	(1) M1	/MyTop/VSS	session

Taps From Result Contexts

During rail analysis and integrity checking, PrimeRail saves analysis results into different result contexts, depending on the type of analysis being performed. Taps in a result context are replaced whenever a new result of that same type is created. When a result context, either rail or integrity result context, is set as current, you can collect or query taps in this result context.

The following kinds of contexts are supported in PrimeRail:

- Session context: The result context that contains taps being created by the tap commands performed in the current session. The session context is the most important of the three, because the other two contexts are sampled from the session context at the time when the run of rail analysis or integrity checking is complete.

Note that if you change the current result context either with the `current_rail_result` or `current_integrity_result` command, the taps in the session context will not be changed accordingly.

- Rail result context: The result context that is created by sampling the session context at the time when the `calculate_rail_voltage` or `calculate_minimum_path_resistances` command is executed. Taps from the session context are therefore sampled and recorded in the rail result data when rail result context is saved for later retrieval.

See [Displaying Static Rail Analysis Results](#) for more information.

- Integrity result context: The result context that is created by sampling the session context at the time when the `check_supply_net_integrity` command is executed. Taps from the session context are therefore sampled and recorded in the integrity result data when the integrity result context is saved for later retrieval.

See [Toggling Between Integrity Results](#) for more information.

See Also

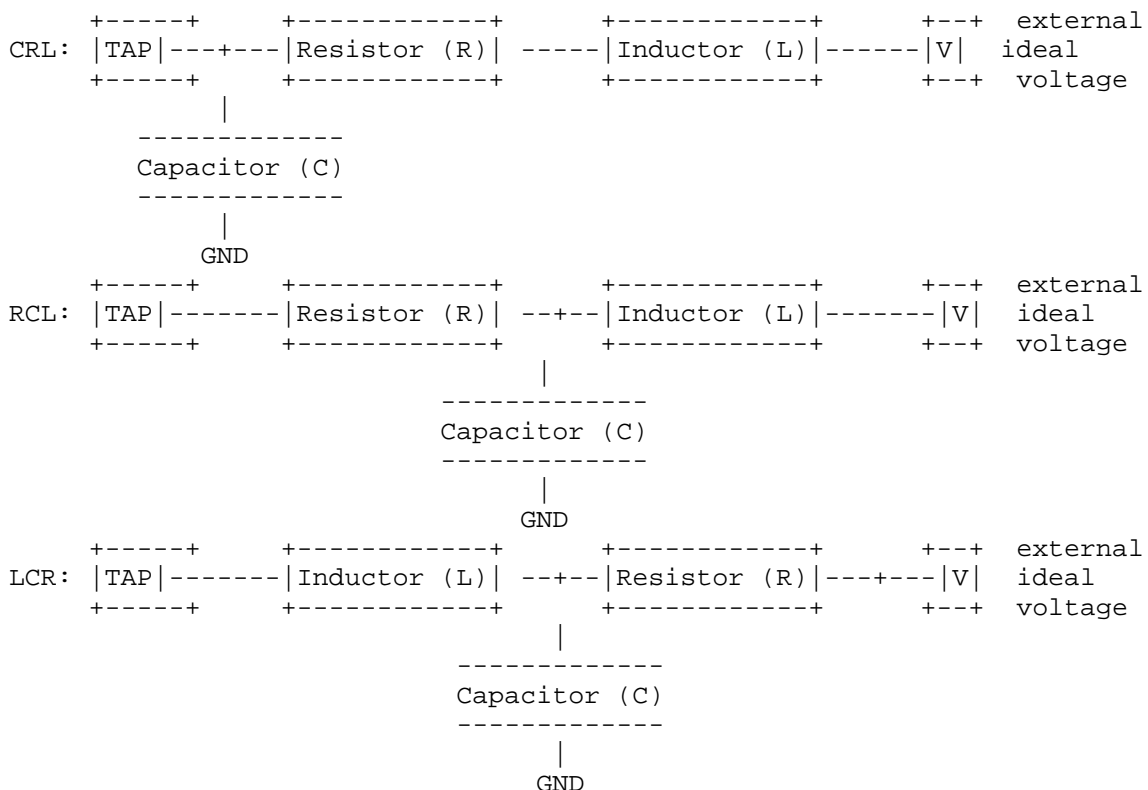
- [Creating Taps for Ideal Voltage Sources](#)

Creating Taps With Packages

To use a tap as an RLC (resistance-inductance-capacitance) model for a package, use the `set_tap_package_model` command to assign a fixed set of electrical characteristics to the taps, so these effects are considered during rail analysis.

The `set_tap_package_model` command provides three types of built-in electrical models for different package configurations: CRL, RCL and LCR. To specify the parasitic values for the various components of the models, select a model type with the `-type` option and the `-R`, `-L` and `-C` options. To model a single electrical component between the tap and the external ideal voltage, specify the `-R`, `-L` or `-C` options without the `-type` option.

The following example shows the component configurations for the built-in electrical models. The difference between the built-in electrical models is the number of resistors, capacitors and inductors placed in series between an external ideal voltage source and a tap point.



The following table lists commonly used options of the `set_tap_package_model` command.

Table 3-2 Commonly Used Options for the `set_tap_package_model` Command

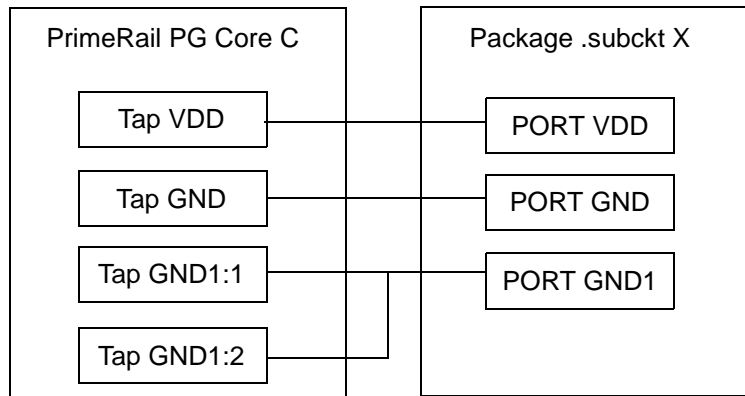
Option	Description
<code>-R, -L, -C</code>	Specifies the values of resistance, inductance, or capacitance in the library units to the taps.
<code>-type</code>	<p>Specifies the parasitic component model when two or more parasitic elements, including resistance, induction or capacitance, are applied to the specified taps. The valid values are RCL (default), CRL, or LCR.</p> <p>Each parasitic model type is determined based on the sequence of the model components when viewed from the tap. For instance, the RCL type indicates that the tap parasitics are modeled as a resistor, a capacitor and an inductor in series, with the resistor connected to the tap and the inductor connected to an external ideal voltage source.</p>
<code>-spice_top</code>	Specifies a subcircuit as the top subcircuit. This is useful when there are multiple top subcircuits in the design or this is a hierarchical SPICE circuit. PrimeRail searches the hierarchy and selects the root subcircuit in the hierarchy tree as top circuit.
<code>-spice_include_path</code>	Specifies the search path to the combined SPICE files that are identified with the <code>.INCLUDE</code> statement.

Note:

When a SPICE package model is set in the design, PrimeRail ignores the taps that are not related to the package model during rail analysis.

[Figure 3-1](#) shows how PrimeRail connects taps to the SUBCKT devices.

Figure 3-1 Mapping Taps and SUBCKT Card Devices During Packaging Flow



Example

The following example defines a simple lumped parasitic model.

```
pr_shell> set_tap_package_model -type crl -R $Rval \
        -L $Lval -C $Cval TAP_23
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Creating Taps With Package SPICE Models

In some cases, the built-in models created with the `set_tap_package_model -R -L -C` command do not describe all the electrical behavior of the device package, such as the coupling between the package inputs. You can create a package model by preparing a SPICE SUBCKT file in which each package is described as a SUBCKT device.

After the package SPICE model is available, you must either run the `set_tap_package_connection` command or ensure that the taps text file includes the `PKG_PORT` reserved word to connect the taps to the package SPICE ports.

To create taps with SPICE models,

1. Create a SPICE file.

See [SPICE Files](#) for more information about how to create the SUBCKT file for the packaging flow.

2. Run the `create_taps` command to create taps.

See [Creating Taps for Ideal Voltage Sources](#) for more information about creating taps.

3. Run the `set_tap_package_model -spice` command to read in the SPICE file. PrimeRail processes these taps as user-defined taps and treats them as interface nodes between package SPICE circuits and core power and ground circuits.
4. Run the `set_tap_package_connection` command to connect the package ports to the taps.

Specify the `-no_mapping` option if you want to connect the specified package ports to the taps with the `package_port_name` attribute before a SPICE file is read with the `set_tap_package_model` command.

5. Run the `report_taps` command with the `-package` option to check the connectivity between the SUBCKT ports and the created taps.

Example

The following example creates taps by importing a tap file with tap-to-package-port mapping and a package SPICE model.

```
pr_shell> exec cat pkg_tap_file
VDD 19 540 597 PKG_PORT BC1
VDD 19 25 597 PKG_PORT BC2
pr_shell> create_taps -nocheck -import ./pkg_tap_file
pr_shell> set_tap_package_model -spice ./simple.sp
```

The following example creates taps by importing a tap file and a package SPICE model. When the taps are available, the tool connects these taps to the SPICE package ports.

```
pr_shell> exec cat tap_file
VDD 19 540 597
VDD 19 25 597
pr_shell> create_taps -nocheck -import ./tap_file
pr_shell> set_tap_package_model -spice ./simple.sp -no_mapping
pr_shell> set_tap_package_connection -port_name PKG_PORT_VDD1 \
[get_taps TAP_0]
pr_shell> set_tap_package_connection -port_name PKG_PORT_VDD2 \
[get_taps TAP_1]
```

The following example creates taps without immediate port-to-tap mapping until the SPICE file is read.

```
pr_shell> create_taps -name TAP_0 -layer 19 -point { 540 597 } \
-supply_net VDD
pr_shell> create_taps -name TAP_1 -layer 10 -point { 25 597 } \
-supply_net VDD
## Set tap package models (no mapping until SPICE file is imported)
pr_shell> set_tap_package_connection -port_name PKG_PORT_VDD1 \
-no_mapping [get_taps TAP_0]
pr_shell> set_tap_package_connection -port_name PKG_PORT_VDD2 \
-no_mapping [get_taps TAP_1]
```

```
## Set tap package model SPICE file
pr_shell> set_tap_package_model -spice package_model.sp
```

SPICE Files

PrimeRail supports the following models:

- Passive element models R, L, and C
- Independent source models I and V
- Dependent source models E, F, G, and H with linear dependency

Here is the syntax of the SUBCKT file:

```
.SUBCKT subckt_name
+ PKG_PORT_VDD1
+ PKG_PORT_VDD2
+ PKG_PORT_VSS1
+ PKG_PORT_VSS2
...
R node_name node_name value
V node_name node_name value
...
.ENDS
```

Table 3-3 Keywords Used in the SUBCKT File

Element	Description
subckt_name	The name of the subcircuit. tapN: The name of the tap port that is located at LayerN XN YN. LayerN: The number of the layer in the database. XN YN: The coordinates of the tap location in the database unit.
R	Defines the location and value of the resistor. Set the node name and the value to 0 if it is an ideal voltage node.
V	Defines the location and value of the voltage source. Set the node name and the value to 0 if it is an ideal voltage node. The voltage value is considered the relative value to the ideal voltage supply.

For more information about using the SUBCKT file, see the HSPICE user documentation.

The following example shows how to set a user-defined package model on the VDD net using the `simple.sp` package file. In this example, the subcircuit package is a package model for the VDD net and has two ports connected to the VDD net.

```
.SUBCKT package bc1 bc2
R1_1 bc1 1 1.0
L1_1 1 v 1.0e-9
C1_1 bc1 0 1.0e-9
R2_1 bc2 2 1.0
L2_1 2 v 1.0e-9
C2_1 bc2 0 1.0e-9
RB 1 2 1.5
LB 1 2 1.0e-9
Vdd v 0 1.5
.ENDS
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Importing a Tap File](#)
- [Reporting Taps](#)
- [Validating Taps](#)

Reporting Taps

To check the tap connectivity in the design, use the `report_taps` command to display information about tap points in the current session context, rail result context, or integrity result context. By default, the command generates a report that includes all current tap points in the session context. You can also choose to report information about only a subset of the taps.

Table 3-4 Commonly Used Options for the `report_taps` Command

Option	Description
<code>-parasitics</code>	Reports the parasitics associated with each tap. When selected, the command writes the lumped RLC model type, the resistance value, the capacitance value, and the inductance value for each tap. See the <code>set_tap_package_model</code> man page for more information about tap parasitics.
<code>-static_current</code>	Reports the <code>static_current</code> attribute on taps during static rail analysis

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

Option	Description
-package	Reports the SPICE package model and its connection with taps. When selected, the report shows the top subcircuit of the SPICE package model and the connectivity between its ports and taps.
-nosplit	Prevents line-splitting and facilitates extracting information from the output report. By default, the report lists the design information in fixed-width columns. If the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column.
-result result_name	Reports the tap information in the specified rail or integrity result context. See Taps From Result Contexts for more information about using taps from the rail result context.

Example

The following report contains all the automatically inserted supply (either power or ground) pin taps, the net for a specific tap using attributes, and all the tap attributes.

```
pr_shell> report_taps [get_taps -filter type==auto_pg]
```

Name	Type	Net	Point	Layer	Object Name	Context
TAP_1	auto_pg	VDD	(123.534, 81.356)	(3) M2	/MyTop/VDD	session
TAP_2	auto_pg	VSS	(652.100 899.505)	(1) M1	/MyTop/VSS	session

```
pr_shell> get_attribute -class tap TAP_1 supply_net {"VDD"}
```

```
pr_shell> list_attributes -class tap -application
```

```
*****
```

```
Report : List of Attribute Definitions
```

```
Version: <Version>
```

```
Date : <Date>
```

```
*****
```

```
Properties:
```

```
  A - Application-defined      U - User-defined
```

```
  I - Importable from design/library (for user-defined)
```

Attribute Name	Object	Type	Properties	Constraints
full_name	tap	string	A	
layer	tap	int	A	
parasitics	tap	string	A	
position	tap	string	A	
supply_net	tap	string	A	

type	tap	string	A
object_name	tap	string	A
context	tap	string	A

The following example reports taps in the rail result context:

```
pr_shell> report_taps -result rail
```

Name	Type	Net	Point	Layer	Object Name	Context	Valid

rail_1	auto_pg	VDD	(10,20)(3)	M2	VDD	rail	Yes
rail_2	auto_pg	VSS	(30,30)(1)	M1	VSS	rail	No

The following example reports tap connections for the SPICE package model subcircuit PACKAGE:

```
pr_shell> report_taps -package
```

```
*****
Report : rail analysis tap package
Design : test
Version: <Version>
Date   : <Date>
*****
Package spice subcircuit: PACKAGE
Package ports           : PKG_PORT_VDD1 PKG_PORT_VDD2
(*) floating package ports
```

Name	Type	Net	Port

TAP_0	absolute_coord	VDD	PKG_PORT_VDD1
TAP_1	absolute_coord	VDD	PKG_PORT_VDD2

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Importing a Tap File](#)
- [Validating Taps](#)

Validating Taps

When creating taps, by default the `create_taps` command checks whether the physical location, specified by the `-point` and `-layer` options, touches a layout geometry. If the defined tap point does not touch any layout geometry, a warning message is issued. To place taps outside a shape without issuing a warning message, disable the checking using the `-nocheck` option. When the `-nocheck` option is enabled, the setting for the `rail_force_tap_check` variable will also be ignored.

When the tap creation is complete, use the `check_taps` command to check if the specified supply nets have valid taps created for rail analysis. If the specified supply net has only the invalid tap points, an error message is issued and the command fails.

Tap Attributes

During validation checking, each inserted tap object in the current session is marked with the `is_valid` attribute to indicate its validity status. When validation checking is enabled, the attribute value is YES for valid taps and NO for invalid taps. When validation checking is disabled, the attribute value is UNKNOWN for all taps.

To check the attribute value for a tap, use the `get_attribute [get_taps] is_valid` command.

The tool updates the tap attributes in the subsequent voltage drop and minimum path resistance analyses. Each tap belonging to the nets is checked against the extracted geometry. When the analysis is complete, the tool updates the tap validity status based on the rail analysis results.

Limitation:

The command checks only the first 100 tap points if they are specified using the `-point` and `-layer` options. If you want to perform tap validation checks beyond the limit of 100, you should do one of the following:

- Set the `rail_force_tap_check` variable to `true` before running the `create_taps` command.
- Group the taps into a tap file and import the tap file using the `create_taps -import` command.

Example

The following example sets the `rail_force_tap_check` variable to `true` and creates the 101st tap point using the same absolute coordinates. A warning message is issued since a tap point does not touch any layout geometry:

```
pr_shell> set rail_force_tap_check true
pr_shell> create_taps -supply_net VDD -layer 3 -point {100.0 200.0}
```

```
Warning: Tap point (100.0, 200.0) on layer 3 for net VDD does not \
touch any polygon (RAIL-305)
```

The following command creates a tap with the `-nocheck` option enabled. No warning message is issued.

```
pr_shell> create_taps -supply_net VDD -layer 3 \
               -point {100.0 200.0} -nocheck
```

The following command creates the 101st tap point. No warning message is issued since only the first 100 taps are checked:

```
pr_shell> create_taps -supply_net VDD -layer 3 -point {100.0 200.0}
```

The following example creates taps using the `create_taps` command with the `-nocheck` option. The `check_taps` command is then used to check the tap validity and to report if the specified supply nets have valid taps defined for rail analysis.

```
pr_shell> create_taps -nocheck -point {100 200} -layer M1 \
          -supply_net VDD
pr_shell> create_taps -nocheck -point {150 250} -layer M2 \
          -supply_net VSS
pr_shell> check_taps {VDD VSS}
Performing validity check for taps on VDD net.
Warning: Tap point {100 200} at layer M1 does not touch any layout
geometry (RAIL-305)
Performing validity check for taps on VSS net.
Error: no valid taps are found for supply net VDD (RAIL-333)
```

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Reporting Taps](#)
- [Finding Invalid Taps](#)

Finding Invalid Taps

If a tap is invalid at a location $\{x,y\}$ during tap creation, use the `-snap_distance` option of the `create_taps` command to find the tap a substitute location. The command searches the nearby layout geometries within the specified snap distance both horizontally and vertically on the same layer as defined by the `-layer` option. The tap is reinserted at the corner of a rectangle which is within the minimum distance to the specified xy location.

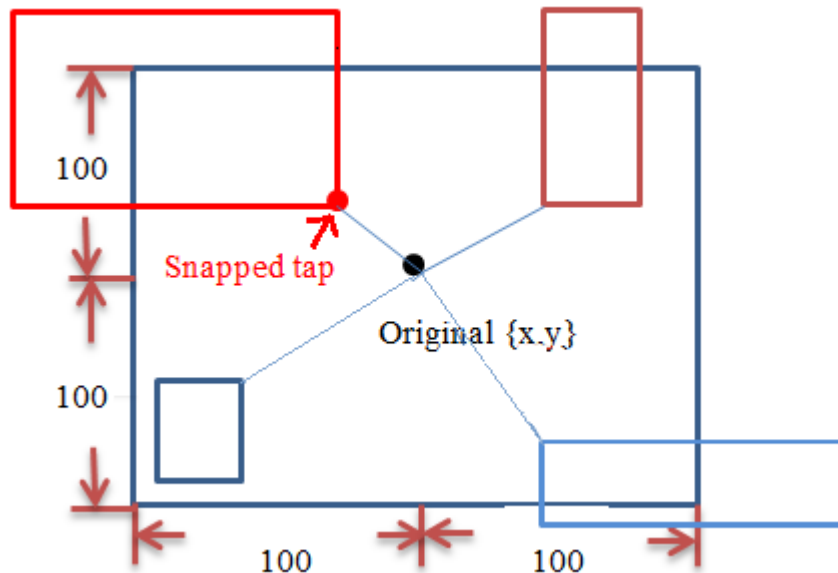
You can search for a substitute location only for the invalid taps that are specified by the `-point` and `-layer` options.

For example, you define a tap location with the following syntax:

```
pr_shell> create_taps -point {x,y} -layer {M1} -snap_distance 100
```

As shown in [Figure 3-2](#), the specified tap location $\{x, y\}$ does not touch any layout geometry. The red rectangle is the one closest to the specified location and its lower-left corner has the minimum distance among all the other rectangles. The tool reinserts the tap at the lower-left corner of the red rectangle.

Figure 3-2 Finding a Substitute Location for an Invalid Tap



Creating Taps Using Layout-Based Interactive Tap Tool

PrimeRail provides an interactive tap tool that allows you to insert or remove a tap on top of the desired physical object without opening a map. The interactive tap tool invokes the `create_taps` command to insert and validate the coordinates. If the location for the new tap is valid, the tool automatically updates the layout drawing. Otherwise, PrimeRail issues a warning.


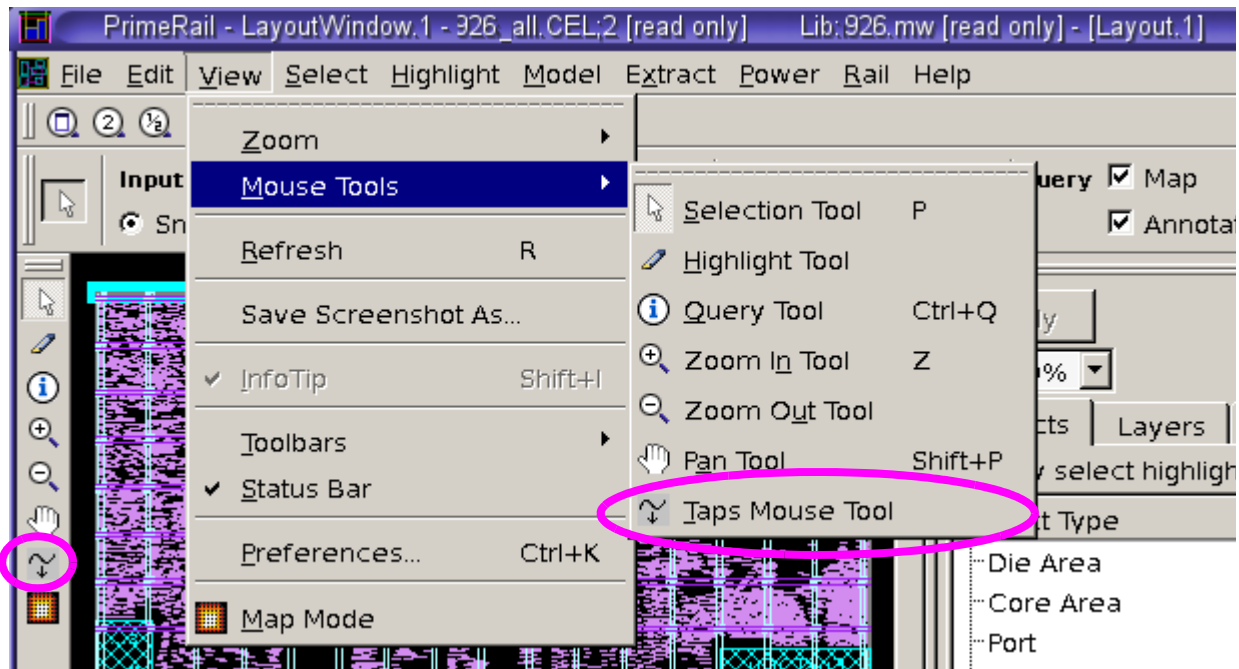
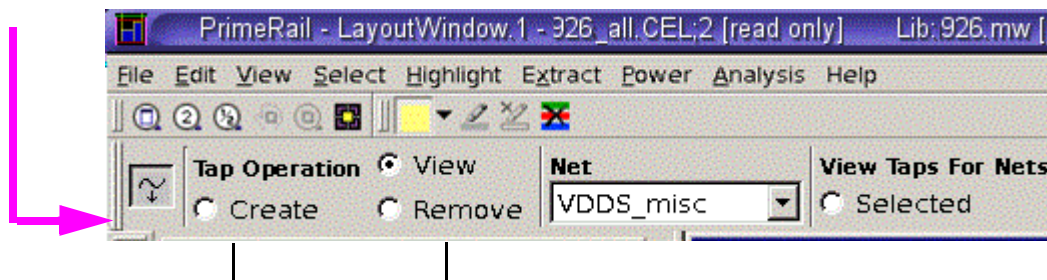
To invoke the interactive tap tool, choose View > Mouse Tools > Taps Mouse Tool or click the Taps button () on the Mouse Tools toolbar in the layout view. The tap-related options appear in the layout view (see [Figure 3-3](#)).

Figure 3-3 Invoking the Taps Mouse Tool



The Tap options are shown in the layout view.



Options for creating, viewing, or removing a tap

To create, view, or remove a tap, select the appropriate radio button in the Tap Operation option. Next, specify the net where the tap is to be inserted or removed in the Net option.

To control the visibility of the taps shown in the layout, use the View Taps For Nets option and choose to show taps on all nets or those on the selected nets only. To view taps by layers, use the View Taps For Layers option and choose to show taps on all layers or the selected layers only.

See Also

- [Creating Taps for Ideal Voltage Sources](#)
- [Validating Taps](#)

Setting TLUPlus Models

TLUPlus is a binary table format that stores the RC coefficients. The TLUPlus models enable accurate RC extraction results by including the effects of width, space, density, and temperature on the resistance coefficients.

To use TLUPlus models for RC estimation and extraction, you specify the following files:

- The map file, which matches the layer and via names in the technology file with the names in the ITF file.
- The maximum TLUPlus model file.
- The minimum TLUPlus model file (optional). Specifying the minimum TLUPlus file is necessary if the minimum and maximum operating conditions are different. If the TLUPlus models have derating factors, you must specify a minimum file, even if it is the same as the maximum file.

You specify these files by using the `set_tlu_plus_files` command. You can specify TLUPlus file names with or without their full paths. If you do not include the paths, the tool uses the search paths defined with the `search_path` variable.

Here is an example:

```
pr_shell> set_tlu_plus_files \  
          -tech2itf_map ./path/map_file_name.map \  
          -max_tluplus ./path/worst_settings.tlup \  
          -min_tluplus ./path/best_settings.tlup
```

For multicorner-multimode designs, use the `set_tlu_plus_files` command to specify the TLUPlus files for each scenario. The `set_tlu_plus_files` command only applies to the current scenario.

For more information about how to set up TLUPlus files using the `set_tlu_plus_files` command, see the command man page.

See Also

- [Extracting Supply Net Parasitics](#)

Specifying Voltage Settings

To specify voltages used in timing, power and current calculations, use the `set_voltage` command. To specify tap voltages for rail analysis, use the `set_supply_voltage` command. You can assign different voltages using the `set_voltage` and `set_supply_voltage` commands if necessary.

If both the `set_voltage` and `set_supply_voltage` commands are used to specify ideal voltages, PrimeRail uses the value specified with the `set_voltage` command for timing and power analysis, and the value specified with the `set_supply_voltage` command for rail analysis. PrimeRail reports if multiple voltages are connected to the same supply net.

If neither of the commands is used, PrimeRail uses the voltage from the operating condition settings for timing and power analysis, and the minimum voltage from all the cell instances connected to the supply net, considering the worst case, for rail analysis. For example, assume there are three cell instances that connect to the supply net VDD. One cell instance has voltage 1.0 and the other two cell instances have voltage 1.1. In this case, PrimeRail uses 1.0 for VDD in rail analysis.

This section describes the following topics:

- [Setting Voltages](#)
- [Setting Rail-Specific Voltages](#)
- [Reporting Supply Voltages](#)

Setting Voltages

Run the `set_voltage` command (or choose Power > Supply Net Voltage in the GUI) to define the voltage on the supply nets, so the parts of the design powered by these supply nets are timed, optimized and analyzed at the specified voltage. The command automatically propagates through power management cells. Therefore, the control power and ground nets and the virtual power and ground nets have the same voltages. This means that the supply nets that are controlled by power switches have the same voltage as their associated main supply nets.

Set voltage constraints before running the `update_currents` command. Doing so guarantees that the same voltage is used in the timing and power analysis and rail analysis. In cases where you want to set voltages in the timing and power analysis that are different from those in rail analysis, use the `set_supply_voltage` command to decouple the voltage used for timing and power analysis from the voltage used for rail analysis.

PrimeRail supports the voltages set in all of the following ways with a precedence order from high to low:

- `set_voltage` on a power or ground pin
- `set_voltage` on a supply net
- `set_operating_conditions` applied at the design level
- Default supply voltage in the library cell definition

For more information about the `set_voltage` command, see PrimeTime documentation.

See the `set_operating_conditions` command man page for details about the precedence rules for deriving cell instance voltages for timing and power analysis.

See Also

- [Specifying Voltage Settings](#)
- [Reporting Supply Voltages](#)

Setting Rail-Specific Voltages

Use the `set_supply_voltage` command to set rail-specific voltages if you want to perform rail analysis with different voltages from those used during timing and power analysis. To expand the voltages through power management cells, run the `get_supply_nets -expand` command.

Use the `-expand` option to get a collection of the supply nets that are connected to the supply net VDD through the power management cells. During rail analysis, these supply nets are connected together through the power management cells and are treated as a single power and ground net when voltage drop analysis is performed.

Example

The following example specifies the worst-case voltages for rail analysis.

```
pr_shell> set_supply_voltage 1.2 -object_list VDD
```

The following command sets the voltage to 1.2 V for VDD and all the supply nets connected to VDD through power management cells.

```
pr_shell> set_supply_voltage 1.2 -object_list [get_supply_nets \
    -expand VDD]
```

See Also

- [Specifying Voltage Settings](#)
- [Reporting Supply Voltages](#)

Reporting Supply Voltages

When rail analysis is complete, use the `report_supply_voltage` command to show the supply voltage used in rail analysis. If you run the command before rail analysis, the command reports settings that are defined with the `set_supply_voltage` command.

For example:

```
pr_shell> report_supply_voltage [collection_of_supply_nets]
```

For each supply net in the argument list, the report shows the supply net name, supply voltage, the voltage source (derived or assigned), and the number of cell instances connected to this supply net. If the `collection_of_supply_nets` argument is not provided, PrimeRail reports voltages for all the supply nets in the design.

See Also

- [Specifying Voltage Settings](#)

Setting Operating Conditions

To improve power analysis accuracy, use the `set_operating_conditions` command to define the operating conditions (or environmental characteristics). When process factor, operating temperature, and operating voltage deviate from their nominal values, the tool uses delay scaling among multiple libraries.

The logic library can specify multiple sets of chip operating conditions (process, voltage, and temperature). To get a list of the operating conditions available in a particular library and to view their characteristics, use either the `report_operating_conditions` or `report_lib -operating_condition` command.

To set the operating conditions for timing analysis and optimization, use the `set_operating_conditions` command. When you run this command, you can specify either a maximum operating condition only or both a minimum and a maximum operating condition (`-min` and `-max` options).

For more information about setting operating conditions, see the PrimeTime documentation.

See Also

- [Analyzing Power and Updating the Currents](#)

Reading SDC Files

PrimeRail supports the Synopsys Design Constraints (SDC) file to obtain clock definition data. Run the `read_sdc` command to read in an SDC script that contains commands for running timing and power analysis.

For more information about SDC files, see the *Using the Synopsys Design Constraints Format Application Note*.

See Also

- [Analyzing Power and Generating Current Waveforms](#)

Loading Signal Parasitics Information

PrimeRail needs the signal parasitic capacitance and resistance information for rail analysis; it accepts parasitic data in Synopsys Binary Parasitic Format (SBPF) or Standard Parasitic Exchange Format (SPEF).

Use the `read_parasitics` command (or choose Power > Read Signal Parasitics in the GUI) to read in the net parasitics information from an SPEF or binary parasitics file (SBPF). The command also uses the parasitic information to annotate the currently linked design. For example, annotated parasitics are used to compute cell and net delays with more accuracy, and signal net parasitics are used to analyze power consumption. For instance, switching power is calculated using the following equation:

$$P = \frac{1}{2}(C \times V^2 \times f)$$

where C is the net (load) capacitance, V is the voltage, and f is the switching frequency.

For a detailed description about reading and removing parasitic information, see the related chapter in PrimeTime documentation suite.

To check the net parasitics back-annotated on the current design, use the `report_annotated_parasitics` command.

See Also

- [Static Rail Analysis](#)

Loading Timing Constraints

To read in a timing constraints file that contains Synopsys Design Constraints (SDC) commands, use the `read_sdc` command or choose Power > Read SDC in the GUI. If the SDC file does not contain unit settings, the tool derives the unit settings from the main library. For more information about logic libraries, see [Logical Libraries](#).

For details about the SDC commands, see the *Using the Synopsys Design Constraints Format Application Note*.

See Also

- [Analyzing Power and Generating Current Waveforms](#)

Reading Switching Activity Information

PrimeRail reads switching activity information for power analysis. The switching activity input can be in various formats, such as VCD, SAIF, `set_switching_activity` commands, or the tool's default. Analysis using the switching activity provides the most accurate results. The more accurate the switching activity, the more accurate the power analysis. Wherever possible, you should provide switching activity generated from simulation. You can provide either RTL or gate-level VCD or SAIF files for power analysis.

When neither a VCD nor a SAIF file is available, use the vector-free power analysis mode and provide switching probabilities at the primary stimulus points in the design. These primary stimulus points can be primary inputs or I/Os and black box outputs. PrimeRail then propagates those switching probabilities forward through the circuit to derive the switching probabilities of the internal signals. Note that the calculated signal probabilities might differ substantially from the actual switching probabilities due to the effects of the re-convergent fanout. If no switching probabilities are available, the tool uses a value of 50% for all stimulus points.

For multivoltage designs, VCD or SAIF files include simulation activity of switch control signals. If neither VCD nor SAIF file is available, calculating power and current waveforms might lead to incorrect results because the default activity setting might be too high for switch control signals. You need to adjust the switching activity accordingly in those cases.

For VCD or SAIF activity inputs, make sure that the hierarchy in the VCD file and the design match. This can be accomplished by providing the design hierarchy in the VCD file with the `-strip_path` option of the `read_vcd` command. It is recommended that you manually check the hierarchy of the activity file by using the `report_vcd_hierarchy` command or reading the VCD file into nWave. The `strip_path` argument is case-sensitive.

See Also

- [Annotating Switching Activity](#)
- [Reporting Switching Activity](#)
- [Analyzing Power and Generating Current Waveforms](#)

Annotating Switching Activity

The power consumption of a design depends on the switching activity of the nets and cell pins of the design. The higher the switching activity is, the more power the design consumes. To calculate the average power, you must first annotate the design with the switching activity as described in this section.

The required switching activity information can be annotated on design objects, such as nets, ports, or pins. You can annotate the following types of the switching activity:

- Simple switching activity on design nets, ports, and cell pins

Simple switching activity consists of the static probability and the toggle rate. The static probability is the probability that the value of the design object has logic value 1. It defines the percentage of time when the signal is at a high state. By default, the value is 0.5, which means the signal is high for half the time and low for half the time. The default might be appropriate for data bus lines but might not be appropriate for some signals, such as reset or test_enable signals.

The toggle rate is the average 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.

You can annotate the switching activity with one of the following methods.

- [Annotating Switching Activity Using VCD Files](#): Generate a VCD file using RTL or gate-level simulation, and read the VCD file using the `read_vcd` command.
- [Annotating Switching Activity Using SAIF Files](#): Generate one or more SAIF files using RTL or gate-level simulation, and read the SAIF files using the `read_saif` or `merge_saif` command.

- **Annotating Switching Activity Using the `set_switching_activity` Command:** Use the `set_switching_activity` command on individual design objects to annotate switching activity.
- Use the default switching activity.

Annotating Switching Activity Using VCD Files

A VCD file records logic value changes in a design. You can provide either RTL or gate-level VCD and read it into memory using the `read_vcd` command. For average power analysis, when you specify the VCD file, the tool automatically derives switching activity from the VCD file. It then converts the switching activity data into toggle rates that are used to calculate the average power of every cell.

When you use the `read_vcd` command for average power analysis, the tool checks the percentage of nets that are annotated, and extracts the toggle information for the nets available in the VCD file. The `read_vcd` command reads the specified file and applies the activity data to the attributes on the design instances. Use the `report_switching_activity` or `get_switching_activity` command to view the activity data derived from the VCD file.

The tool applies the default annotation to primary ports that are not annotated. It uses zero-delay simulation to propagate the activity to the annotated nets in the design.

For more information about specifying a VCD file, see the *PrimeTime PX User Guide*.

Annotating Switching Activity Using SAIF Files

The SAIF file contains statistical data to describe switching probabilities by capturing the probability when a signal (cell or block output) changes state over the period. When PrimeRail reads an incomplete SAIF file (such as, missing switching probabilities for some signals), it propagates signal probabilities forward to the missing signals where possible. This is applied when performing a vector-free power calculation for static rail analysis.

You can specify a gate-level or RTL SAIF file. The `read_saif` command reads the specified file and applies the activity data to the attributes of the design instances.

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

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

Annotating Switching Activity Using the `set_switching_activity` Command

When the toggle-rate information generated from simulation is not available, you can specify the switching activity of the primary inputs by using the `set_switching_activity` command.

To annotate the switching activity on nets, ports and pins, use the `-toggle_rate` and `-static_probability` options. The toggle rate represents the number of glitch-free transitions, either from 0 to 1 or 1 to 0, during the time period specified by the `-period` or `-base_clock` option. The time units used for the toggle rate are the same as those in the target library.

The specified toggle rate is converted to an absolute value by dividing it by the period. If the period is not specified, the default is 1 library time unit. In the following example, net1 is at logic 1 for 20 percent of the time. It transitions between logic 0 and 1 at an average of 10 times in 1,000 time units. The toggle rate applied to net n1 is 10/1000, or 0.01:

```
pr_shell> set_switching_activity [get_net n1] \
        -static_probability 0.2 -toggle_rate 10 -period 1000
```

The options of the `set_switching_activity` command let you do the following:

- Specify state-dependent toggle rates for pins (that is, the rates that depend on the states of specified input pins)
- Specify the ratio of rise transitions to fall transitions for pins characterized with both rise and fall internal power
- Specify path-dependent toggle rates for pins (that is, the rates that depend on which input pin is toggled)
- Specify the static probability for signals. The fraction of time that signals are in specified states.

To specify the time period for the toggle count, use either the `-clock_domains` or the `-base_clock` option. The `-clock_domains` option indicates the toggle rate relative to the `power_base_clock` attribute; while the `-base_clock` option indicates the toggle rate relative to the named clock. If neither of these options are applied, then the period is assumed to be the library time unit.

The `power_base_clock` attribute is set for all the nets in the design and references the clock domain where they belong. If they belong to multiple clock domains, the `power_base_clock` attribute is set to the fastest of the clocks. Specifying the `-base_clock` option does not modify the `power_base_clock` attribute value; it controls the toggle rate applied to the node.

The `-clock_domains` option is a filter. When you use this option, the switching activity is set only on the specified clock domain. The toggle rate unit is determined by the value of the `power_base_clock` attribute of the objects in the clock domain specified by the `-clock_domains` option.

The `set_switching_activity` command does not override the activity of the clock input `clk1` if you specify the `-clock_domains` option in combination with the `-type` option. However, if you do not specify the `-clock_domains` option, the `set_switching_activity` command has higher precedence than the `create_clock` command.

For more information about the `set_switching_activity` command, see the command man page.

Reporting Switching Activity

The `report_switching_activity` command reports switching activity data from various sources. You can report the switching activity after the `read_saif` or `read_vcd` command is executed.

As shown in [Example 3-1](#), 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 certain types of drivers. When nets are driven by multiple drivers, the net is counted only one time. Also, for nets with multiple drivers, the nets have the following decreasing order of priority:

- Primary input
- Black box
- Tristate
- Sequential
- Combinational

Example 3-1 Example Report Generated by the `report_switching_activity` Command

```
pr_shell> report_switching_activity
report_switching_activity
*****
Report : Switching Activity
Design : test1
Version: <Version>
Date : <Time>
*****
```

Object	File (%)	SSA	SCA	Clock	Default	Propa- gated	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	
Combi-	2012 (100.0%)	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	2012	National

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

To display the design objects that do not have user-annotated switching activity information, use the `-list_not_annotated` option. Use the following attributes to get the value of toggle rate and static probability:

```
toggle_count
toggle_rate
glitch_rate
static_probability
```

Reading Electromigration Rules

PrimeRail uses electromigration rules to perform electromigration analysis on the power supply network. The rules include width- and temperature-dependent current limits, and all other technology and library information. When you open the design with the `open_mw_cel` command, by default PrimeRail reads the electromigration rules from the design library. If no rules are stored in the library, PrimeRail reads the electromigration rules stored in the technology file instead. Or you can read in the electromigration rules from an electromigration rule file in either ALF or TLUPlus format, using the `read_em_rules` command.

Run the `read_em_rules -format rule_format` command (or choose Rail > Read EM Rules in the GUI) to read the electromigration rule information into the current session. Set the `rule_format` keyword to

- `mw_cached_em_rule` to use the electromigration rules stored in the library.
- `mw_tech_file` to use the electromigration rules stored in the technology file.
- `alf` to read the electromigration rules from the specified ALF file.

When electromigration rules are specified by the ALF file, the file might contain library information in addition to the electromigration rules. If this is the case, the command does not check or save the additional library information since the `read_em_rules` command performs only limited syntax checking on the non-electromigration portions of the electromigration rule file.

- `tluplus` to read the electromigration rules from the specified TLUPlus file. You must also specify the name of the layer mapping file using the `-design2itf_map` option. The mapping file defines the mapping between the layers in the design database and the ITF file.

Unlike a traditional electromigration rule which defines temperature and current density limits, a TLUPlus file contains advanced electromigration rules to calculate electromigration violations by considering metal length and width dependence, electron current direction and power rail recognition.

For example, a TLUPlus file defines a cross section profile of the process and consists of an ordered list of conductor and dielectric layer definition statements. The layers are defined from the topmost dielectric layer to the bottommost dielectric layer, excluding the substrate, in a way that is consistent with the physical manufacturing process.

Note:

Regardless of the electromigration rule input format, the electromigration rules establish current density limits for the layout geometry on a device layer.

You might contact your foundry for an EM file with advanced electromigration rules.

See Also

- [Performing Static Voltage Drop Analysis](#)
- [Performing Static Electromigration Analysis](#)
- [Reporting Electromigration Rules](#)

Reporting Electromigration Rules

Use the `report_em_rules` command to check the electromigration rules that are currently loaded in the session. The electromigration rules can come from two sources: rules specified within technology file, or the rules that are read in with the `read_em_rules` command.

Electromigration rules establish layer-by-layer current density limits for the layout geometries. The rules can establish limits on the average current, peak current, and rms (root mean square) current. Electromigration rules are typically temperature and width dependent and other dependencies, such as route length.

Example

```
pr_shell> report_em_rules

*****
Report : EM rules
Design : test
Version: <Version>
Date   : <Date>
*****
EM rules are in tluplus format.
Reporting EM rules in file EM_RULE_dump.txt
1
```

The following is an example of EM_RULE_dump.txt file

```
Source: Milkyway
Units:
  width : 1e-06 Meter
```

```

area      : 1e-12 Meter**2
current: 0.001 Amp
Layer: metall
Mode : average
Temperature
+-----+-----+-----+-----+
| Width | -40 | 25 | 105 | 125 |
+-----+-----+-----+-----+
| 0.100 | 0.198 | 1.980 | 0.198 | 1.980 |
| 1.000 | 0.198 | 1.980 | 0.198 | 1.980 |
+-----+-----+-----+-----+
                                (I,J)=Max_Current

Layer: metal2
Mode : average
Temperature
+-----+-----+-----+-----+
| Width | -40 | 25 | 105 | 125 |
+-----+-----+-----+-----+
| 0.100 | 0.254 | 2.540 | 0.254 | 2.540 |
| 1.000 | 0.254 | 2.540 | 0.254 | 2.540 |
+-----+-----+-----+-----+
                                (I,J)=Max_Current

Layer: vial
Mode : average
Temperature | Max_Current
+-----+-----+
| -40 | 0.060 |
| 25 | 0.06 |

```

See Also

- [Reading Electromigration Rules](#)

Creating Rail Setup Data in the IC Compiler Tool

The PrimeRail and IC Compiler tool read and write data to the Milkyway database. You can run the `analyze_rail -script_only` command in the `icc_shell` to generate the settings required for running rail analysis in the standalone version of PrimeRail. You must use the `analyze_rail -script_only` command along with the analysis mode, such as voltage drop or electromigration. By default, the `analyze_rail -script_only` command writes data to the `synopsys_rail_setup` directory under the PrimeRail work directory. The output files that are saved to the directory are

- The PrimeRail command file
- The SDC file
- The signal parasitic file in the SPEF format

- The `analyze_rail.tcl` file, which includes the commands that are required for running rail analysis

To use the generated `analyze_rail.tcl` file in the standalone version of PrimeRail, choose File > Execute Script in the PrimeRail GUI.

For more information about generating rail analysis settings in the IC Compiler tool, see the In-Design rail analysis chapter in the *IC Compiler Implementation User Guide*.

4

Preparing and Checking Design Data

PrimeRail reads and writes data to the Milkyway or IC Compiler II design library where saves your design and other related data. If you have a design database in the Library Exchange Format/Design Exchange Format (LEF/DEF), translate these database into the Milkyway format using the PrimeRail database converter.

Alternatively, you can create rail macro models with these LEF/DEF data, and then perform full-chip rail analysis on the created macro models. For more information about creating rail macro models, see [Rail Macro Models for Milkyway Designs](#).

This chapter provides the following sections:

- [Setting Up the Technology File](#)
- [Checking CCS Power Libraries](#)
- [Converting LEF/DEF Cell Data](#)
- [Setting Up the Design](#)
- [Before Opening a Design](#)
- [Opening Milkyway Designs](#)
- [Opening IC Compiler II Designs](#)
- [Checking Design Readiness](#)
- [Reporting Design Checking Results](#)

Setting Up the Technology File

By default, PrimeRail reads the physical library information, including design libraries or reference libraries, from the database. The database contains not only leaf-level physical cell information and technology information, but also design-specific physical information, such as the placement and routing of the design.

To analyze voltage drop and current density violations in your design, you must have a certain set of data in the technology file. The technology file defines the characteristics of a cell library, such as units, layers, design rules, and capacitance models. The settings defined in the Technology, Layer, and ContactCode sections of the technology file must be correct for performing rail analysis.

The following sections describe the necessary settings in the technology file for a successful rail analysis run. If you need more information about how to create and load a technology file in the IC Compiler environment, see the *Synopsys Technology File and Routing Rules Reference Manual*.

- [Technology Section](#)
- [Layer Section](#)
- [ContactCode Section](#)

Note:

PrimeRail does not make changes to the design (that is, database). You must create or load a technology file in the IC Compiler or IC Compiler II tool before reading the database in PrimeRail.

Technology Section

The following power-related units must be defined in the technology section of your technology file:

- The voltage unit and precision, using the `unitVoltageName` and `voltagePrecision` attributes
- The current unit and precision, using the `unitCurrentName` and `currentPrecision` attributes
- The power unit and precision, using the `unitPowerName` and `powerPrecision` attributes

These fields are in bold in the following example:

Example: Units in the Technology File

```
Technology      {
    name          =      CMOS18FV"
    dielectric     =      3.942e-05
    unitTimeName   =      "ns"
    timePrecision  =      1000
    unitLengthName =      "micron"
    lengthPrecision =      1000
    gridResolution =      20
    unitVoltageName   =      "v"
    voltagePrecision =      1000
    unitCurrentName  =      "mA"
    currentPrecision =      10000
    unitPowerName    =      "mW"
    powerPrecision   =      10000
    unitResistanceName =      "ohm"
    resistancePrecision =      10000
    unitCapacitanceName =      "pf"
    capacitancePrecision =      1000000
    unitInductanceName =      "nh"
    inductancePrecision =      1000
}
```

The valid units for voltage, current, and power are volt (V), ampere (A), and watt (W), respectively, with or without any of the following prefixes:

```
f = femto
p = pico
n = nano
u = micro
m = milli
```

Layer Section

In each layer section for metals and vias in your technology file, specify the maximum threshold current density that the layer can safely sustain using the `maxCurrDensity` attribute. This information is process-dependent. Consult your foundry or library vendor for the appropriate values.

The maximum current density values for each metal and via layer are required to perform error checking during electromigration diagnosis. When segments of metal rails or via objects are found with a current density higher than the specified value, PrimeRail flags these as errors and reports them to an ASCII file.

Table 4-1 shows the units and default values of the `maxCurrDensity` attributes.

Table 4-1 Units and Default Values of the `maxCurrDensity` Attribute

Layer type	Units	Default
Conducting metal layer	amp/cm	1e+23 amp/cm
Via layer	amp/cm ²	1e+23 amp/cm ²

The following example shows a layer section of the technology file with the `maxCurrDensity` setting in bold:

```

Layer "M1" {
    layerNumber           = 3
    maskName              = "metal1"
    isDefaultLayer        = 1
    defaultWidth          = 0.6
    minWidth              = 0.6
    minSpacing            = 0.6
    fatThinMinSpacing     = 3
    fatFatMinSpacing      = 6
    pitch                 = 1.4
    fatWireThreshold      = 30
    maxSegLenForRC        = 1000
    blink                 = 0
    visible               = 1
    selectable            = 1
    lineStyle             = "solid"
    pattern               = "slash"
    color                 = "cyan"
    fillPattern           = "outlineStippleFill"
    unitMinHeightFromSub  = 0.8
    unitNomHeightFromSub  = 0.8
    unitMaxHeightFromSub  = 0.8
    unitMinThickness      = 0.7
    unitNomThickness      = 0.7
    unitMaxThickness      = 0.7
    maxDeltaWidth         = 0.7
    nomDeltaWidth         = 0.7
    minDeltaWidth         = 0.7
    maxCurrDensity       = 10.00
}

```


Note:

The maximum current density (indicated in bold in the example) is multiplied by the defined current precision value before it is stored in the database. If the scaled value is too large because of a very high `maxCurrDensity` value or a very high `currentPrecision` setting, overflow occurs. This occurs when the value exceeds the value of $2E31-1$.

To solve this problem, reduce the `currentPrecision` setting in the technology section until no overflow occurs.

ContactCode Section

In the contact code section of your technology file, make sure the upper and lower layers for poly contact, contact, and via layers are defined. The poly contact layer is the layer between poly and metal. The contact layer is the layer between diffusion and metal. The via layer is between metal_N and metal_N+1.

```

ContactCode      "contactcode_polycont" {
    contactCodeNumber      = 1
    cutLayer                = "POLYCONT"
    lowerLayer              = "POLY1"
    upperLayer              = "METAL1"
    isDefaultContact        = 1
    cutWidth                = 0.16
    cutHeight               = 0.16
    upperLayerEncWidth      = 0
    upperLayerEncHeight     = 0
    lowerLayerEncWidth      = 0.07
    lowerLayerEncHeight     = 0.07
    minCutSpacing           = 0.18
}
ContactCode      "contactcode_contact" {
    contactCodeNumber      = 2
    cutLayer                = "CONT"
    lowerLayer              = "N-Diff"
    upperLayer              = "METAL1"
    isDefaultContact        = 1
    cutWidth                = 0.16
    cutHeight               = 0.16
    upperLayerEncWidth      = 0
    upperLayerEncHeight     = 0
    lowerLayerEncWidth      = 0.07
    lowerLayerEncHeight     = 0.07
    minCutSpacing           = 0.18
}
ContactCode      "vial" {
    contactCodeNumber      = 3
    cutLayer                = "VIA12"
    lowerLayer              = "METAL1"
    upperLayer              = "METAL2"

```

```

        isDefaultContact          = 1
        cutWidth                  = 0.19
        cutHeight                 = 0.19
        upperLayerEncWidth        = 0.005
        upperLayerEncHeight       = 0.05
        lowerLayerEncWidth        = 0.01
        lowerLayerEncHeight       = 0.05
        minCutSpacing             = 0.29
        unitMinResistance          = 0.00102
        unitNomResistance          = 0.00102
        unitMaxResistance          = 0.00102
    }

```

See Also

- [Preparing and Checking Design Data](#)

Checking CCS Power Libraries

PrimeRail requires the information from the CCS power libraries to accurately calculate the voltage and electromigration violations in the design, such as intrinsic parasitics (including channel resistance and intrinsic capacitance) and dynamic current waveforms at the power and ground ports. Use the `check_ccs_power` command to check the availability and accuracy of the required information for the specified library or library cell. Doing this ensures that the calculated voltage and current values are well correlated with the ones from a SPICE tool.

Note:

Before running the `check_ccs_power` command, you must open the design using the `open_mw_cel` command.

The `check_ccs_power` command checks if the library contains the following information:

- The power and ground pin information for all the standard cells; and the pin-specific power and current waveform information for the multivoltage and multisupply cells
- The state-dependent intrinsic capacitances, including the ON and OFF resistance specification
- The current waveforms for the specified input pins and the power arcs from the input pins to the output pins
- The I-V curve tables for power-gating transistors, like positive I-V curves for the header cells and negative I-V curves for the footer cells. For switch cells with the complicated logic, I-V curves are characterized on the transistors, not the external switch pins.

The `check_ccs_power` command also checks if

- The intrinsic capacitance values are equal to or smaller than 10 pF.
- The ratio of the channel resistances between the VDD and the GND nets is equal to or smaller than 100.
- The peak current range for a standard cell in the CCS power library is equal to or smaller than 1 mA.
- The switching current ratio between VDD and VSS nets is equal to or larger than 100.
- The ratio between switching and non-switching currents for the same input pin is equal to or smaller than 10.
- The leakage current is above 0.5 uA.
- The dimension for the slew and capacitance table meets the minimum width. For instance, a 5x5 table is recommended. The slew index should be at least 100 ps, whereas the capacitance index should be at least 50 fF.
- The power management cell I-V curves and linear resistances are between 10 and 1000 Ohm.

See Also

- [Preparing and Checking Design Data](#)

Settings for Generating Macro Cell Waveforms

In a CCS power library, standard cells have PG pin currents associated with their different switching power arcs in the event file. For example, a standard cell has both the input pin switching event and its associated output pin switching event.

To generate PG pin current waveforms for both the input and output switching events in a macro cell, define the following attribute in the CCS power library:

```
library ("macro_cell_library") {
  define (current_waveform_exclude_internal_current, cell, string) ;
  ...
  cell (test) {
    current_waveform_exclude_internal_current : "excluded";
    ...
  }
}
```

By default, PrimeRail generates current waveforms only for the output switching event.

See Also

- [Checking CCS Power Libraries](#)

Converting LEF/DEF Cell Data

To prepare a reference library from LEF cell data, you import the data into PrimeRail and translate that data into the Milkyway database. To create a Milkyway design library from DEF cell data, you read in the netlist information and annotate the design data to a reference library.

Note:

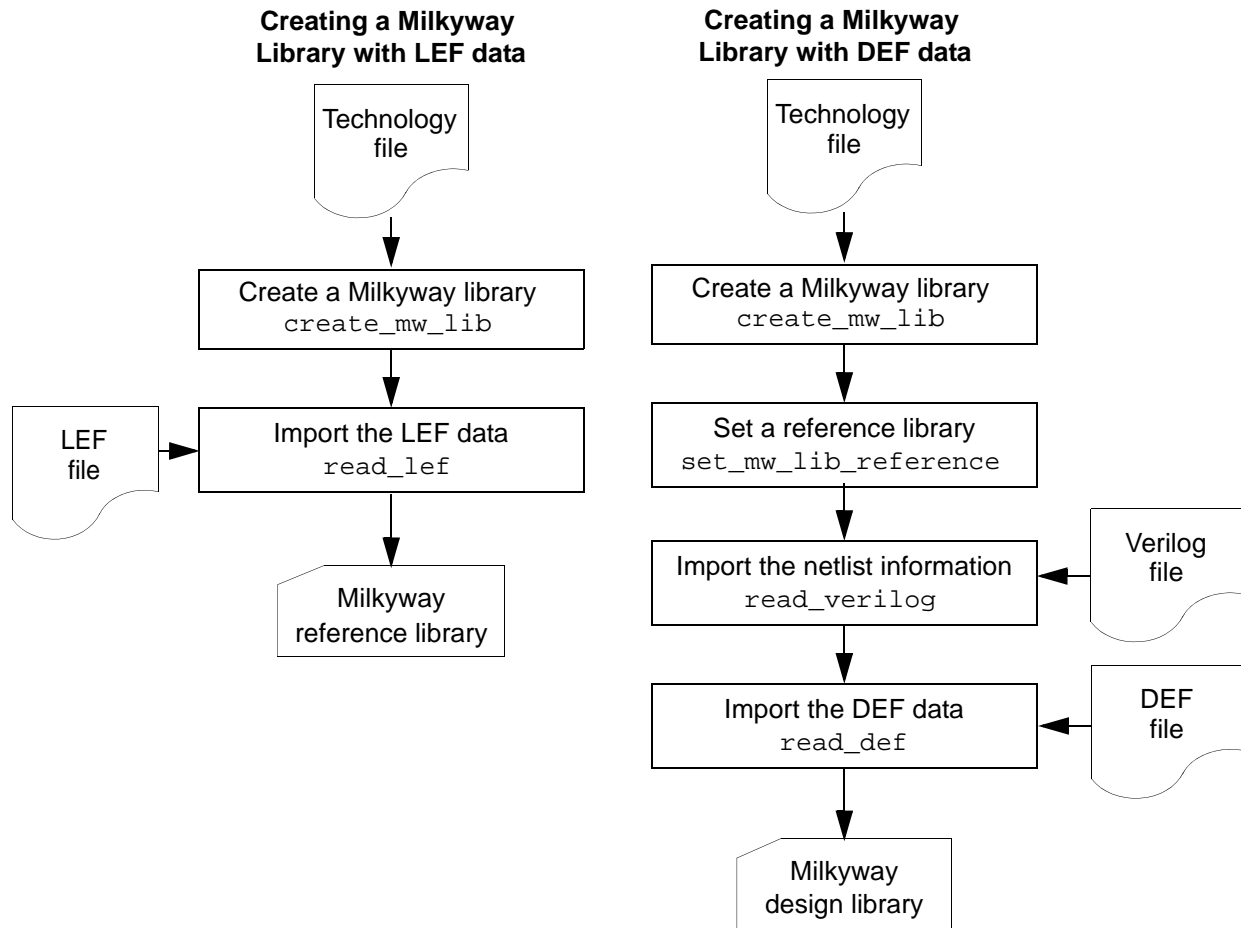
Reading in the LEF and DEF cell data with the `read_lef` and `read_def` command requires a Milkyway license key in addition to the PrimeRail license.

You can also create a Milkyway library with the LEF/DEF cell data within the Milkyway environment. For more information, see the *Library Data Preparation for IC Compiler User Guide*.

[Figure 4-1](#) describes the steps for translating the LEF/DEF data into the Milkyway library in PrimeRail. The details are described in the following sections:

- [Converting LEF Files into the Milkyway Format](#)
- [Converting DEF Files into the Milkyway Format](#)
- [Importing Multiple DEF Files for Hierarchical Designs](#)

Figure 4-1 Translating the LEF/DEF Data into the Milkyway Format



Converting LEF Files into the Milkyway Format

The LEF format defines the elements of a process technology and the associated library of cell models, including layer, via, placement site type and macro cell definitions.

To convert the LEF data into the Milkyway format,

1. Run the `create_mw_lib` command to create a Milkyway reference library using the specified technology file.

For example,

```
pr_shell> create_mw_lib -technology ./libs/milkyway/test.tf reflib
```

2. Read the LEF file using the `read_lef -lib_name` command.

For example,

```
pr_shell> read_lef -lib_name reflib -cell_lef_files \
           {test.lef ram_test.lef}
```

When the process is complete, PrimeRail creates CEL views for all the cells and saves them into the Milkyway reference library created at Step 1.

See Also

- [Converting LEF/DEF Cell Data](#)

Converting DEF Files into the Milkyway Format

The DEF format defines the elements of a specific design that are related to physical layout, including the placement and routing information, design netlist, and design constraints.

To convert the DEF cell data into the Milkyway format,

1. Run the `create_mw_lib` command to create a Milkyway design library using the specified technology file.

For example,

```
pr_shell> create_mw_lib -technology ./libs/tech/milkyway/test.tf mwlib
```

2. Set up a Milkyway reference library for the Milkyway design library created at Step 1 using the `set_mw_lib_reference` command. You must close the Milkyway design library with the `close_mw_lib` command before running the `set_mw_lib_reference` command.

To list the reference libraries that are defined for a particular design library, run the `report_mw_lib -mw_reference_library` command.

For example,

```
pr_shell> close_mw_lib mwlib
pr_shell> set_mw_lib_reference -mw_reference_library {./reflib} mwlib
pr_shell> report_mw_lib -mw_reference_library mwlib
```

3. Open the Milkyway design library created at Step 1 with `open_mw_lib` command.

For example,

```
pr_shell> open_mw_lib mwlib
```

4. Read in the netlist information from the specified Verilog files with the `read_verilog` command.

For example,

```
pr_shell> read_verilog finish.v
```

5. Annotate the CEL view with physical design data taken from the specified DEF files with the `read_def` command.

```
pr_shell> read_def finish.def
```

The tool reads the DEF files, converts the data into a CEL view and stores the data into the specified Milkyway library.

6. Save the Milkyway cell using the `save_mw_cel` command. This is to ensure the design data is saved to the disk and ready for the subsequent rail analysis.
7. Save and close the cell and the library using the `close_mw_cel` and `close_mw_lib` commands. You should always close the design when data preparation is finished, and reopen the design for the subsequent analysis step. This is to ensure proper setup and libraries are loaded for the design.

See Also

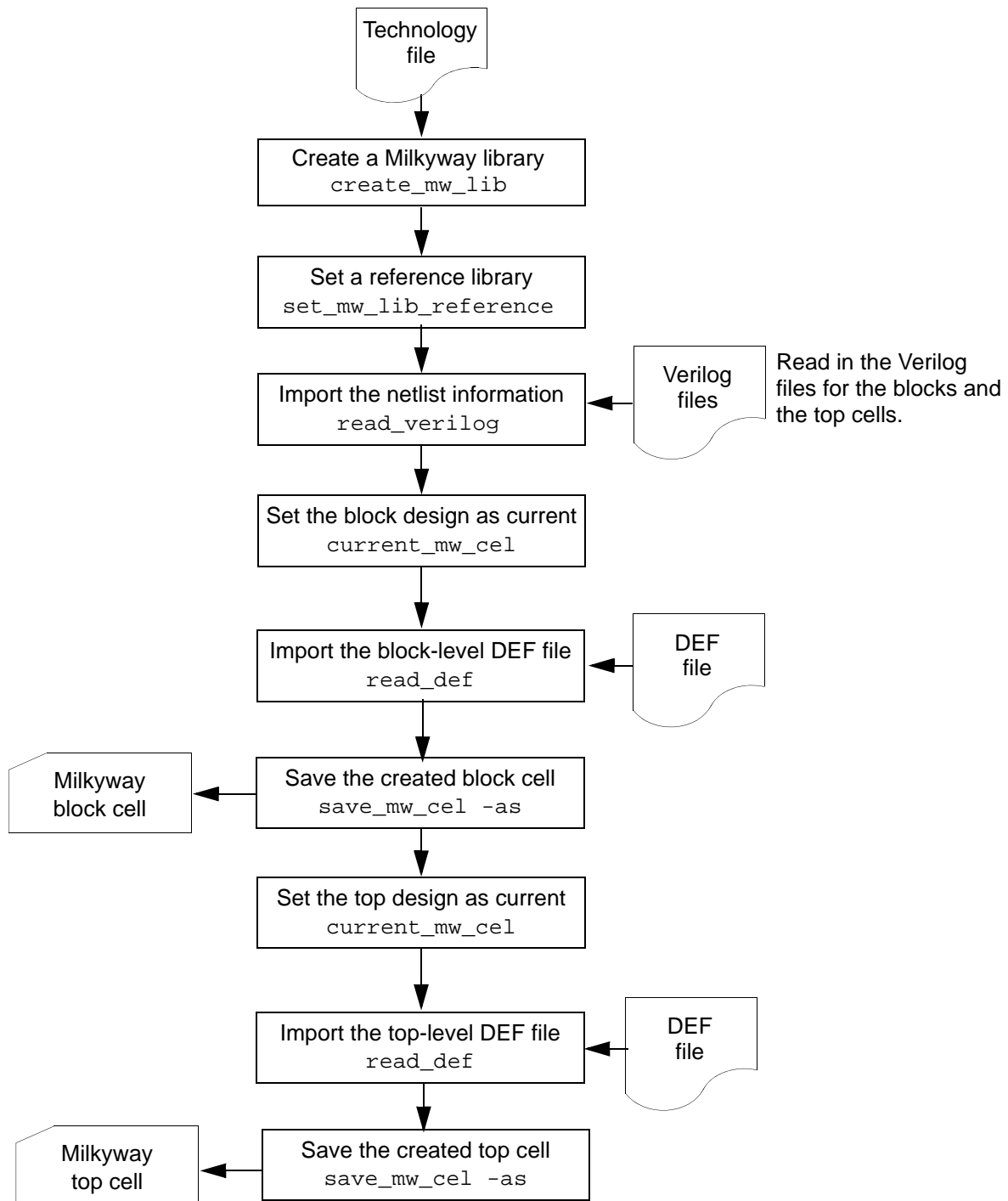
- [Converting LEF/DEF Cell Data](#)

Importing Multiple DEF Files for Hierarchical Designs

When you translate multiple DEF files into the Milkyway format for a hierarchical design, first import the DEF file for the block-level cell, then the one for the top-level cell.

[Figure 4-2](#) describes the steps for creating a Milkyway library with multiple DEF files for a hierarchical design.

Figure 4-2 Importing Multiple DEF Files for Hierarchical Designs



To convert the DEF cell data into the Milkyway format in PrimeRail,

1. Create a Milkyway design library with the specified technology file using the `create_mw_lib` command.

For example,

```
pr_shell> create_mw_lib -technology ./libs/test.tf mwlib
```

2. Set up the Milkyway reference library for the Milkyway design library created at Step 1 using the `set_mw_lib_reference` command. You must close the Milkyway library before running the `set_mw_lib_reference` command.

To check if the reference library is properly defined for the design library, run the `report_mw_lib -mw_reference_library` command.

For example,

```
pr_shell> close_mw_lib mwlib
pr_shell> set_mw_lib_reference -mw_reference_library {./reflib} mwlib
pr_shell> report_mw_lib -mw_reference_library mwlib
```

3. Run the `open_mw_lib` command to open the Milkyway design library created at Step 1.

For example,

```
pr_shell> open_mw_lib mwlib
```

4. Read in the netlist information from a Verilog netlist that contains a top module and instances of modules with the `read_verilog` command.

Use the `keep_module` option to specify which subdesigns are to be preserved in an order from the lowest level or the highest level.

For example,

```
pr_shell> read_verilog {TOP_Final_out.v IP_Final_out.v} \
               -keep_module {IP} -top IPTOP
```

5. Annotate the block CEL view with the physical design data taken from the a block DEF file with the `read_def` command. Specify the `-lef` option to import the rotated vias and the LEF file into the design.

For example,

```
pr_shell> current_mw_cel IP
pr_shell> read_def -lef MW.lef.inc IP_Final_out.def
```

The tool reads the DEF file, converts the data into a CEL view and stores the data into the specified Milkyway library.

6. Save the Milkyway block cell with the `save_mw_cel` command.

7. Annotate the top CEL view with the physical design data taken from the a top DEF file using the `read_def` command. Specify the `-lef` option to import the rotated vias and the LEF file into the design.

For example,

```
pr_shell> current_mw_cel IPTOP
pr_shell> read_def IPTOP_Final_out.def -lef MW.lef.save_mw_cel
```

8. Save the Milkyway top cell with the `save_mw_cel` command.
9. Close the cell and the library with the `close_mw_cel` and `close_mw_lib` commands. You should always close the design when data preparation is finished and reopen the design for the subsequent analysis step. This is to ensure proper setup and libraries are loaded for the design.

See Also

- [Converting LEF/DEF Cell Data](#)

Setting Up the Design

PrimeRail requires both [Logical Libraries](#) and [Physical Libraries](#). PrimeRail reads the timing and functionality information from the logical libraries for all standard cells, and the physical library information from the reference libraries and technology files. The reference libraries contain the physical information about the standard cells and macro cells in the logical library. The technology file provides technology-specific information, such as the name and physical characteristics of each metal layer.

For more information about libraries, see the *Library Data Preparation for IC Compiler User Guide*, *IC Compiler II Library Preparation User Guide*, and the *Milkyway Database Application Note*.

Logical Libraries

PrimeRail reads the timing and functionality information for all standard cells and timing information for hard macros from the logical libraries. The tool supports logical libraries that use Composite Current Source (CCS) models, nonlinear delay models (NLDM) or nonlinear power models (NLPM), and automatically selects the timing and power models for power and rail analysis, based on the content of the logical libraries.

In each session, you must set up the logical libraries by defining the search path, link libraries, and target libraries. For more information, see [Before Opening a Design](#).

Note:

PrimeRail supports NLDM and NLPM models for static analysis only. For dynamic analysis, only CCS power libraries are required.

For more information about CCS power libraries, see [Checking CCS Power Libraries](#). For a detailed description about logical libraries, see the Library Compiler documentation.

Physical Libraries

PrimeRail uses reference libraries and technology files to obtain physical library information for the standard cells and macro cells in your logical library. The technology file provides technology-specific information, such as the name and characteristics of each metal layer.

The physical library information is stored in the design library. For each cell, the design library contains several views of the cell, which are used for different physical design tasks. The views that are used by PrimeRail are

- The layout (CEL) view
- The place and route (FRAM) view
- The power and ground connectivity (CONN) view

PrimeRail uses the units defined in the library as the default units for the design, including the unit for time, voltage, current, resistance, capacitance, and power. If your design or SDC file does not have unit settings, the tool uses these default settings.

See Also

- [Preparing and Checking Design Data](#)

Before Opening a Design

Before opening a design, you need to

- Define the search path by setting the `search_path` variable (or choose File > Path Setup in the GUI).

The search path lists, in order, the directories where the tool looks for logical library files that are specified without directory names.

When the tool looks for a logical library file, it starts searching in the first directory specified in the `search_path` variable and uses the first matching library file it finds. For example,

```
pr_shell> set_app_var search_path ". /test/lib"
```

To view the `search_path` settings, run the following command:

```
pr_shell> echo $search_path
```

- Define the `link_path` variable to specify where PrimeRail looks for design files and library files for linking the design.

For example,

```
pr_shell> set_app_var link_path "** test.db"
```

In the previous example, the `test.db` library is set as the default library where PrimeRail reads the necessary design data during an analysis run.

When scaling occurs, the remaining libraries are read automatically after the design is linked.

PrimeRail uses a nonlinear interpolation method for voltage scaling of leakage power and internal power and for temperature scaling of leakage power. For temperature scaling of internal power, a linear interpolation method is used.

Multivoltage Analysis

Use the `link_path_per_instance` variable to provide multivoltage library data for PrimeRail to use different libraries for different cell instances. Before the linking process, if you set the `link_path_per_instance` variable, the tool overrides the default `link_path` variable setting for the specified leaf cell or hierarchical cell instances.

- Define the mapping between the power and ground pins, if the voltage value of a ground pin is not defined in the `.db` file.

The power and ground pin information is used by the `set_rail_power` command to control how the current is distributed from the power pins to the ground pin. By default, PrimeRail reads the mapping data from the `.db` file. If the mapping data is not available, PrimeRail derives power and ground pin data from the FRAM view and maps with the pin data in the `.db` file. However, the derived data might be incorrect, which might cause inaccurate current calculation results.

Manual PG Pin Mapping

Use the `set_pg_pin_mapping` command to manually define how a ground pin is mapped to a list of power pins. To check the PG pin mapping data that is set by the `set_pg_pin_mapping` command, run the `report_pg_pin_mapping` command. You need to invoke these commands before the `open_mw_cel` command.

Multivoltage Analysis

When the mapping is set, PrimeRail first calculates the current value for the power pin that is mapped to this ground pin, and then applies the calculated current to the ground pin. When multiple power nets are mapped to the same ground pin, PrimeRail first calculates the current value for each power pin using the corresponding voltage, and then applies the sum of the current values to the ground pin.

- Use the `create_pg_pin_only_lib_cell` command to create an in-memory library cell with the power and ground pins that are not defined in the `.db` file.

In some cases, the power and ground pin only cells, such as decoupling capacitance cells, are added to the physical design during the implementation stage, but are not defined in the .db file. In this case, you cannot assign capacitance or leakage values to these cells through their power and ground pins.

To enable later capacitance or leakage current assignment to the power and ground pins of these cell instances, you must create a session-based in-memory library reference (which is similar to a .db model) for these cells and their power and ground pins. This allows for a rail analysis run that accepts and incorporates assignment information for instances referring to these in-memory library references.

To report the content of the created library cell, run the `report_pg_pin_only_lib_cell` command.

See Also

- [Preparing and Checking Design Data](#)
- [Before Opening a Design](#)

Opening Milkyway Designs

Open a Milkyway design using the `open_mw_cel` command (or choosing File > Open Design in the GUI). You can open only one Milkyway design at a time.

When you open a Milkyway design, the tool

- Checks the database model (Schema) version and automatically updates the database, if necessary

For more information about converting the database model, see the *Library Data Preparation for IC Compiler User Guide*.

- Checks the correctness and consistency of the netlist-related objects in the Milkyway design

To reduce runtime, the tool determines whether the design previously passed these checks; if it passed, the checks are skipped.

- Reads electromigration rules from the Milkyway design library or technology file
- Links the design
- Performs rail checks and writes potential issues to the ASCII files that are saved in the `PR_CHECKING_DIR` directory
- Flattens the PG netlist for the design. While flattening the design, the tool removes the information for the hierarchical ports and supply nets inside the physical hierarchy cells, including their connection to the upper-level supply nets.

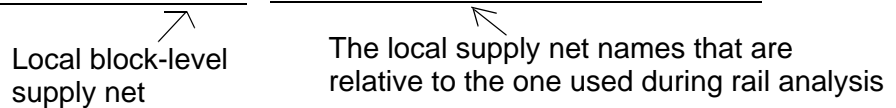
To report the overall physical hierarchy for a top-level supply net, run the `report_supply_net_hierarchy` command.

For example,

```
pr_shell> report_supply_net_hierarchy -hierarchy_level 2 \
          [get_supply_nets VDDTOP]
```

The following is an example of the report.

```
Report : report_supply_net_hierarchy
Design : <Design>
Version: <Version>
Date : <Date>
*****
0 - (Master design_top_cel ) (2): Supply ports:
    Supply Nets: Rail visible supply net: VDDTOP
1 - core/block (Master block_cel) (0): Supply ports:
    VDD Supply Nets: VDDBLOCK Rail visible supply net: VDDTOP
```



Local block-level
supply net

The local supply net names that are
relative to the one used during rail analysis

For more information about the command options, see the man pages.

Note:

If unit inconsistency errors are issued when opening a Milkyway design, check if the main library is correctly defined with the `search_path` variable.

If the Milkyway design library uses an older Milkyway database version (Schema) and a database update is needed, the `open_mw_cel` command automatically updates the library data and issues a message about the conversion. For more information about Milkyway database versions and database updates, see the *Library Data Preparation for IC Compiler User Guide*.

See Also

- [Preparing and Checking Design Data](#)
- [Before Opening a Design](#)
- [Checking Design Readiness](#)

Opening IC Compiler II Designs

Open an IC Compiler II design library or a block using the `open_lib` or `open_block` command.

When you open an IC Compiler II design, the tool

- Checks the correctness and consistency of the netlist-related objects in the IC Compiler II design

To reduce runtime, the tool determines whether the design previously passed these checks; if it passed, the checks are skipped.

- Reads electromigration rules from the design library or technology file
- Links the design
- Performs rail checks and writes potential issues to ASCII files that are saved in the `PR_CHECKING_DIR` directory

For more information about IC Compiler II design libraries, see *IC Compiler II Data Model User Guide*.

Checking Design Readiness

By default, PrimeRail performs certain design readiness checks whenever the data is needed or when setting up the design or preparing data for analysis. For example, PrimeRail performs readiness checks when executing the `open_mw_cel`, `read_supply_parasitics`, `update_currents`, and `calculate_rail_voltage` commands. A summary message is issued whenever a check is performed or a problem is found. PrimeRail does not issue a message when no problem is found. The notification message includes a brief description of the problem, the number of times that the problem is found, and a pointer to the file that contains details about each specific problem.

PrimeRail writes all issues identified by the rail readiness checks to log files which are saved in the `PR_CHECKING_DIR` directory. The checking directory remains after the session is terminated, allowing you to review the checking files after your run completes. If you start a new session that creates a session working directory by reusing the same suffix as an existing checking directory, the existing checking directory is deleted so that you do not have log files from multiple sessions in the same checking directory. If you want to ensure that the checking logs are not deleted by a later session, copy the directory to another location before the session is terminated.

The following example saves a copy of the session checking directory with a new name that uses a date/time stamp as an additional suffix:

```
pr_shell> ls -d ${pr_checking_dir}* PR_CHECKING_DIR_2
pr_shell> exec cp -r $pr_checking_dir \
    ./${pr_checking_dir}.[exec date +%G%m%d:%I.%m]
pr_shell> ls -d ${pr_checking_dir}*PR_CHECKING_DIR_2 \
    PR_CHECKING_DIR_2.20101208:09.12
```

See Also

- [Preparing and Checking Design Data](#)

Reporting Design Checking Results

The `report_rail_checks` command provides a summary report of all of the rail analysis readiness checks performed on the design and related data during the analysis session. PrimeRail performs these readiness checks to determine whether there are problems with the design or library setup that could potentially invalidate the results of the analyses you perform.

The readiness report documents many issues, including the following:

- Missing power or ground pins in the .db file
- Inconsistent power or ground pins between the .db file and the FRAM view
- Unrealistically large intrinsic resistance and capacitance values on a power or ground pin
- Unrealistically large power values derived for a cell
- Supply net shapes that are not connected to any ideal voltage sources (that is, tap points)

Example

```
*****
Report : report rail checks
Design :
Version: <Version>
Date   : <Date>
*****
```

Rail Readiness Check	Man Page ID	Checked	Issues	File
####Library Checking####				
Missing library PG pins	RAIL-001	Yes	0	-
Missing Liberty library CCSP waveform	RAIL-002	Yes	386	
	/PR_CHECKING_DIR/current_waveform.2			
Physically unconnected PG pins	RAIL-010	No	-	-
Unusual switch cell linear resistance	RAIL-018	Yes	0	-
####Power Calculation Checking####				

Unusually large signal pin cap/slew	RAIL-108	No	-	-
Exceeded peak current threshold	RAIL-109	No	-	-
Missing power-arc CCSP waveform	RAIL-110	No	-	-
Over-extrapolation for CCSP cap/slew	RAIL-111	No	-	-
Over-interpolation for CCSP cap/slew	RAIL-112	No	-	-
Large cell instance PG pin power	RAIL-113	No	-	-
####Design Checking####				
Unusually large PG pin intrinsic RC	RAIL-114	No	-	-
Cells not connected to supply nets	RAIL-115	Yes	0	-
Supply nets without cell connections	RAIL-116	Yes	0	-
Inconsistent soft macro CEL/FRAM view	RAIL-117	Yes	0	-
####PrimeTime Report Checking####				
Unrecognized PTPX report cells	RAIL-139	No	-	-
Unknown PTPX report cell PG pins	RAIL-140	No	-	-
Unknown PTPX report PG pin voltages	RAIL-141	No	-	-
####Constraints Checking####				
Different supply net voltages	RAIL-143	No	-	-
Mismatched supply net voltages	RAIL-144	No	-	-
Physically unconnected PG pins	RAIL-301	No	-	-
Missing PG pins for hard macro cells	RAIL-325	Yes	0	-
####Tap Checking####				
Invalid taps	RAIL-333	No	-	-
####Electromigration Rule Checking####				
Missing EM rules	RAIL-567	Yes	33	
	/PR_CHECKING_DIR/em.missing_rules.2			
#### Checking####				
Extracted terminals unmatched to PGs	PGEX-002	No	-	-
Mismatched supply PG pins	PGEX-003	No	-	-
Missing library macro PG pins	PGEX-004	No	-	-
Extracted PG terminals unconnected	PGEX-005	No	-	-
####Milkyway Versus Liberty File Checking####				
PG pins inferred from Milkyway	MKW-016	Yes	0	-
Inconsistent Milkyway library PG pins	MKW-017	Yes	0	-
Unmatched Liberty PG pins in Milkyway	MKW-018	Yes	0	-
More Milkyway PG pins than library	MKW-019	Yes	0	-
Boundary ports for bus naming style	MKW-026	Yes	0	-
Reference library changes	MKW-027	Yes	0	-

See Also

- [Preparing and Checking Design Data](#)

5

Analyzing IC Compiler II Designs

PrimeRail supports IC Compiler II design libraries for both static and dynamic rail analyses. PrimeRail provides the same set of analysis capabilities, such as integrity checking, inrush analysis or minimum path resistance analysis, as in the Milkyway flow.

This chapter contains the following topics:

- [Rail Analysis Flow With IC Compiler II Designs](#)
- [Input Data for Analyzing IC Compiler II Designs](#)
- [Displaying Maps in the IC Compiler II GUI](#)
- [Sample Scripts for Analyzing IC Compiler II Design Libraries](#)

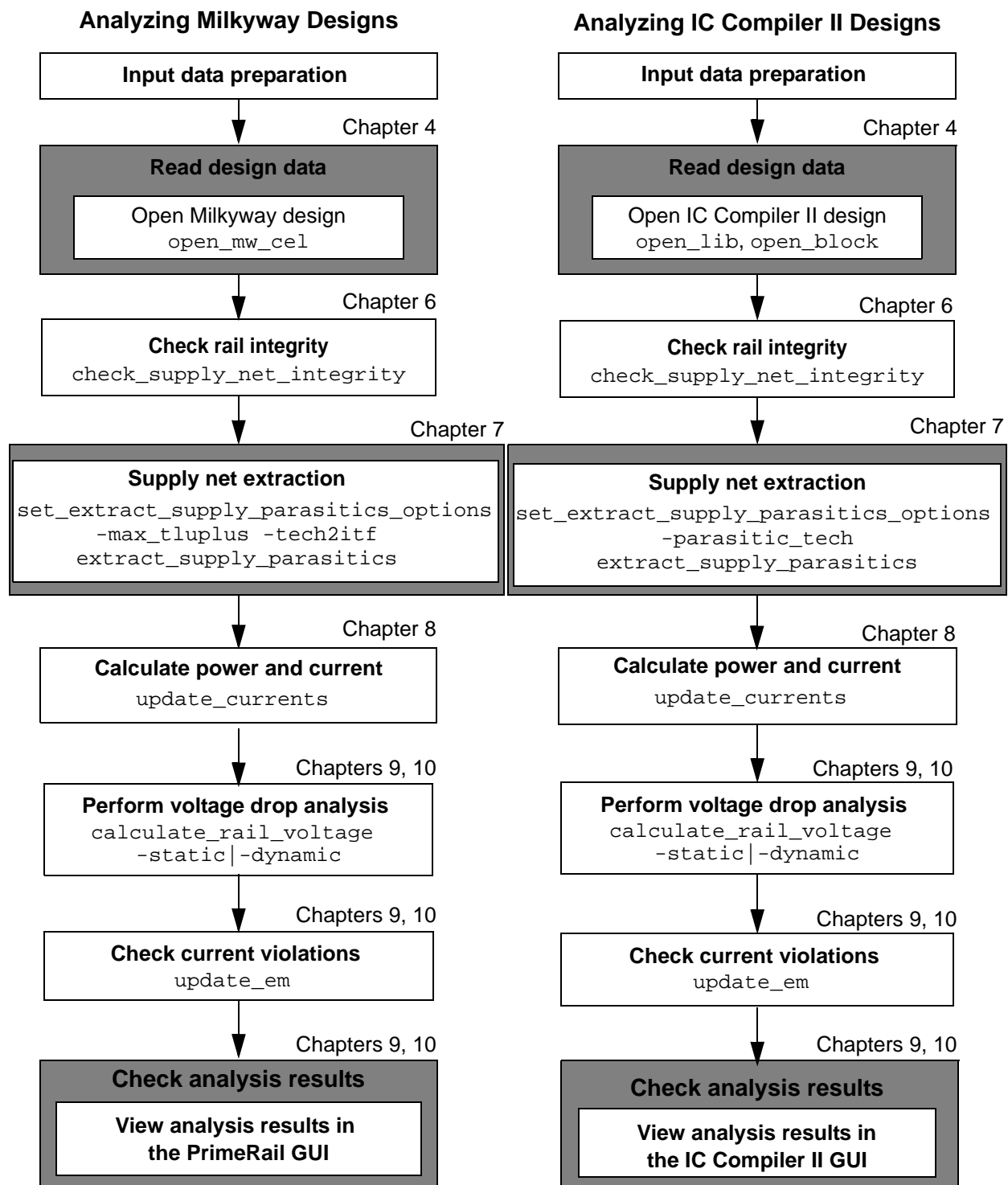
Rail Analysis Flow With IC Compiler II Designs

You first check for physical connectivity, such as missing vias or floating pins, when analyzing IC Compiler II designs in PrimeRail. Then you extract the parasitic information for the power and ground network using the built-in power and ground net extraction engine, and run timing and power analysis to generate power and current values. With the extracted parasitic data and current values, you can analyze the voltage drop and electromigration violations of the design in the static and dynamic rail analysis flows.

[Figure 5-1](#) compares the steps for analyzing IC Compiler II and Milkyway design libraries in PrimeRail. The steps are similar, except that in the IC Compiler II flow you open a design with the `open_lib` and `open_block` commands and display maps in the IC Compiler II GUI. The rail macro model creation flow requires additional steps, too.

See [Displaying Maps in the IC Compiler II GUI](#) for more information about displaying maps for NDM designs in the IC Compiler II GUI.

Figure 5-1 Steps for Analyzing Milkyway Designs and IC Compiler II Designs



To analyze IC Compiler II designs, run the following steps:

1. Prepare the required input data, as described in [Preparing Input Data](#).
2. Define the `search_path` and `link_path` variables to specify where PrimeRail looks for design files and library files for linking the design.

Specify a location and directory name for saving rail analysis results by using the `set_rail_database dir_name` command

See [Before Opening a Design](#) for more information.
3. Open the design using the `open_lib` and `open_block` commands.

See [Opening IC Compiler II Designs](#) for more information.
4. Check for physical connectivity using the `check_supply_net_integrity` command.

When the checking is finished, run the `report_integrity_results` command to report the integrity results within the session.

See [Integrity Checking](#) for more information.
5. Configure settings for supply net parasitic extraction using the `set_extract_supply_parasitics_options` command.

You must use the `-parasitic_tech` option to specify the location of the technology and TLUPlus files that are loaded to design, as well as the IC Compiler II referenced libraries.

Perform supply net parasitic extraction with the `extract_supply_parasitics` command.

See [Setting Extraction Options](#) for more information.
6. Generate power and current values with the `update_currents` command, as explained in [Analyzing Power and Updating the Currents](#).
7. Perform voltage drop analysis and check current violations with the `calculate_rail_voltage` and `update_em` command, as explained in [Static Rail Analysis](#) or [Dynamic Rail Analysis](#).
8. To view the generated maps in the layout, run the `start_icc2_gui` command to invoke the IC Compiler II GUI in which you display maps to check for problematic areas where violations occur. Displaying error views in the error browser is also supported.

See [Displaying Maps in the IC Compiler II GUI](#) for more information.

See Also

- [PrimeRail Analysis Flows](#)

Input Data for Analyzing IC Compiler II Designs

PrimeRail uses a certain set of data to be available in the database, such as ideal voltage drop sources, timing constraints, voltage settings, signal parasitics or switching activity, when analyzing an IC Compiler II design.

You also need to complete parasitic settings in the IC Compiler II environment before running parasitic extraction on IC Compiler II designs. Use the `read_parasitic_tech` command to specify the locations of the TLUPlus models and the layer mapping files, and the `set_parasitic_parameters` command to save the late parasitic tech specification files in the IC Compiler II design library.

Here is an example:

```
icc2_shell> read_parasitic_tech -tlup test.tluplus \
               -layermap layermap.txt -name test1
icc2_shell> set_parasitic_parameters -late_spec test1
icc2_shell> save_block
icc2_shell> save_lib
```

For more information about preparing input data for running rail analysis in PrimeRail, see [Preparing Input Data](#).

See Also

- [PrimeRail Analysis Flows](#)

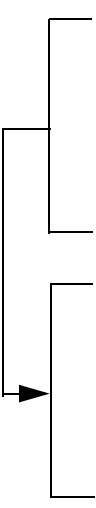
Displaying Maps in the IC Compiler II GUI

When running rail analysis on an IC Compiler II design is complete, run the `start_icc2_gui` command to display the generated maps or error views in the IC Compiler II GUI. This is because GUI-based analysis is available only in `icc2_shell`, not `pr_shell`. [Figure 5-2](#) explains the data flow for displaying maps in the IC Compiler II GUI.

When the `start_icc2_gui` command is executed, PrimeRail first creates a script, called `icc2_open_rail_result.timestamp.tcl`, and then opens the IC Compiler II GUI (see [Figure 5-3](#)). The script contains all the required commands and design settings to invoke the IC Compiler II GUI (without requiring the IC Compiler II license). Design data and analysis results are automatically loaded for investigating the problematic areas in the map. The generated integrity or current violation errors are also available for display in the error browser.

If you rerun rail analysis within the same session in PrimeRail after invoking the IC Compiler II GUI, you need to manually open the latest rail result context from the IC Compiler II GUI by choosing File > Rail Results. A warning is issued if you re-invoke the IC Compiler II GUI without exiting the previous one.

The following is an example of the `icc2_open_rail_result.timestamp.tcl` script generated by the `start_icc2_gui` command.



```
set ::search_path { . /install_lib_path/pr_icc2/reflibs/dummyLib/LM
                  /install_lib_path/pr_icc2/reflibs/test/LM}
open_lib -read /test.nlib
open_block -read test_CASE_2
create_taps -import ./icc2_open_rail_result.taps.01_05_35
set_app_options -name rail.database -value RAIL_DATABASE
open_rail_result RAIL_RESULT_1
gui_start
```

1. Set path to where PrimeRail is installed
2. Open the design being analyzed
3. Create tap information by importing the data from `pr_shell`
4. Create a directory for the IC Compiler II tool to retrieve the analysis results
5. Open the rail context that was saved during the `pr_shell` session
6. Invoke the IC Compiler II GUI

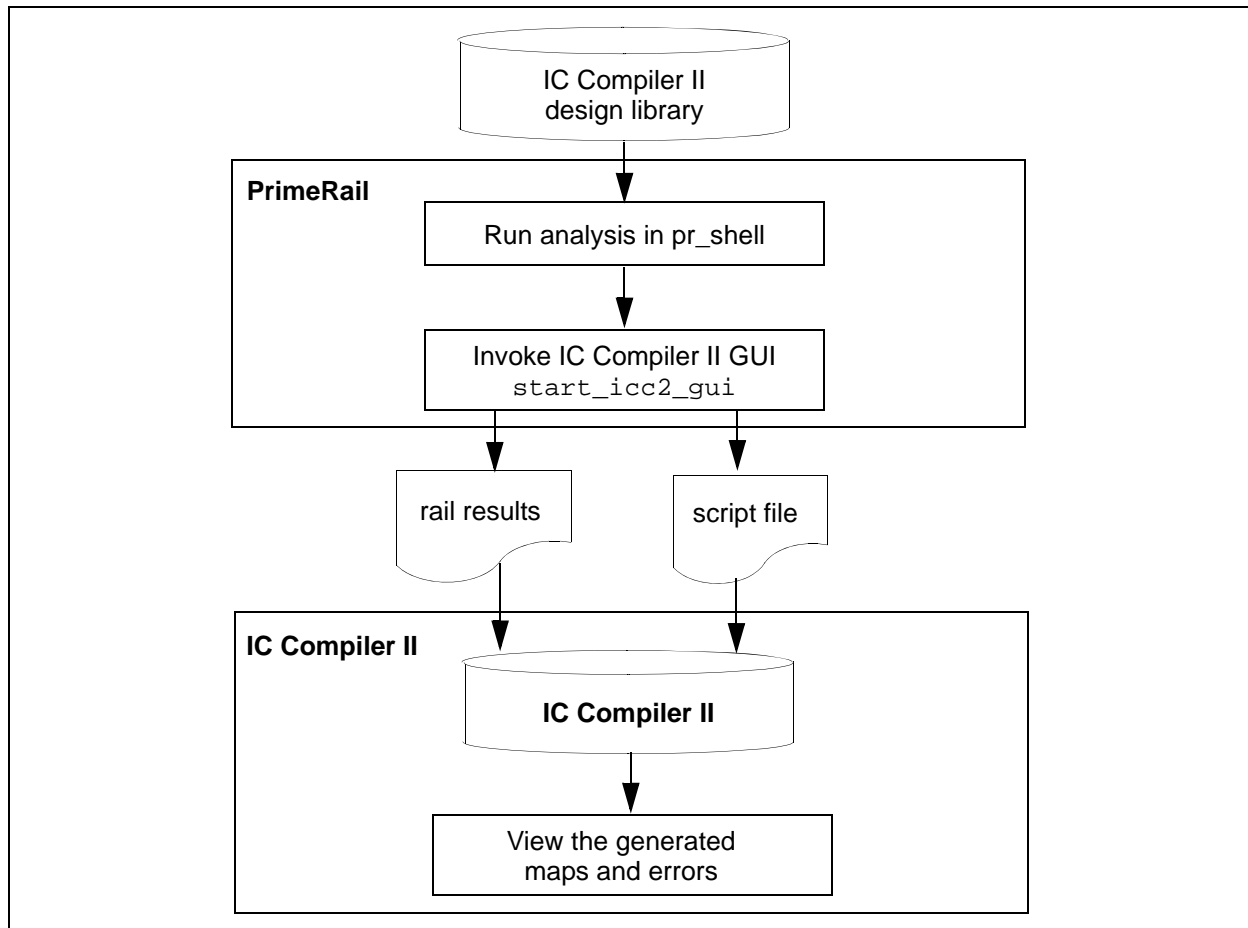
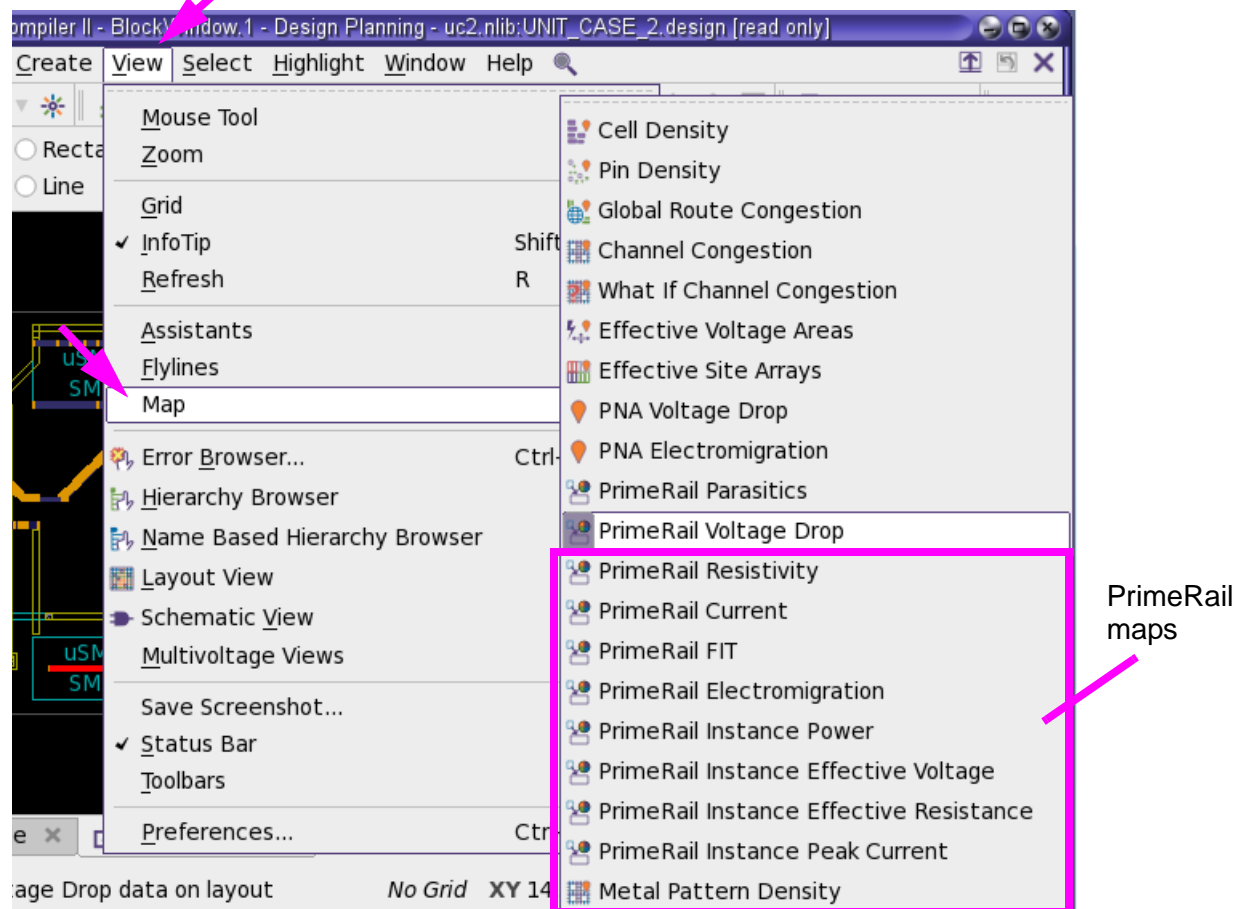
Figure 5-2 Displaying Maps in the IC Compiler II GUI

Figure 5-3 Invoking the IC Compiler II GUI from *pr_shell* for Map Display

Choose View > Map. Choose a map you want to display.



See Also

- [Rail Analysis Flow With IC Compiler II Designs](#)

Sample Scripts for Analyzing IC Compiler II Design Libraries

The following script runs voltage drop analysis on an IC Compiler II design and opens the IC Compiler II GUI for checking the problematic areas.

```
set rail_database ./pr_work/RAIL_DATABASE
set_host_options -max_cores 16

set ::search_path {;}
lappend ::search_path /global/test1
lappend ::search_path /global/test2

set ::link_path {}
lappend ::link_path *
lappend ::link_path lib_1.db
lappend ::link_path lib_2.db

open_lib mylib
open_block myblock

create_taps -import tap.file
set_voltage 1.0 -min 1.0 -object_list VDD
set_voltage 0.0 -min 0.0 -object_list VSS
set_extract_supply_parasitics_options -parasitic_tech test

extract_supply_parasitics {VDD VSS}
read_supply_parasitics {VDD VSS}

read_sdc my.sdc
read_parasitics my.spef
update_currents -static

calculate_rail_voltage {VDD VSS}
# start_icc2_gui to check maps in icc2_shell
start_icc2_gui

# (optional) start_icc2_gui -icc2_shell_path install_dir/bin
```


6

Integrity Checking

The PrimeRail rail integrity checking engine analyzes top designs to report insufficient physical connectivity in the power network of the design.

The PrimeRail integrity checking engine provides the following features:

- Physical connectivity checking, including floating shapes, floating pin shapes, dangling pin access edges, missing vias, maximum distances between vias, discontinuous connections, and dangling vias
- Integrity error collection commands to enable user-driven fixing flows in the In-Design rail analysis flow in the IC Compiler environment

This chapter contains the following sections:

- [Rail Integrity Checking Flow](#)
- [Rail Integrity Checking](#)
- [Integrity Checking Types](#)
- [Toggling Between Integrity Results](#)
- [Reporting Integrity Results](#)
- [Deleting Integrity Results](#)
- [Viewing Integrity Errors in the Error Browser](#)

Rail Integrity Checking Flow

The PrimeRail integrity checking engine uses the physical information from the database. Execute the following commands before you run rail integrity checking on the design:

1. Open the Milkyway design with the `open_mw_cel` command, or the IC Compiler II design with the `open_lib` command, so PrimeRail can access the design.

See [Opening Milkyway Designs](#) and [Opening IC Compiler II Designs](#) for more information.

2. Specify where the ideal voltage sources are with the `create_taps` command.

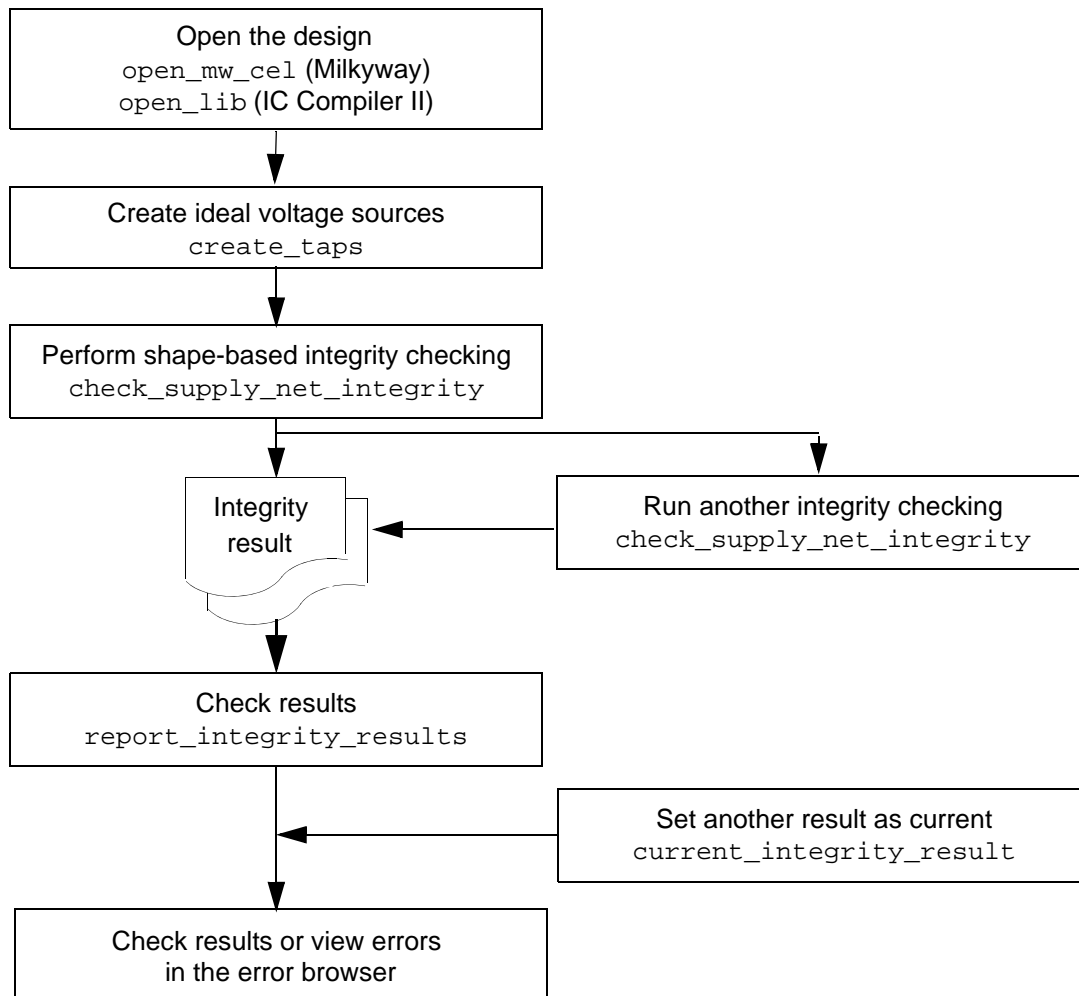
See [Creating Taps for Ideal Voltage Sources](#) for more information.

3. Perform shape-based integrity checking with the `check_supply_net_integrity` command, as explained in the [Rail Integrity Checking](#) section.

4. When the checking process is complete, run the `report_integrity_results` command to report the integrity results within the session.

[Figure 6-1](#) illustrates the integrity checking flow.

Figure 6-1 PrimeRail Integrity Checking Flow

**See Also**

- [Integrity Checking](#)

Rail Integrity Checking

Use the `check_supply_net_integrity` command analyzes the power and ground routing associated with a supply net. It checks for missing via candidates, floating shapes, floating pin shapes, dangling pin access edges, dangling vias, maximum distance between vias, and discontinuous shape connections.

When integrity checking is complete, the tool saves results to a result context specified by the `-result` option, or generates error data specified by the `-error_view` option. You can examine these integrity errors in the error browser.

To load and set a previously generated result as the one to be used in the current session, use the `current_integrity_result` command. For a detailed description about switching between integrity results, see [Toggling Between Integrity Results](#).

For more information about displaying the error views generated during integrity checking, see [Viewing Integrity Errors in the Error Browser](#).

The `current_integrity_result` command checks for the following types of errors:

- Floating shapes, to determine if any power and ground shapes (non-pin shapes) are electrically floating. You can
 - Use the `-non_ideal_sourced_floating_shapes` option to check for floating supply net segment shapes.
 - Use the `-dont_group_contiguous_floating_shapes` option to prevent grouping of contiguous floating shapes into connected groups.

See [Creating Taps for Ideal Voltage Sources](#) and [Floating Shapes](#) for more information.

- Discontinuous connections, to find if any two metal shapes are placed closely enough to possibly be connected with each other. You can
 - Use the `-enable_discontinuous_connection_check` option to enable discontinuous connection checking.
 - Use the `-discontinuous_connection_threshold`
`list_of_metal_layer_gap_distance_pairs` option to specify the search distance. This is useful if you want to define the gap distance per layer.

Use the `list_of_metal_layer_gap_distance_pairs` argument to specify a list of `pair_list` elements. Each `pair_list` element consists of two elements; the first element is a metal routing layer and the second element is a distance value in library units.

By default, the distance value per layer is the minimum spacing on that layer as specified in the technology file.

- Use the `-ignore_discontinuous_connection_routing_layers`
`list_of_routing_layers` option to specify which layers are to be skipped during checking.

The `list_of_routing_layers` argument specifies a list of routing layers, not mask layers.

- Use the `-discontinuous_connection_directionality` `list_of_metal_layer_directionality_pairs` option to indicate the search direction for the discontinuous connection checking.

The `list_of_metal_layer_directionality_pairs` argument specifies a list of `pair_list` elements. Each `pair_list` element consists of two elements; the first is a routing metal layer and the second is a directional setting of either H, V, or HV. The default directionality per layer is HV.

Note that the engine ignores the 45-degree paths.

- Use the `-ignore_discontinuous_connection_shape_types` `list_of_shape_types` option to indicate which shape type to ignore during discontinuous connection checking.

By default, the engine assumes no shape types are ignored.

The valid shape types are `pin_shape`, `net_shape`, `via_enclosure`, and `terminal`.

See [Discontinuous Connection Checking](#) for more information.

- Floating pin shapes, to find electrically floating pin shapes of the supply nets, that is, pin shapes that are not connected to any ideal voltage source that is specified with the `create_taps` command. You can
 - Use the `-pg_pin_floating_connection_check` `minimal` option to find the pin that is neither directly nor indirectly connected to the tap point or to a supply net with at least one pin shape of each electrically equivalent setting.
 - Use the `-pg_pin_floating_connection_check` `maximal` option to find all pin shapes of a given connected pin to a supply net such that the physically connected supply shapes are not directly or indirectly connected to tap points.
 - Use the `-ignore_touching_floating_pin_shape_errors` option to allow floating supply pin shapes that are either touched or connected to the same supply net. This is useful when the electrically equivalent (or CONN view electrically equivalent) settings are ignored and the cells with supply pins are abutted in a daisy chain fashion.

See [Floating Pin Shapes](#) for more information.

- Dangling pin shapes, to find pin shapes that have pin access edge markings but no physical connections along the edge to a netlist-connected supply net shape. Use this feature when a long pin is physically connected with one edge only. You can
 - Use the `-dangling_pin_shape_edge_check` option to find non-floating pin shapes if one of the edges has a pin access edge marking but no physical connection to a supply net.
 - Use the `-ignore_pins_by_cell` `cel_name` option to specify the cells to be excluded from dangling pin shape edge checking.

- Use the `-ignore_pins_by_cell_type` option to specify the cell types to be excluded from dangling pin shape edge checking.
- Use the `-ignore_pin_shape_layers list_of_routing_layers` option to ignore dangling pin shape edges by layer, where *list_of_routing_layers* is a list of routing layers, not mask layers.
- Dangling vias, to find dangling vias that have pin edge access markings but no physical connections along the edge to a netlist-connected supply net shape.
 - Use the `-ideal_sourced_dangling_vias` option to find dangling vias that are electrically connected (non-floating) but with one open-ended enclosure.

See [Dangling Vias](#) for more information.

- Missing vias, to find overlaps of supply net routing on different layers that do not have vias connected. If the overlap is large enough to accommodate a minimum-sized via and no via is found between the overlapping shapes in the layer at this location, PrimeRail flags this as a missing via.

You can

- Use the `-missing_via_rule {layer_name1 object_type_spec1 layer_name2 object_type_spec2}` option to find possible missing vias with specified objects.
- Use the `-via_existence_check_partially_enclosed` option to allow existing via cuts being partially enclosed within the overlap area between shapes.
- Use the `-existing_stacked_via_check_mode via_enclosure` option to allow staircase stacked vias.

To check for vertically aligned stacked vias, use the `via_cut` argument for the `-existing_stacked_via_check_mode` option.

To disable stacked via checking and check for missing vias only in adjacent metal routing layers, use the `-no_stack` option.

- Use the `-missing_via_net_shape_direction` option to filter out the pair of net shape objects in the missing via rule such that they are only checked if they obey this directive. The directive states that the pair of endpoints are to be checked only if they are either directionally perpendicular or parallel or both.
- Use the `-merge_same_layer_shapes shape_type_spec_list` option to merge all shapes on the same layer associated with the shape types defined in the `shape_type_spec_list` argument. The command reports the bounding box of the merged shape. The `shape_type_spec_list` is a list of shape types, including `pin_shape`, `net_shape`, `terminal` and `via_metal_enclosure`.
- Use the `-cut_merged_shape` option to cut a merged shape into several rectangles when a polygonal error is returned. This helps ensure the errors returned do not overlap with each other.

- Use the `-via_existence_check_merge_via_cut` option to merge all via cuts into one virtual via cut.
- Use the `-ignore_via_enclosure_shape` option to remove via enclosure metals from consideration as overlapping endpoints.
- Use the `-ignore_small_overlapping_area` option to allow overlaps that are smaller than the user-defined area.
- Use the `-report_specific_missing_via_in_stacked_via` option to write the layer information for a missing stacked via in the error report.

See [Missing Vias](#) for more information.

Use the options listed in [Table 6-1](#) to report checking errors only on a specified area. If you use both the `-included_area_polygon_list` and `-excluded_area_polygon_list` options to create a rectilinear area, the command reports errors that are only inside the calculated final combined area. See [Area-Based Filtering](#) for more information.

Table 6-1 Options for Reporting Checking Errors

Option	Description
<code>-area_edge_condition</code>	Reports all errors, whose associated area is either completely inside the area with line touching or without line touching or partially within the area but intersecting.
<code>-included_area_polygon_list</code>	Reports all errors, whose associated area is inside the specified area.
<code>-excluded_area_polygon_list</code>	Reports all errors, whose associated area is outside the specified area.

Integrity Checking Types

This section describes the checking types that PrimeRail integrity checking engine provides. The available checking types are

- [Floating Shapes](#)
- [Floating Pin Shapes](#)
- [Dangling Pin Shapes](#)
- [Discontinuous Connection Checking](#)
- [Dangling Vias](#)
- [Missing Vias](#)

The first two types are commonly used in DRC and LVS checking tools, such as IC Validator, and they are checked relative to terminals of a physical block. In contrast, in PrimeRail, these checks are performed with taps or voltage source rail constraints. If there is no path from a shape to a tap, PrimeRail flags it as an error.

You can specify different checking criteria to find problems based on your intent. For example, you can use [Area-Based Filtering](#) by creating rectilinear areas. In this case, the integrity checking engine reports errors that are only inside the specified area.

See Also

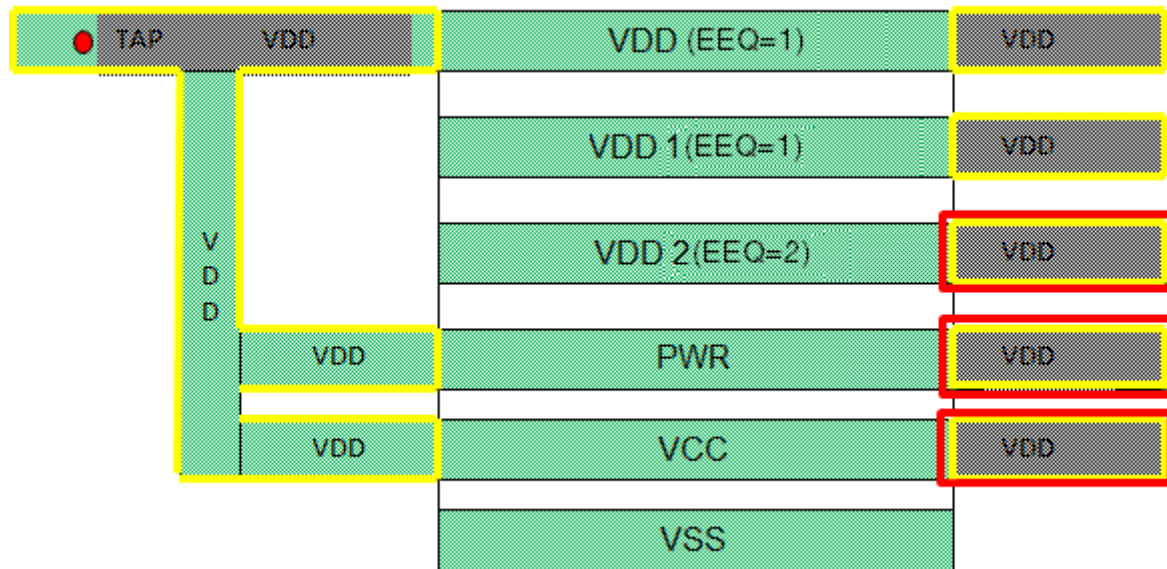
- [Integrity Checking](#)
- [Rail Integrity Checking Flow](#)

Floating Shapes

Shapes that are not contiguously and physically connected to an ideal voltage source are identified as floating shapes. If two shapes are touching, intersecting, or overlapping each other, PrimeRail considers them as connected and marks them with the same group ID. You can use the group ID on a set of shapes to color the grouped shapes in the layout using the error browser.

[Figure 6-2](#) shows a floating shape example. The VDD net is connected in the netlist only to the VDD pin on the cell instance in the middle. PrimeRail reports the last three shapes (framed in color red) on the right hand side as floating errors. By default, PrimeRail reports contiguous floating shapes by group. To disable it, specify the `-dont_group_contiguous_floating_shapes` option with the `check_supply_net_integrity` command.

Figure 6-2 An Example of Floating Geometry Checking



Floating Pin Shapes

To make sure all cell instances have sufficient power, use the `-pg_pin_floating_connection_check disable | minimal | maximal` option to check if any pin shapes are connected physically to an ideal voltage source. When you set the checking criterion as `minimal`, the integrity engine identifies a pin as nonfloating when at least one pin shape of the pin that has the same `eeq_class` or `conn_view_eeq_class` value is not floating. When you set the criterion to `maximal`, the engine identifies a pin as non-floating when all of its pin shapes that belong to the same `eeq_class` or `conn_view_eeq_class` value are not floating.

To satisfy the minimal criterion, as there might be N different `eeq_class` values on a pin with M pin shapes, there must be at least N non-floating connections in the pin shapes for the given pin.

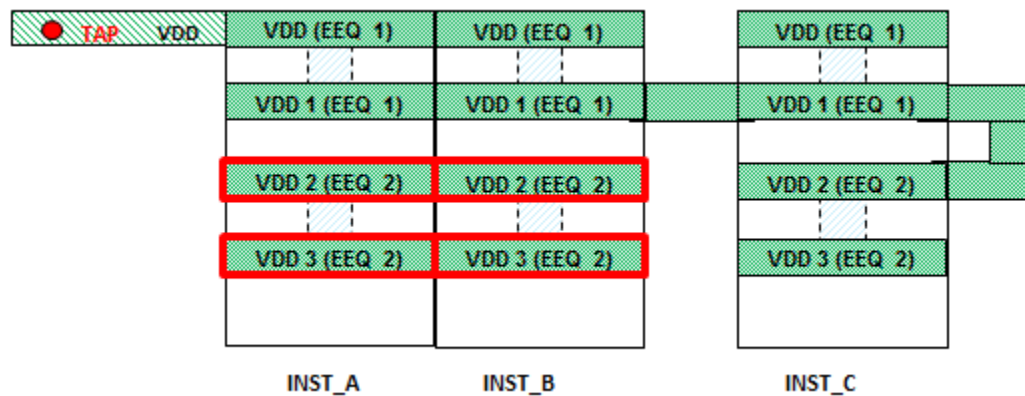
To satisfy the maximal criterion, all M pin shapes of a pin must have non-floating connections. However, since the integrity checking engine traces floating physical connectivity between pin shapes of the same `eeq_class` or `conn_view_eeq_class` value, all of the other pin shapes with the same value are considered not floating if one pin shape of the given pin is not floating.

To avoid such a problem, use the `-ignore_eeq_for_physical_connectivity_tracing` option (or the `-ignore_conn_view_eeq_for_physical_connectivity_tracing` option if the `-use_conn_view` setting is enabled). This disables tracing within the model and to ensure only pin shapes are connected from the containing physical level of hierarchy.

To filter out the results based on the `mask_layout_type` setting of the cell that has the pin shapes, use the `-ignore_pins_by_cell_type` option. To filter pin shapes by layer, use the `-ignore_pin_shape_layers list_of_routing_layers` option, where `list_of_routing_layers` is a list of routing layers, not mask layers.

For example, the cells shown in [Figure 6-3](#) have two electrically equivalent pin shape groups (that is, EEQ class) and these pin shape groups internally have physical connectivity. PrimeRail considers these cells well powered only if all of these electrically equivalent pin shape groups are connected physically to an ideal voltage source. In some cases, such as hard macros, you might need to make sure all pin shapes are connected to an ideal voltage source to avoid potentially large voltage drops. To avoid such problems, specify the `-pg_pin_floating_connection_check` option with the `check_supply_net_integrity` command.

Figure 6-3 An Example of Floating Pin Shapes With Minimal Checking



Physical Model Settings

When running floating checks on the design, use the options to specify the physical model settings which alter the behavior of the physical shape tracing through the pin shapes within the model. These options are

- `-ignore_eeq_for_physical_connectivity_tracing`
- `-override_eeq_class_to_interpret_as_same_class`
- `-use_conn_view`
- `-override_conn_view_eeq_class_to_interpret_as_same_class`
- `-ignore_conn_view_eeq_for_physical_connectivity_tracing`

By default, the checking engine uses electrically equivalent classes to model internal connectivity based on the `eeq_class` attribute on the FRAM view's terminals. If electrically equivalent classes are not set correctly, set the electrically equivalent classes and trace the

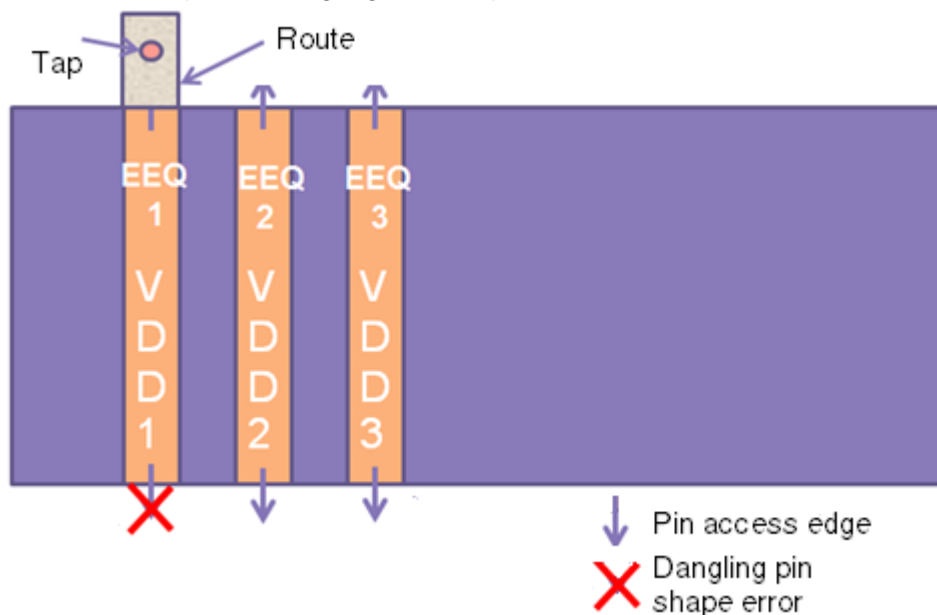
connectivity by using the `check_supply_net_integrity -use_conn_view` command. To disable connectivity tracing by electrically equivalent classes, use the `-ignore_eeq_for_physical_connectivity_tracing` option. To consider all pin shapes connected internally, use the `-override_eeq_class_to_interpret_as_same_class` option.

Dangling Pin Shapes

Use the `-dangling_pin_shape_edge_check` option to find non-floating pin shapes if one of their edges has a pin access edge marking but is not physically connected to a netlist-connected supply net. To ignore dangling pin shape edges by cell or cell type, use the `-ignore_pins_by_cell` or `-ignore_pins_by_cell_type` option accordingly. To ignore dangling pin shape edges by layer, use the `-ignore_pin_shape_layers` `list_of_routing_layers` option, where `list_of_routing_layers` is a list of routing layers, not mask layers.

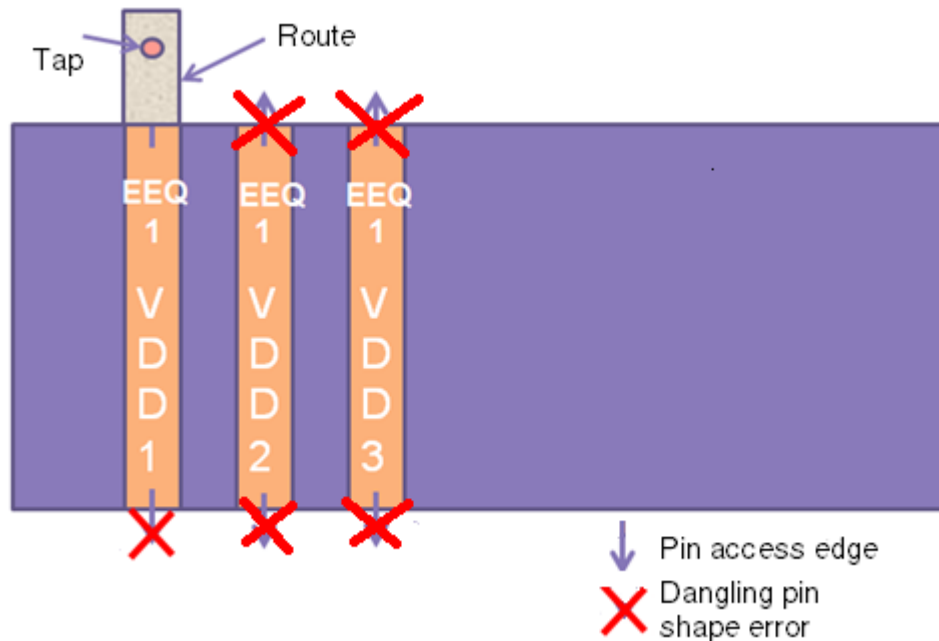
Shown in [Figure 6-4](#), the I/O cell has three pin shapes; each pin shape has a pin access edge defined. The engine identifies a dangling pin shape error on pin shape VDD 1 because its upper edge connects to a supply net but the lower one does not. The VDD 2 and VDD 3 pin shapes are reported as floating because they have different `eeq_class` attribute settings from VDD 1.

Figure 6-4 An Example of Dangling Pin Shapes



However, if the pin shapes VDD 2 and VDD 3 have the same electrically equivalent setting as VDD 1 (as shown in [Figure 6-5](#)), the engine reports both edges of pin shapes VDD 2 and VDD 3 as dangling, along with the lower edge of VDD 1.

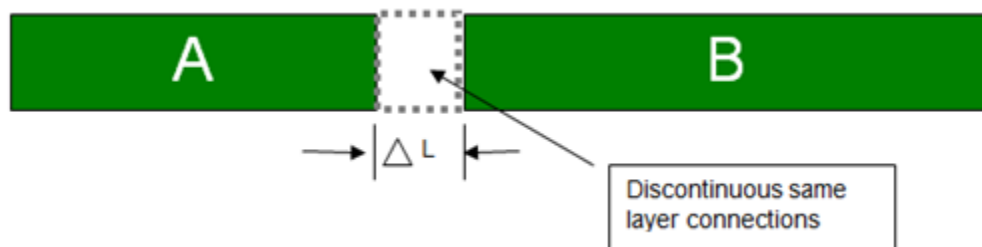
Figure 6-5 Reporting Dangling Pin Shapes for Shapes of the Same EEQ Class



Discontinuous Connection Checking

The integrity checking engine flags a discontinuous connection error when the engine finds two metal shapes that are placed closely enough to be connected with each other. Shown in Figure 6-6, the design has two shapes A and B that are on the same metal layer and the same net. The integrity engine detects a gap (ΔL) between A and B. If the gap (ΔL) is smaller than the user-specified threshold, PrimeRail flags the gap as a discontinuous connection error.

Figure 6-6 Detecting a Discontinuous Gap



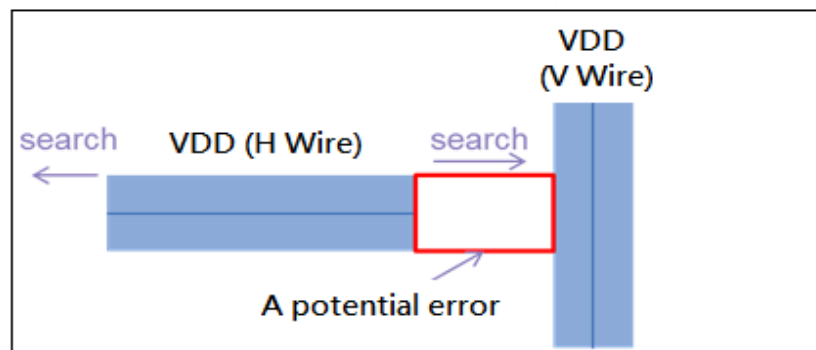
The integrity checking engine is able to find discontinuous errors on the physical net shape objects: wire, path, pin, and rectangle. The following sections describe how the engine performs discontinuous connection checking for these net shape objects:

- [Wires](#)
- [Paths](#)
- [Pin Shapes](#)
- [Rectangles](#)

Wires

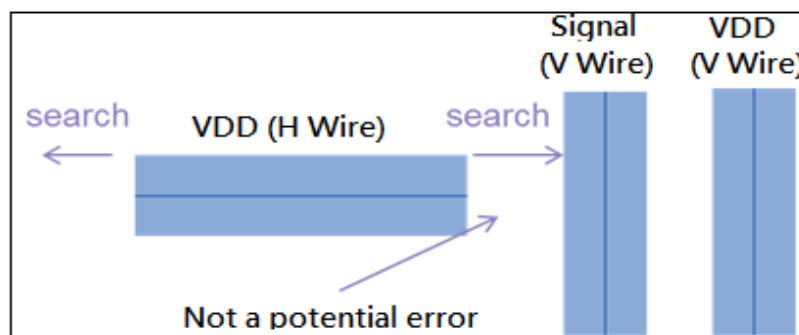
As shown in [Figure 6-7](#), the engine checks a horizontal wire by searching the nearest metal along the x-axis direction. Because the nearest shape is a VDD wire, the engine flags the gap as a discontinuous connection error. You can specify the search distance by using the `-discontinuous_connection_threshold` option with the `check_supply_net_integrity` command.

Figure 6-7 Checking for Discontinuous Errors on a Horizontal Wire



However, if the nearest same layer metal shape is not on the same net but within the minimum spacing of the horizontal wire, PrimeRail does not consider the gap as a discontinuous connection error (see [Figure 6-8](#)).

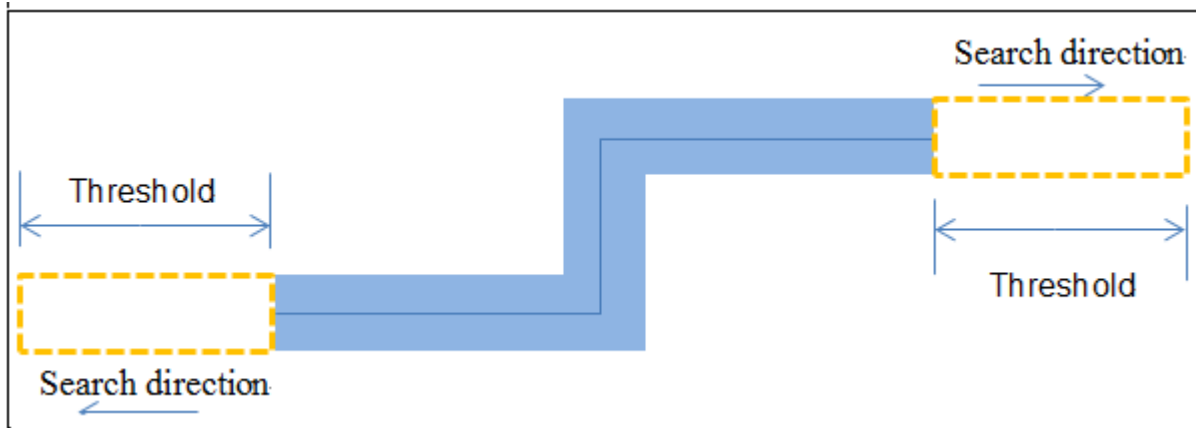
Figure 6-8 Not a Discontinuous Connection When the Wires Are Not on the Same Net



Paths

The integrity engine checks for discontinuous connections from the ends of a path, as shown in [Figure 6-9](#). Use the option to ignore other wires between two end shapes of the same supply net which would then enable these to be flagged as errors.

Figure 6-9 Checking for Discontinuous Connections From the End of a Path



Pin Shapes

The integrity checking engine checks for discontinuous connection errors on pins starting with the access edge of the pin. The engine searches for errors in the same direction as that of the access edge. You can examine the access edge and the direction in the GUI (see [Figure 6-10](#)). As shown in [Figure 6-11](#), the VDD pin has one access edge. PrimeRail finds a discontinuous connection error at the gap with wire A.

Figure 6-10 Examining the Access Edge and Direction in the GUI

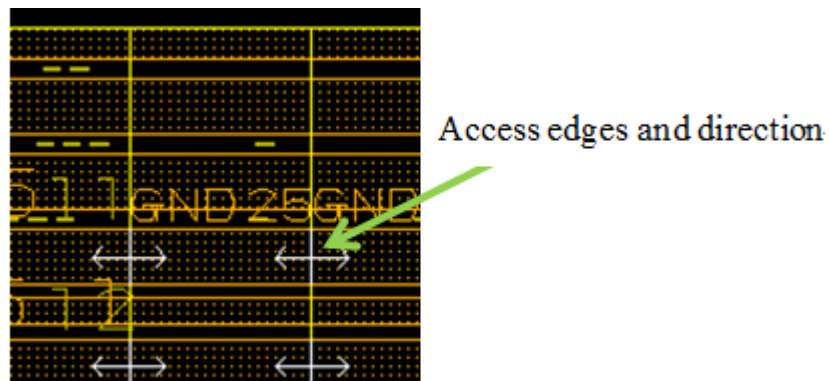
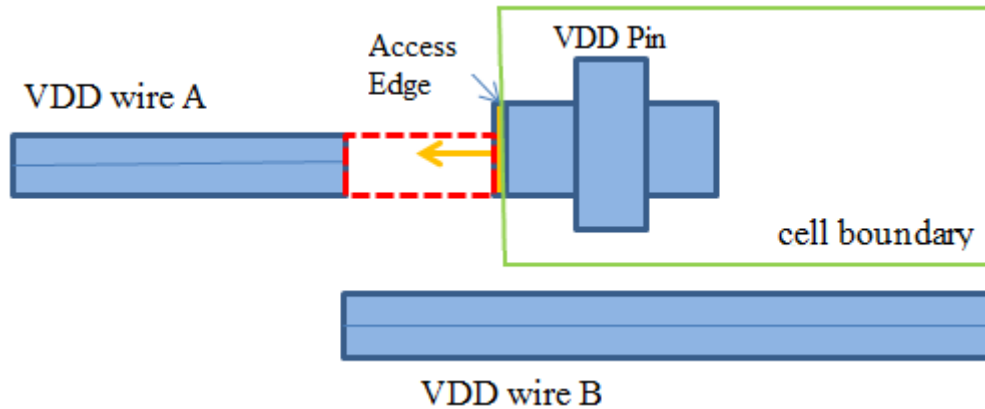


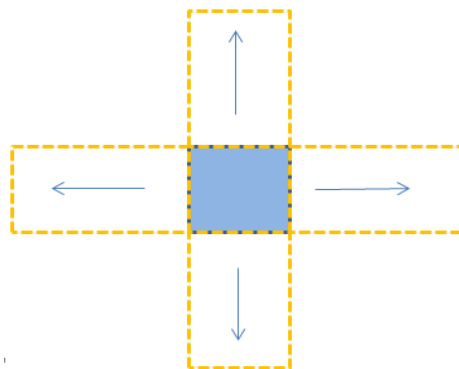
Figure 6-11 Checking for Discontinuous Connection Errors by the Access Edge



Rectangles

The engine checks for all four directions (left, bottom, right, and top) of a rectangle shape as shown in [Figure 6-12](#).

Figure 6-12 Checking for Discontinuous Connection Errors on a Rectangle Shape

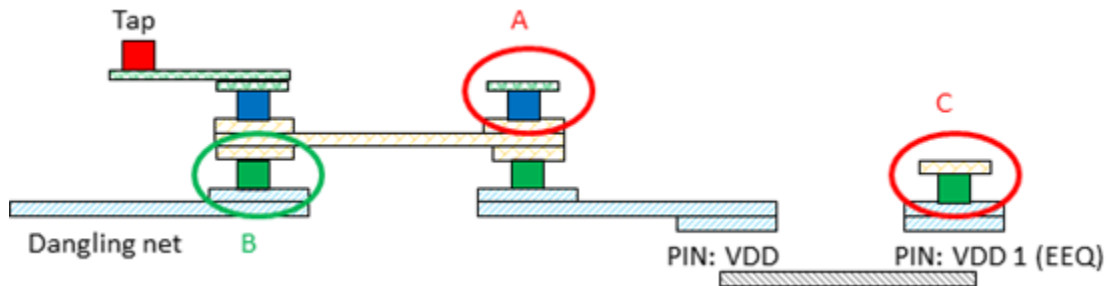


Dangling Vias

In PrimeRail, dangling vias are vias with one open-ended enclosure. For most cases, dangling vias are the result of removing the upper or lower metal layer during routing. Dangling vias are unwanted vias and should be removed from the design.

As shown in [Figure 6-13](#), PrimeRail reports only dangling via A as a dangling via error. PrimeRail does not flag dangling via C as a dangling via error; instead, it is a floating via.

Figure 6-13 An Example of Dangling Vias



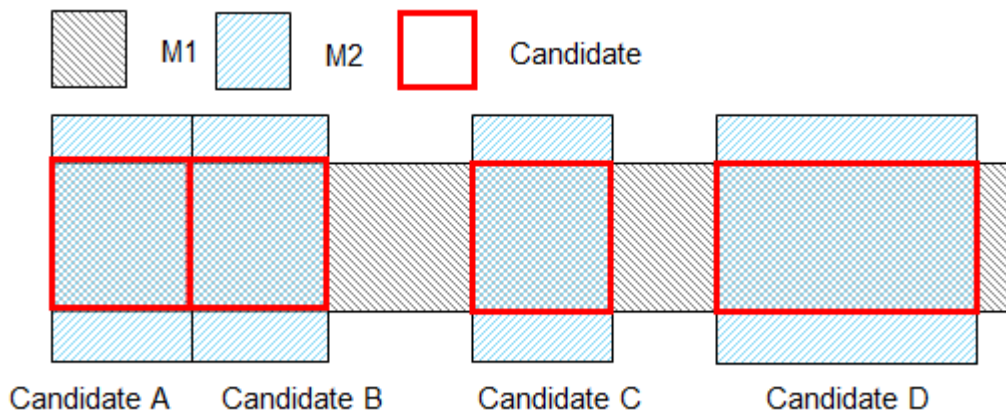
See Also

- [Integrity Checking Types](#)

Missing Vias

If two overlapping metal shapes contain no via within the overlapping area, PrimeRail considers it a potential missing via error, as shown in [Figure 6-14](#).

Figure 6-14 An Example of Missing Vias



The following sections describe how the engine detects missing vias with different considerations:

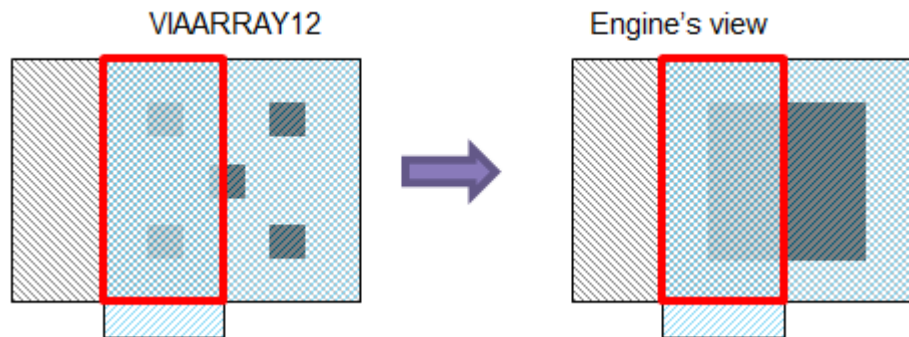
- [Checking for Fully or Partially Enclosed Vias](#)
- [Modeling Via Cuts by Bounding Boxes](#)
- [Checking for Stacked Vias](#)
- [Ignoring Via Enclosure Metals as Endpoints](#)
- [Allowing Endpoints Outside the Overlapping Areas](#)

- [Merging Same Layer Shapes](#)
- [Cutting a Merged Shape into Rectangles](#)
- [Specifying a Maximum Distance Between Vias](#)
- [Allowing Small Areas](#)

Checking for Fully or Partially Enclosed Vias

By default, PrimeRail assumes via cuts are fully enclosed when checking for missing vias, as shown in [Figure 6-15](#). To allow via cuts that are partially enclosed in the design, use the `-via_existence_check_partially_enclosed` option with the `check_supply_net_integrity` command.

Figure 6-15 Skipping Via Enclosure During Missing Via Checking



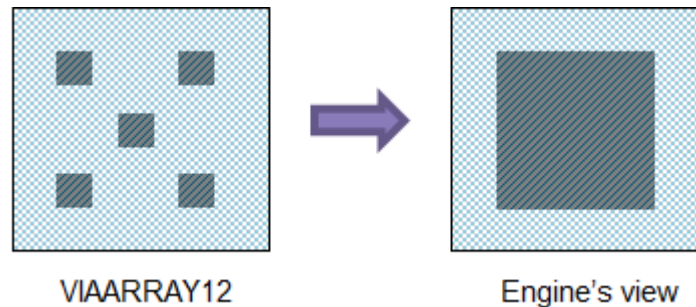
PrimeRail considers this a missing via error if the `-via_existence_check_partially_enclosed` option is disabled.

Modeling Via Cuts by Bounding Boxes

When checking for non-fully enclosed via arrays, the integrity checking engine models the via cut of a via array by its bounding box (as shown in [Figure 6-16](#)). To create a single virtual via cut using the bounding box to model the via cut, use the `-via_existence_check_merge_via_cut` option of the `check_supply_net_integrity` command.

If you use a bounding box to model a via cut, PrimeRail reports this overlapping area as a potential missing via error.

Figure 6-16 An Example of Using Bounding Boxes to Model Via Cuts



Checking for Stacked Vias

When checking for stacked vias, the PrimeRail integrity checking engine groups adjacent layer vias whose via cuts overlap with each other.

To enable the stacked via checking capability, run the `check_supply_net_integrity` command with the `-existing_stacked_via_check_mode` option.

To disable stacked via checking and check for missing vias only in adjacent metal routing layers, use the `-no_stack` option.

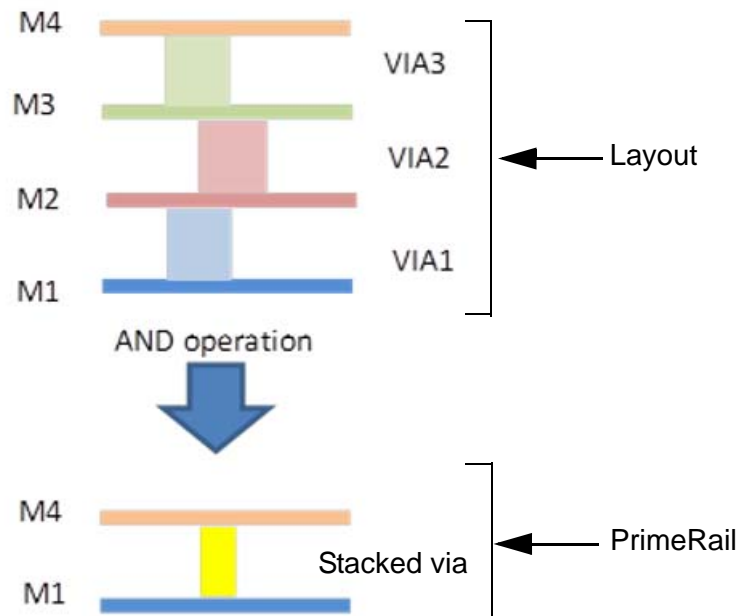
The stacked via checking provides the following modes:

- The `via_cut` Mode

To check for stacked vias that are vertically aligned, set the `-existing_stacked_via_check_mode` option to `via_cut`.

For example, in [Figure 6-17](#), the engine collects shapes VIA1, VIA2 and VIA3, and then does an AND operation to generate an internal “stacked via model.” This model infers the existence of a set of vias in a stacked formation, and the tool identifies this as a stacked via.

Figure 6-17 Checking for Stacked Vias Using `via_cut` Mode



- The `via_enclosure` mode

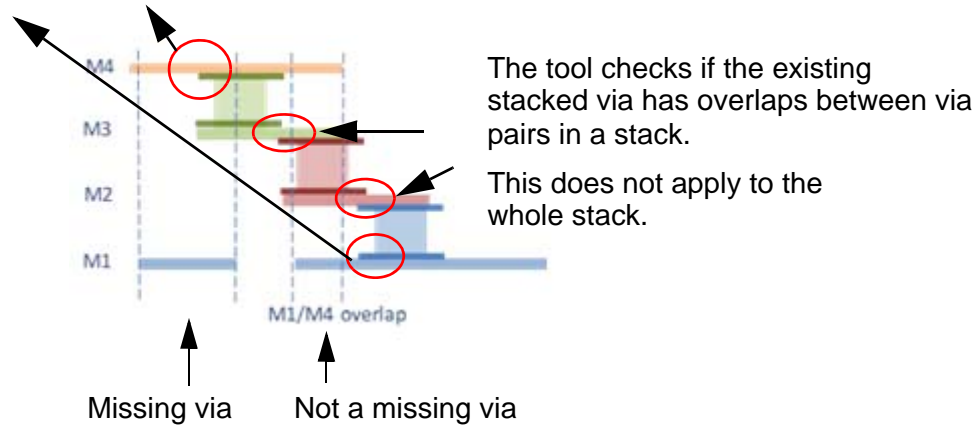
When the metal enclosure shapes of two vias on the adjacent via cut layers overlap with each other, these two vias are identified as stacked vias in this mode.

To check for staircase stacked vias, set the `-existing_stacked_via_check_mode` option to `via_enclosure`.

For example, in [Figure 6-18](#), the integrity engine collects shapes and their metal enclosures, and identifies stacked via locations by touching closure shapes. When you specify the `-allow_stacked_endpoint_vias_outside_of_overlapping_area` option, only one missing via is reported because the engine looks for missing vias outside the overlapped area. When not specified, the two M1/M4 overlap areas are reported as missing via errors.

Figure 6-18 Checking for Stacked Vias Using *via_enclosure Mode*

The tool checks if the existing stacked via has overlaps between endpoint vias and endpoint shapes.



Output Report

By default, PrimeRail writes only the information of the FROM and TO layers for a missing via on a stack via in the report (see [Example 6-1](#)). To indicate on which via layer the missing via is detected, use the `-report_specific_missing_via_in_stacked_via` option with the `check_supply_net_integrity` command.

Example 6-1 Missing Via Error Report Without the `-report_specific_missing_via_in_stacked_via` option

```
1: Layer: M2(12)-M4(14)          Type: Missing Vias
Net type: Ground
Type Summary: Possible missing vias for net VDD
Obj Info: Missing via between layers M2 and M4
         Net: VDD
Error ID: 1 Status: Error
Bbox: (123.123 456.456) (123.321 456.654)
```

Example 6-2 Missing Via Error Report With the `-report_specific_missing_via_in_stacked_via` option

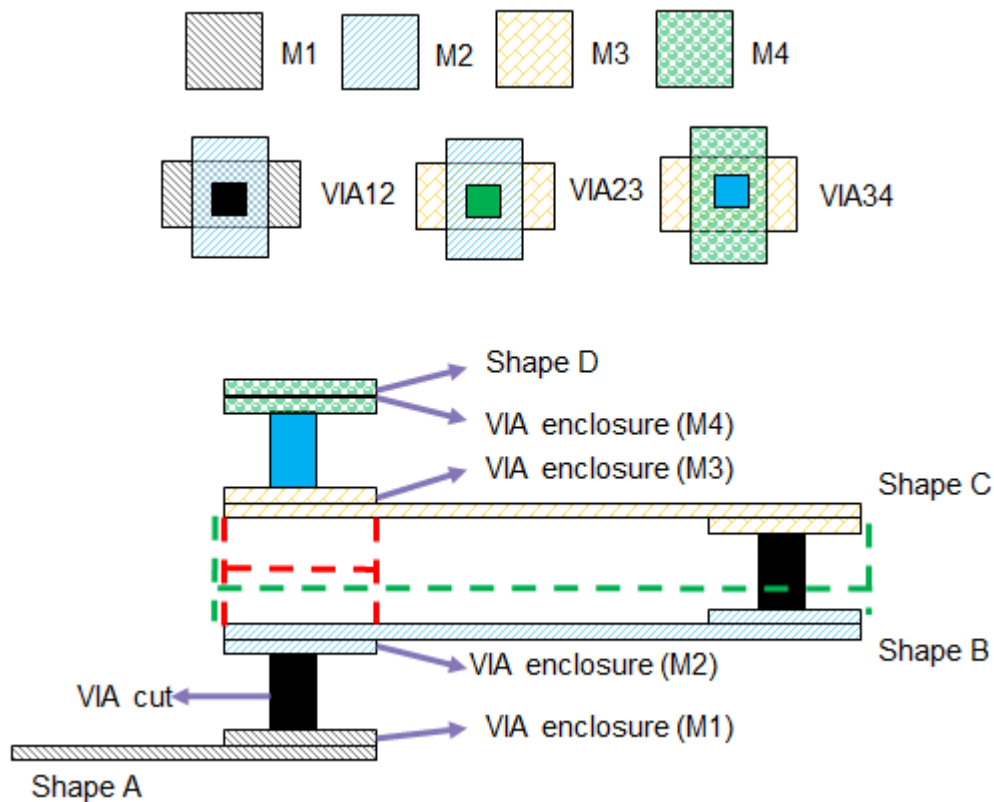
```
Layer: M2(17)-M5(20)          Type: Missing Vias
Net type: Power
Type Summary: Possible missing vias for net vdd
Obj Info: Missing via layers in the stack: M3-M4(V3)
         Net: vdd
Error Id:1 Status: Error
Bbox: (123,000 456.000) (321.000 654.000)
```

Ignoring Via Enclosure Metals as Endpoints

By default, the integrity checking engine uses via enclosure metals as overlapping shape endpoints, which might cause extraneous missing via errors. To avoid this, use the `check_supply_net_integrity -ignore_via_enclosure_shape` command.

For example, in [Figure 6-19](#), PrimeRail reports four missing via errors between Metal 2 and Metal 3 when you do not specify the `-ignore_via_enclosure_shape` option. The errors are via enclosure M3 versus via enclosure M2, shape C versus via enclosure M2, via enclosure M3 versus shape B, and shape C versus shape B.

Figure 6-19 Skipping Via Enclosure During Missing Via Checking



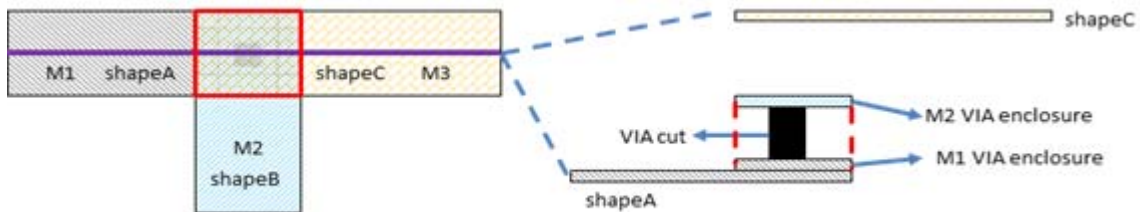
Allowing Endpoints Outside the Overlapping Areas

By default, the engine requires endpoint via cut layers to be physically connected to and within the overlapping area. To allow the endpoint of a stacked via exceeds the overlapping area, use the `-allow_stacked_endpoint_vias_outside_of_overlapping_area` option.

Merging Same Layer Shapes

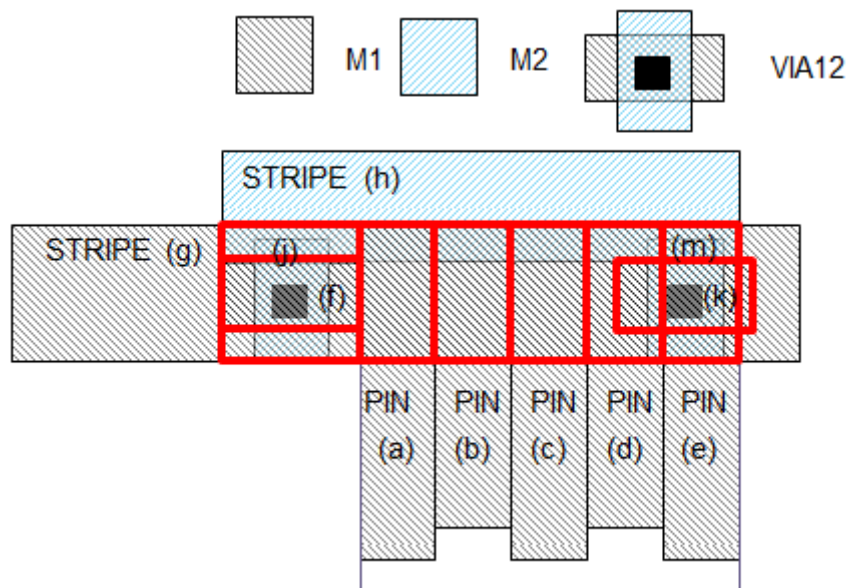
In [Figure 6-20](#), a dangling via is in a location where you might want to create a stacked via between Metal 1 and Metal 3. If you always skip a via enclosure, you might not find this kind of error if shape A and shape C are too small to form an overlapping area.

Figure 6-20 An Example of Dangling Vias



To address these potential problems, merge all the same layer shapes using the `check_supply_net_integrity -merge_same_layer_shapes` command before running missing via checking. See [Figure 6-21](#) for an example.

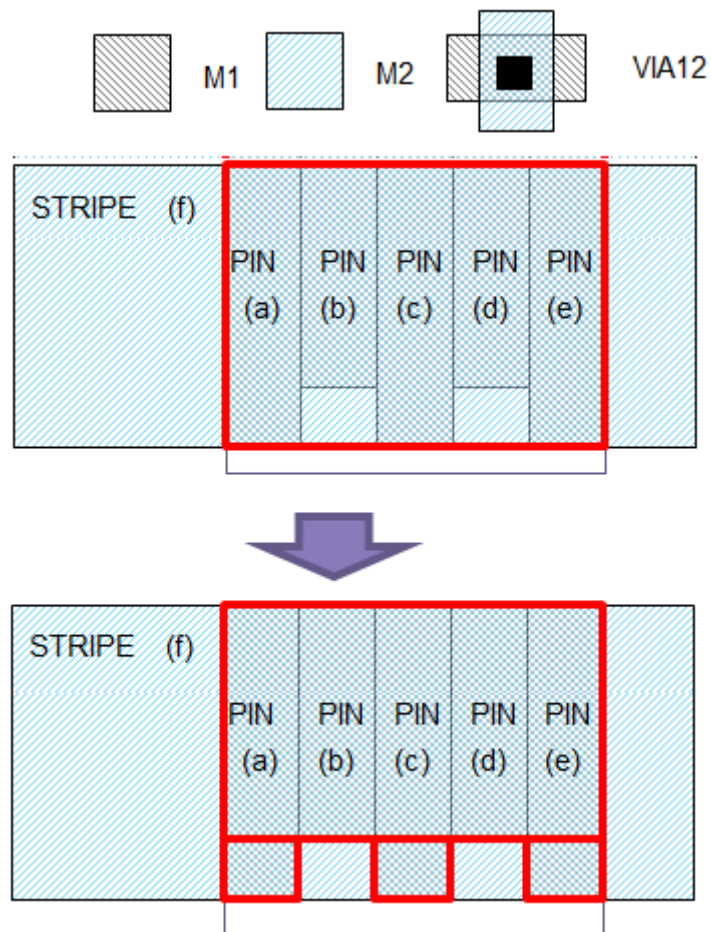
Figure 6-21 Layer Shapes Before Missing Via Checking



Cutting a Merged Shape into Rectangles

When checking the overlapping areas of two merged rectilinear shapes, the tool returns a rectangular bounding box and changes the original rectilinear shape of the overlapping area. To retain the original error rectilinear shape, use the `-cut_merged_shape` option of the `check_supply_net_integrity` command to cut a returned rectilinear shape into multiple error rectangles, as shown in [Figure 6-22](#). By default, PrimeRail reports bounding boxes of the merged shapes.

Figure 6-22 Cutting Merged Shapes



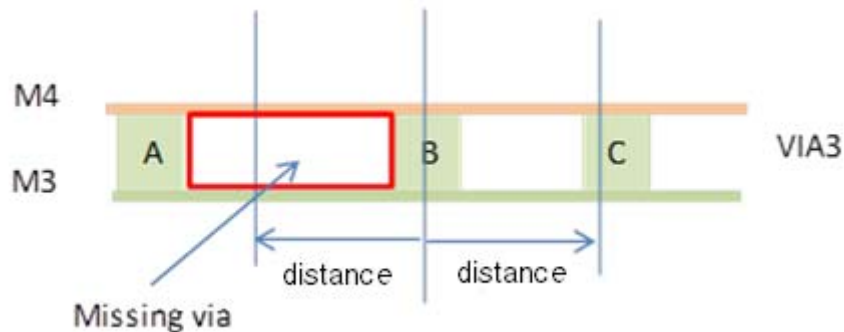
Specifying a Maximum Distance Between Vias

Use the `-max_via_distance_along_parallel_wires distance_threshold` option of the `check_supply_net_integrity` command to check for missing vias only when the distance between two neighboring vias along the direction of overlapping parallel wires is larger than the specified `distance_threshold` (as shown in [Figure 6-23](#)).

Note:

When the `-max_via_distance_along_parallel_wires` option is specified, the engine groups the errors specifically under the `MissViaMaxDist` category in the error browser.

Figure 6-23 An Example of Setting a Maximum Distance Between Parallel Wires



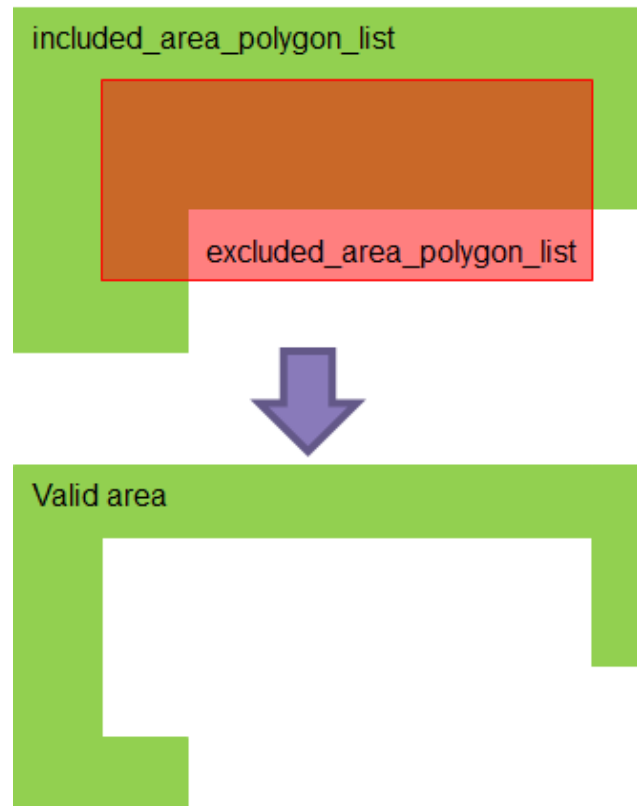
Allowing Small Areas

By default, the tool checks if the overlapping area is larger than the via size of the default contact code. Areas that are smaller than the minimum sized via are not identified as missing via candidates. To ignore this criterion, use the `-ignore_minimum_via_criterion` option. Specifying this option might increase the number of false checking errors.

Area-Based Filtering

During integrity checking, PrimeRail provides the `-included_area_polygon_list` and `-excluded_area_polygon_list` options to create rectilinear areas. The rectilinear area is based on subtracting the excluded area from the included area. If you do not specify the included area, PrimeRail uses the cell area of the entire design. Only errors inside the specified area are reported. You can define the polygon list either clockwise or counterclockwise. [Figure 6-24](#) illustrates how the specified area is derived.

Figure 6-24 Area-Based Filtering Checking



Toggleing Between Integrity Results

Whenever the `check_supply_net_integrity` command is run successfully, the tool automatically creates a named integrity result in the cache. This result context contains the data created by the command and a snapshot of all taps created in the session. PrimeRail retrieves the data from this current integrity result to display the integrity errors in the error browser.

Use the `current_integrity_result` command to load and set a result that was previously generated in the same session as the current result. This allows you to toggle between two or more result sets for data comparison. Use the `create_taps -result` command to load taps into the current session from the current integrity result.

To see which integrity results already exist in the current session for subsequent toggling, use the `report_integrity_results` command.

Run the `current_integrity_result -clear` command to unassign the current integrity result without replacing it with a different integrity result, as if the `check_supply_net_integrity` command was not executed. The unassigned current

integrity result still resides in the session and you can reload it with the `current_integrity_result` command. To completely remove the integrity result from the memory, use the `remove_integrity_result` command.

To specify or change the name of an existing integrity result, use the `rename_integrity_result` command. If the specified new name is already applied to an existing result, use the `-replace` option.

See Also

- [Integrity Checking](#)
- [Reporting Integrity Results](#)
- [Deleting Integrity Results](#)

Reporting Integrity Results

Run the `report_integrity_results` command to report the integrity results within the session.

By default, the command lists only the names of the integrity results. Use the `-verbose` option to include the number of integrity checking errors and a snapshot of taps recorded when this result is created. Use the `-summary` option to include information about how many errors are in the integrity result.

Example

The following example shows a simple report for the `MY_NAMED_INTEGRITY_RESULT` and `INTEGRITY_RESULT_1` results:

```
pr_shell> report_integrity_results
*****
Report integrity check results
Design : micro
Version: <Version>
Date   : <Date>
*****
    MY_NAMED_INTEGRITYRESULT
    INTEGRITY_RESULT_1
```

The following example shows a summarized report that includes information about how many errors are reported in the integrity result:

```
pr_shell> report_integrity_results -summary
*****
Report : integrity check results
Design : micro
Version: <Version>
```



```
Date      : <Date>
*****
MY_NAMED_INTEGRITY_RESULT
28 missing vias errors, 2 floating geometry errors
INTEGRITY_RESULT_1
2 floating geometry errors
```

The following example shows a verbose report that includes information about how many errors are in the integrity result and a snapshot of taps recorded when the result is created:

```
pr_shell> report_integrity_results -verbose
*****
Report : integrity check results
Design : micro
Version: <Version>
Date   : <Date>
*****
MY_NAMED_INTEGRITY_RESULT
28 missing vias errors, 2 floating geometry errors
Taps in the result: int1 int2
INTEGRITY_RESULT_1
2 floating geometry errors
Taps in the result: int3 int4 int5
```

See Also

- [Rail Integrity Checking](#)
- [Toggling Between Integrity Results](#)

Deleting Integrity Results

Use the `remove_integrity_result` command to remove an integrity result data set from the current session. Integrity results are generated by the `check_supply_net_integrity` command and are saved to disk during the session. These integrity results are automatically deleted when you exit the session.

If you execute multiple `check_supply_net_integrity` commands in the session, you might have results that are no longer useful but still show up in the reports and consume disk space. Use the `remove_integrity_result` command to remove the integrity results that are no longer needed. Then, run the `report_integrity_results` command to check which integrity results are still available in the current session.

See Also

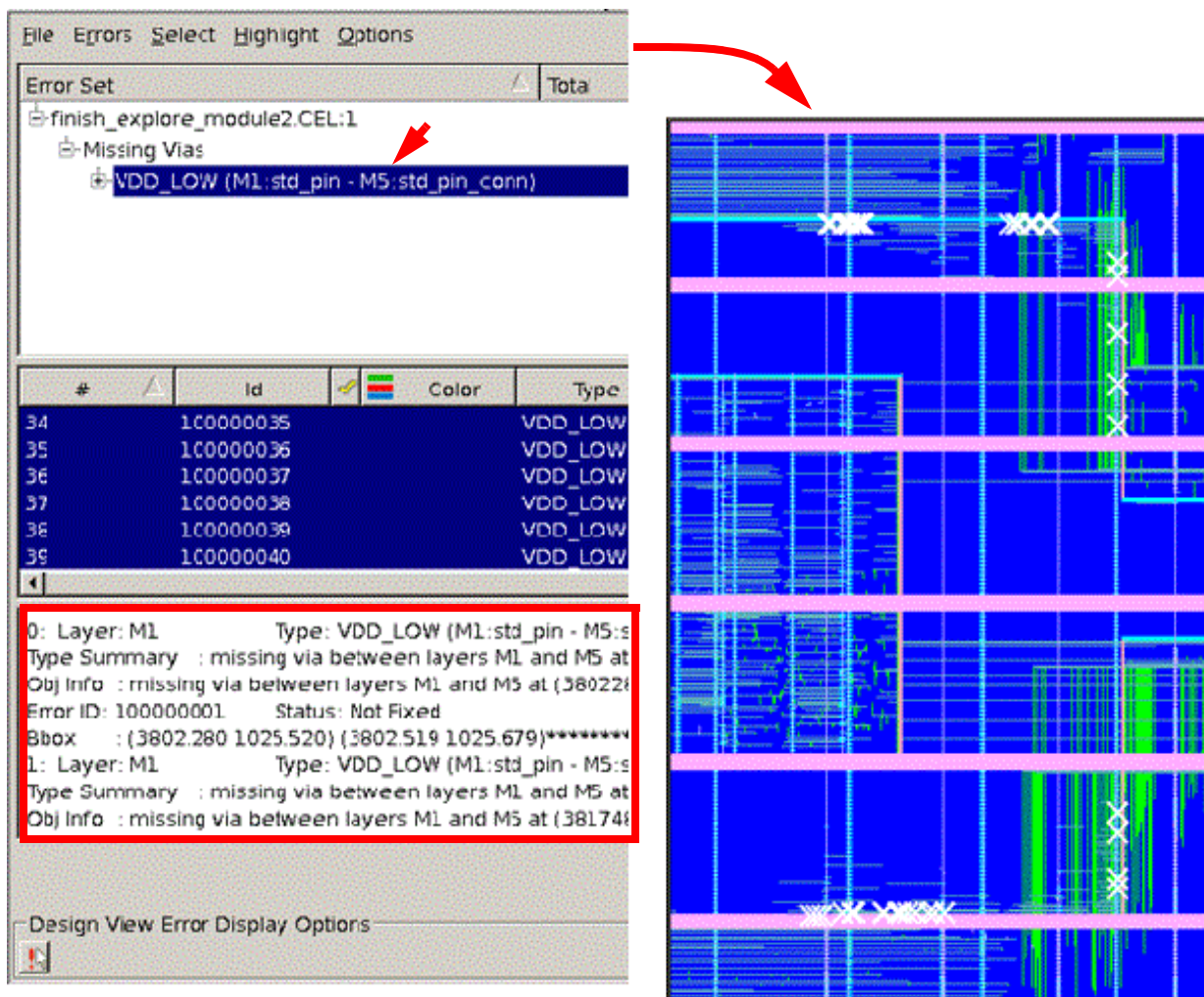
- [Rail Integrity Checking](#)

Viewing Integrity Errors in the Error Browser

If you save the integrity analysis results to an error view during the `check_supply_net_integrity` command run, you can open the result in the error browser for checking the errors generated during integrity checking. You must set the property of the current library to be writable using `set_mw_lib_write_access` command before you save the integrity results to an error view.

Alternatively, you can open and view the generated error views in the IC Compiler error browser. This allows you to check and fix the errors in the GUI. For more information about viewing integrity errors in the IC Compiler error browser, see the related chapter in the *IC Compiler Implementation User Guide*.

Figure 6-25 Viewing Integrity Errors in the Error Browser



As shown in [Figure 6-25](#), select the Missing Vias M1:M5 and all the errors in the middle error list in the error browser. The errors are highlighted in the layout. The details of the errors are also reported in the lower pane.

See Also

- [Rail Integrity Checking Flow](#)
- [Rail Integrity Checking](#)

7

Extracting Supply Net Parasitics

PrimeRail requires the geometries and resistances of the power and ground networks in the database to perform rail analysis. The `extract_supply_parasitics` command performs extraction on the specified power or ground nets.

You can run supply net extraction and power analysis concurrently on different available CPU cores. The multicore processing feature provides quick turnaround time and improved performance. For more information about using the multicore processing feature, see [Enabling Multicore Processing](#).

This chapter contains the following sections:

- [Setting Extraction Options](#)
- [Extracting Supply Net Parasitics](#)
- [Handling Via Arrays](#)
- [Hierarchical Extraction](#)
- [Determining PG Shape Overlaps](#)
- [Multicore and Parallelism](#)
- [Reporting Power and Ground Net Hierarchy](#)
- [Viewing Parasitic Maps](#)
- [Reusing Power and Extraction Data](#)

Setting Extraction Options

Before performing supply net extraction, use the `set_extract_supply_parasitics_options` command (or Extract > Extract Options in the GUI) to configure settings on the TLUPlus files, ITF map files, and temperatures that are needed for running the `extract_supply_parasitics` command.

Specify a temperature value in degrees Celsius for the supply net extraction. By default, the nominal temperature from the default operating condition in the first library in the link path is used. The nominal values in the TLUPlus file are derated according to the coefficients and the nominal temperature from the default operating condition in the first library. The allowed temperature values are -300.0 through +500.0.

You can run the `set_extract_supply_parasitics_options` command multiple times to set the options incrementally. To report the settings of these setup options, use the `report_extract_supply_parasitics_options` command.

Here are the options that are commonly used when specifying settings for parasitic extraction:

- (For Milkyway designs) Use the `-max_tluplus` and `-tech2itf_map` options to specify the TLUPlus model and layer mapping files for supply net parasitic extraction.

For example,

```
pr_shell> set_extract_supply_parasitics_options \
-max_tluplus my.tluplus \
-tech2itf_map my.map -temperature 25.00
```

- (For IC Compiler II designs) Use the `read_parasitic_tech` command to specify the locations of the TLUPlus files.

For example,

```
icc2_shell> read_parasitic_tech -tlup c1n10t_1p12m_typ.tluplus
-layermap layermap.txt -name test1
icc2_shell> set_parasitic_parameters -late_spec test1
icc2_shell> save_block
icc2_shell> save_lib
```

Then complete the parasitic settings with the `set_extract_supply_parasitics_options` command in PrimeRail.

```
pr_shell> set_extract_supply_parasitics_options -parasitic_tech test1
```

For more information, see the Specifying the Parasitic Technology Information topic in *IC Compiler II Timing Analysis User Guide*.

- Use the `-skip_metal_layer skip_metal_file` option to specify a metal layer or a hard macro to be excluded from extraction. When specified, the command skips the specified layer along with the layers that are lower than the specified layer.

In the following metal file example, the tool skips the metal2 layer and the lower layers for cell master SRAM8x16, and the metal1 layer and the lower layers for cell master ALU during extraction.

```
;metal_layer cell_master
2 SRAM8x16
1 ALU
```

- Use the `-include_shapes list_additional_shapes` option to include the special shapes in the standard cells for extraction.

The following example includes the vias inside standard cells for extraction.

```
pr_shell> set_extract_supply_parasitics_options \
           -include_shapes {stdcell_vias} \
           ...
```

- Use the `-via_array_partition_size {{via1 n1} {via2 n2} ...}` option to break a large square or a rectangular via array into small via arrays based on the specified via mask name and via array dimension. The via mask name can be ALL, meaning all via layers are applied with the same partition size setting. By default, via array clustering interprets via arrays as one big resistor.
- Use `-derive_macro_pin_shapes true|false` option to control whether PrimeRail gets physical block instance supply pin shapes from its FRAM view (by setting to `false`) or derives all the overlaps from the block and top shapes by ignoring the FRAM view pin shapes (by setting to `true`).

Use this option to determine the supply pin shapes based on the supply shape overlaps between the enclosing net shapes and the physical block instance net shapes. The default is `false`.

- Use the `-implicit_port_connection mw_cell_type_list` option to connect the pin shapes that are implicitly but not explicitly connected for black box cells based on the EEQ class setting.

When a cell does not have a macro model, PrimeRail uses its FRAM view as its macro model. A FRAM view is an abstract model and might not model its pin shapes connection explicitly.

- Use the `-exclude_of_implicit_port_connection cell_master_name_list` option to exclude the specified cell masters from the implicit port connection.

See Also

- [Extracting Supply Net Parasitics](#)

Extracting Supply Net Parasitics

Use the `extract_supply_parasitics` command (or Extract > Extract Supply Parasitics in the GUI) to extract the RC network of the power and ground nets. The command extracts the supply net parasitics and saves them on the disk in the /pgext directory of the specified RAIL_DATABASE directory. Use the `read_supply_parasitics` command to load the parasitics back into the session.

The `extract_supply_parasitics` command

- Extracts the parasitics of the power and ground nets inside a soft or hard macro from the top level.
- Extracts the parasitics of all the supply nets that are connected to the specified nets through multithreshold-CMOS switch cells (also known as power management cells), so you do not need to identify the full set of related supply nets. If you do not want to include the full set of related supply nets, use the `-noexpand` option. For best performance, you might list all supply net names in one command call, instead of extracting one net at a time, respectively.
- Extracts the parasitics for a soft macro design with a power management cell which has virtual power and ground nets internal to the macro block. However, the tool does not extract a hard macro that contains a power management cell, such as a memory cell with the low power function.
- Considers vias inside a standard cell with dual rail connections. For example, when a standard cell has PG pins on both M1 and M2 metal layers, and each PG pin connects to its parent cell's net shapes individually, the command extracts both M1 and M2 pins and their connected vias. The vias are grouped and extracted as one edge and the two pins are extracted as two nodes. Upon the completion of extraction, the command creates a current source node on the M1 pin.

When standard cells are connected to each other through abutment, the command extracts the M1 pin "trunk" shape and creates one current source node on it. The "trunk" shape of a pin is the main body rectangle part of the pin, usually touching the top or bottom boundary of the standard cells. The command ignores the details inside the standard cells, called fingers, during extraction.

- Extracts the parasitics on the nwell, pwell and well contacts, and generates current sources at the center of the body-bias pins.

See [Considering Body-Bias Effects](#) for a detailed description about how to include the body bias effects during parasitic extraction and rail analysis.

PrimeRail saves the parasitic data files (per master and per net) to the local working directory in binary format when the extraction process is complete. You must load the

extraction data into memory with the `read_supply_parasitics` command before proceeding to the rail analysis step.

The following is an example of the extraction summary:

```
Parasitic file output directory: /RAIL_DATABASE/pgext/para
Start PG extraction for all the masters...
Please refer to detail log file /RAIL_DATABASE/pgext/log/
RedirectedLog.pgextDetails for PG extraction progress and detail
messages.

Information: PG Parasitics Extraction Summary
4 net(s) extracted successfully, 0 net(s) failed.
-----
Net_Name                                Status
VDD                                     Done
VDDS_p0                                Done
VDDS_misc                               Done
VDDS_p1                                 Done
Information: Total PG extraction  Memory: 356.566 MB  CPU: 163 seconds
Elapse: 166 seconds
**End Of Extractor Binary**

Information: 356.566 MB PG extraction memory is counted.
```

Extraction Results and Log Files

When supply net extraction is complete, PrimeRail saves the parasitic data and related log files in the `RAIL_DATABASE/pgext` directory. By default, the rail database directory is created in your working directory. To set a different location and directory name, use the `set rail_database dir_name` command.

When extraction is complete, PrimeRail generates the following data:

- Parasitic results

Saved in the `/RAIL_DATABASE/pgext/para` directory, each parasitic data file is named with a `.para` extension. One `.para` file is created for each supply net.

Here is an example of the parasitic data file:

```
/RAIL_DATABASE/pgext/para:
index.reg

##Para file for each PG net extracted##
LIB->finish_voltdrop_VDD_125.000_RC.para
LIB->finish_voltdrop_VDDS_misc_125.000_RC.para
LIB->finish_voltdrop_VDDS_p0_125.000_RC.para
LIB->finish_voltdrop_VDDS_p1_125.000_RC.para

#Para file for memory PG net(s)#
ram_test->SRAM32x256_rw_VDD_125.000_RC.para
ram_test->SRAM32x64_VDD_125.000_RC.para
```

```
ram_test->SRAM39x32_VDD_125.000_RC.para
```

- Log files

Saved in the /RAIL_DATABASE/pgext/log directory, the log file includes an extraction summary, memory usage information, parasitic file location, and so on.

Here is an example of the detailed /RAIL_DATABASE/pgext/log file:

```
##Log file for extracting child processes##
RedirectedLog.pgextDetails

Information: Net <VSS> of cell <micro> has 2610 internal nodes, 2
boundary nodes, 2655 resistors, 2484 current sources
Information: Total net <VSS> cap = 0.0049349 nF.
Information: Sum of all the edge resistances on this net <VSS> inside
the master micro = 600.2689819 Ohm

Information: Net <VDD> of cell <micro> has 2906 internal nodes, 2
boundary nodes, 2991 resistors, 2484 current sources

Information: Total net <VDD> cap = 0.0059493 nF.
Information: Sum of all the edge resistances on this net <VDD> inside
the master micro = 2113.0588379 Ohm
```

See Also

- [Setting Extraction Options](#)
- [Hierarchical Extraction](#)
- [Determining PG Shape Overlaps](#)
- [Multicore and Parallelism](#)
- [Viewing Parasitic Maps](#)
- [Reusing Power and Extraction Data](#)

Handling Via Arrays

To break a large square or a rectangular via array into small square via arrays, use the `-via_array_partition_size` option of the `set_extract_supply_parasitics_options` command and specify the via mask name and via dimension for re-clustering. The specified sizing rule also applies to the via arrays inside the CONN view.

Here is the syntax:

```
set_extract_supply_parasitics_options \
  -via_array_partition_size \
  {{via_mask1 via_dimension} {via_mask2 via_dimension}}
```

For example, if you set the via dimension to 5, and the design has one 1 x 10 via array, one 10 x 1 via array and one 9 x 9 via array, the cutting results are as follows:

Original via layer	After cutting
1 x 10 via array	Two 1 x 5 via arrays
10 x 1 via array	Two 5 x 1 via arrays
9 x 9 via array	One 5 x 5 via array, one 4 x 5 array, one 5 x 4 via array, and one 4 x 4 array

If the `via_mask` is specified as ALL, all the via arrays in the design are resized following the defined sizing rule. By default, PrimeRail treats a via array as one big resistor during via array clustering.

Note:

A bitmap via array in the design is always merged into a super via. Re-clustering is not supported for a bitmap via array.

See Also

- [Setting Extraction Options](#)
- [Extracting Supply Net Parasitics](#)

Hierarchical Extraction

When analyzing a hierarchical design with macro cells, PrimeRail extracts parasitics for the cell masters and builds a flattened parasitic network with the `read_supply_parasitics` command. Power and ground pins are used to route the child block at the top level; they are marked as boundary nodes in the child block and as port instances at the top level. PrimeRail creates one boundary node for each well-defined small rectangle pin.

When extracting long pins, PrimeRail promotes long pins from a subblock to their parent block only, but not to their grandparent block. Consequently, the connectivity in the design might not be established between the top-level routing and the grandchild block.

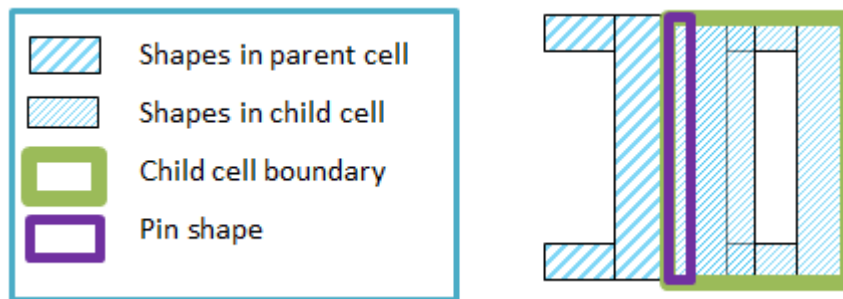
When the hierarchical design contains macro cells, PrimeRail

- Promotes all inter hierarchical boundary shapes as part of the net shapes in the parent cell
- Promotes as few shapes as possible
- Ensures all overlapped shapes are merged if they are to be promoted
- Ensures the boundary nodes in both child and parent cells are identical

By tracing the connectivity to the first non-intersected node encountered, PrimeRail ensures the same node is created in both the parent and child cells.

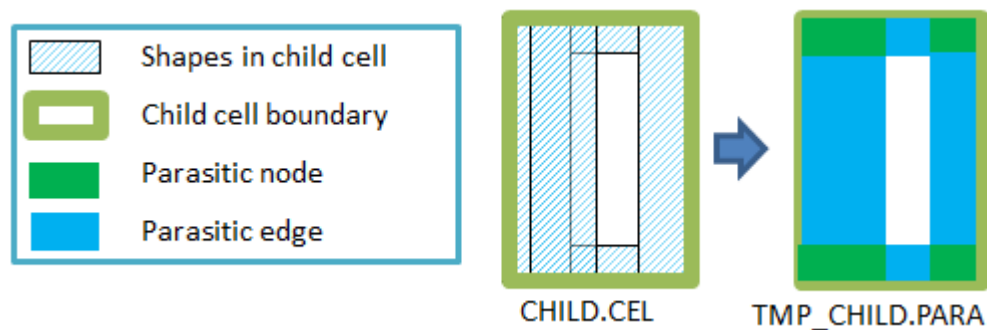
In [Figure 7-1](#), the boundary of the child cell is marked in green and the pin shape is marked in purple. When extracting the design, PrimeRail first creates a temporary parasitic network for the child cell, as shown in [Figure 7-2](#).

Figure 7-1 An Example of Extracting a Hierarchical Design



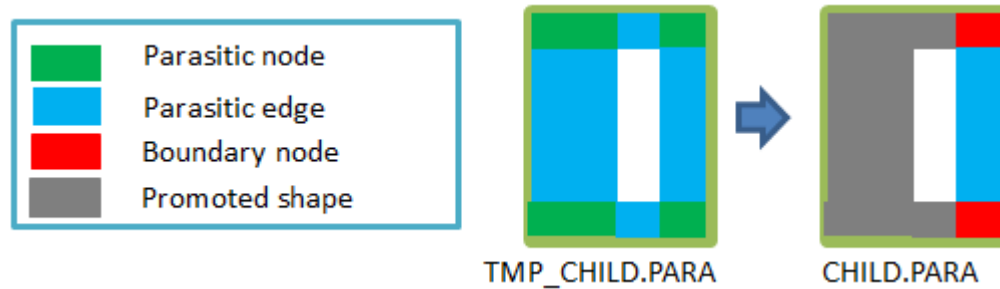
In [Figure 7-2](#), PrimeRail uses all the extracted parasitic nodes or edges that intersect with the pin shapes as sources for tracing connectivity in the parasitic network. PrimeRail stops tracing at the first non-pin-shape node it encounters in the path.

Figure 7-2 Creating the Parasitic Network for the Child Cell



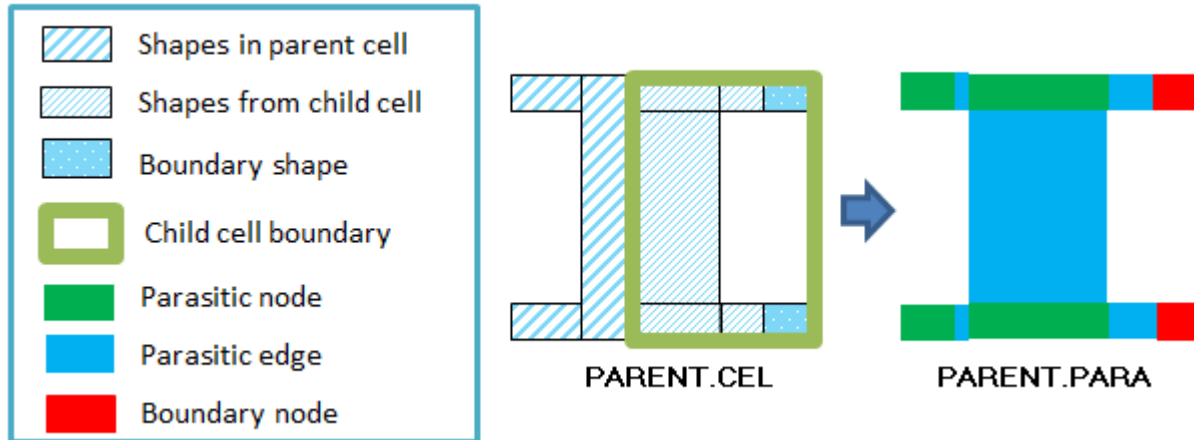
Each node and edge on the path are promoted to their parent cell. PrimeRail marks those non-pin-shape nodes as new boundary nodes in both the parent and child cells, as shown in [Figure 7-3](#). In other words, pin shapes are used as access points between the parent and child cells; this implies that all cut rectangles that intersect with pin shapes in both parent and child cells are likely to be merged with rectangles. PrimeRail always promotes these rectangles to the top level and does not use them as boundary nodes.

Figure 7-3 Creating Boundary Nodes



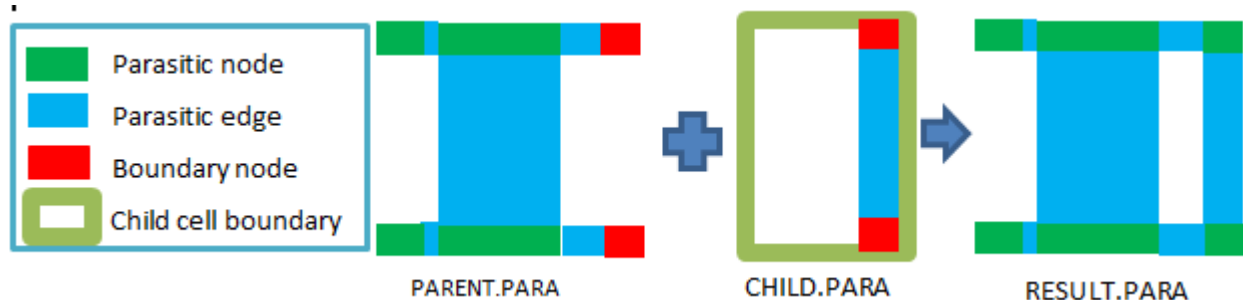
Next, PrimeRail extracts the parent cell and considers the promoted shapes part of the net shapes. The promoted boundary shapes are marked as boundary nodes, as shown in Figure 7-4.

Figure 7-4 Creating Boundary Nodes for the Promoted Boundary Shapes



When extraction is complete, PrimeRail merges all boundary nodes in the parent and child cells and creates a flattened parasitic network with the `read_supply_parasitics` command.

Figure 7-5 Merging Boundary Nodes to Generate a Flattened Parasitic Network



See Also

- [Setting Extraction Options](#)
- [Extracting Supply Net Parasitics](#)

Determining PG Shape Overlaps

During full-chip extraction, PrimeRail hooks up the block parasitics to the full-chip parasitics based on the pin shape locations in the FRAM view. Sometimes the current terminal information in a block FRAM view (and the related pin shapes on the instances) might not overlap with other enclosing shapes in the physical hierarchy but the shapes inside the block do. In this case, set the `-derive_macro_pin_shapes` option of the `set_extract_supply_parasitics_options` command to `true` to ignore the FRAM view pin shapes and to derive all the overlaps from the block and top shapes automatically.

By default, PG extraction ignores the FRAM view pin shapes and determines these supply pin shapes based on the supply shape overlaps between the parent cell and the physical block net shapes.

See Also

- [Setting Extraction Options](#)
- [Extracting Supply Net Parasitics](#)

Multicore and Parallelism

PrimeRail supports multicore processing, whether through distributed processing or multithreading. Multicore processing improves turnaround time by performing tasks in parallel much more quickly than if they were run in serial on a single core.

To enable parallel extraction in multicore machines, use the `set_host_options -max_cores number_of_cores` command to control the number of parallel processes. By default, the number of cores is set to 4 for best performance. Half of the available cores are used for power and ground extraction child process and the other half are used for the subsequent power analysis and current calculation runs.

The following log example shows how PrimeRail handles distributing processes in a multicore run:

```
Information: running timing/power analysis and p/g extraction in
parallel.
Information: p/g extraction is going to run on CPU(0,3)
Information: timing/power analysis is going to run on CPU(1,2)
```

In a multicore analysis run, PrimeRail performs extraction in the child process and timing and power analysis in the main `pr_shell` session process. Use the following options to control the process:

- Use the `-wait` option to complete extraction on all the specified nets before proceeding to the power analysis stage.
- Use the `-nowait` option to return without finishing all the extraction processes. In this case, the extraction process is running in the background, and you can proceed to the subsequent step in the analysis flow, like power analysis.

For best turnaround time, you run extraction immediately after opening the design.

If neither the `-wait` nor `-nowait` option is specified, the tool decides whether to run extraction in the background or foreground. PrimeRail computes the available system memory and the estimated memory consumption for extraction and power analysis. If the system memory is sufficient, the tool runs extraction and power analysis in parallel in the `-nowait` mode.

See Also

- [Extracting Supply Net Parasitics](#)

Extraction and Power Analysis in Parallel

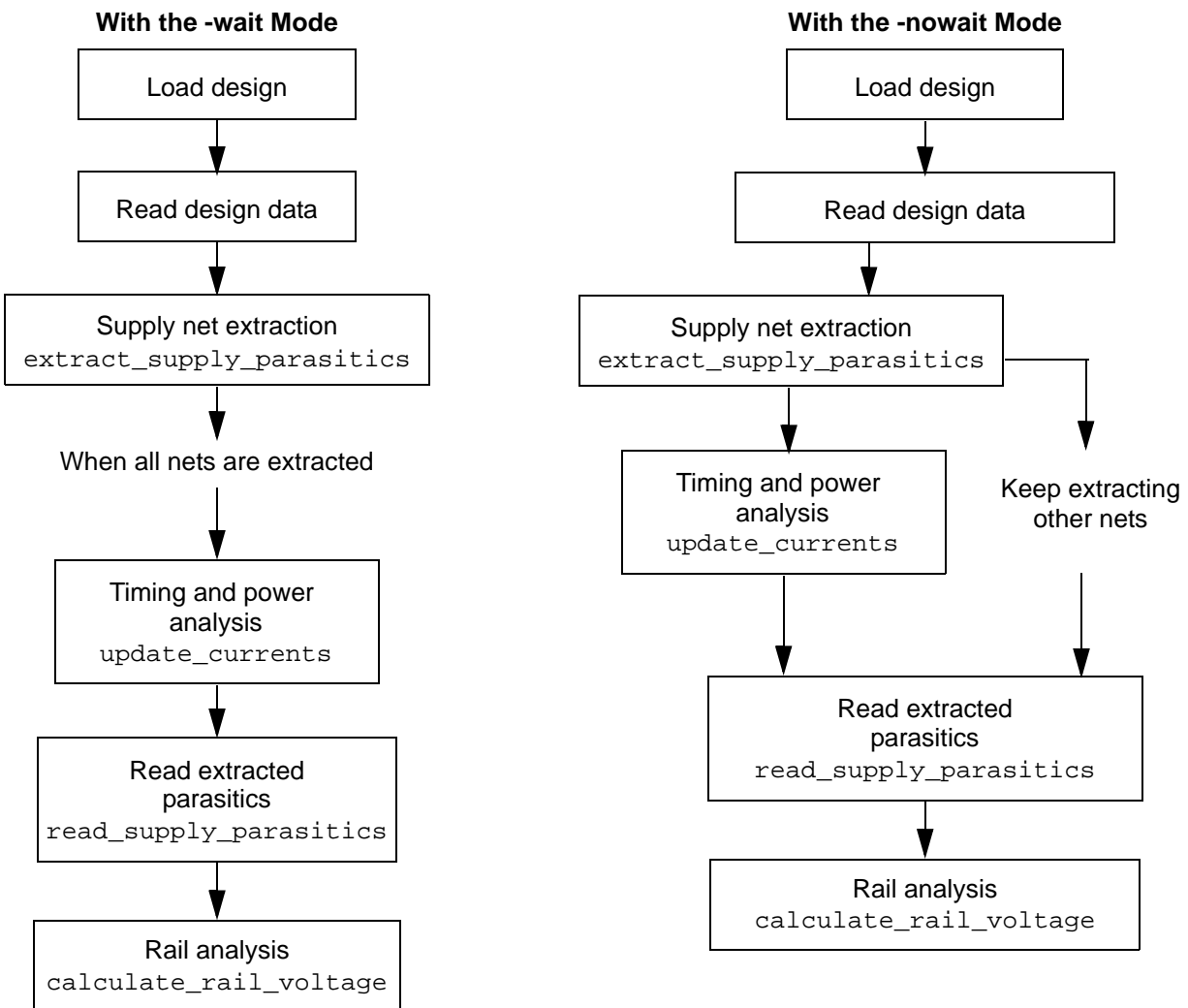
If multiple supply net extractions are to be done on multicore machines, PrimeRail supports the multithreading multicore capability that allows you to perform supply net extraction in parallel with timing and power analysis. This reduces overall runtime on multicore machines. For more information about multicore extraction, see [Multicore and Parallelism](#).

By default, PrimeRail launches a separate process in the background whenever PrimeRail encounters the `extract_supply_parasitics` command in a script. The tool keeps executing the rest of the commands until the `read_supply_parasitics` command is executed. It is recommended that you use the `read_supply_parasitics` command to control the cross-command parallelism because you do not need to change your script.

PrimeRail provides the `update_currents` command for running timing and power analysis and the `extract_supply_parasitics` command for supply net extraction.

[Figure 7-6](#) illustrates how timing and power analysis and extraction run in parallel on a single machine.

Figure 7-6 Running Extraction and Power and Timing Analysis in Parallel in PrimeRail



See Also

- [Extracting Supply Net Parasitics](#)

Reporting Power and Ground Net Hierarchy

When an extraction process is complete, PrimeRail lists the hierarchical structure of the power and ground nets in the log file. This helps you to confirm the design hierarchy and power and ground net connectivity when an analysis run is complete.

PrimeRail also provides the `report_extraction_hierarchy` command for reporting power and ground netlist hierarchy during extraction. Use the reported information to debug the issues found in the analysis run.

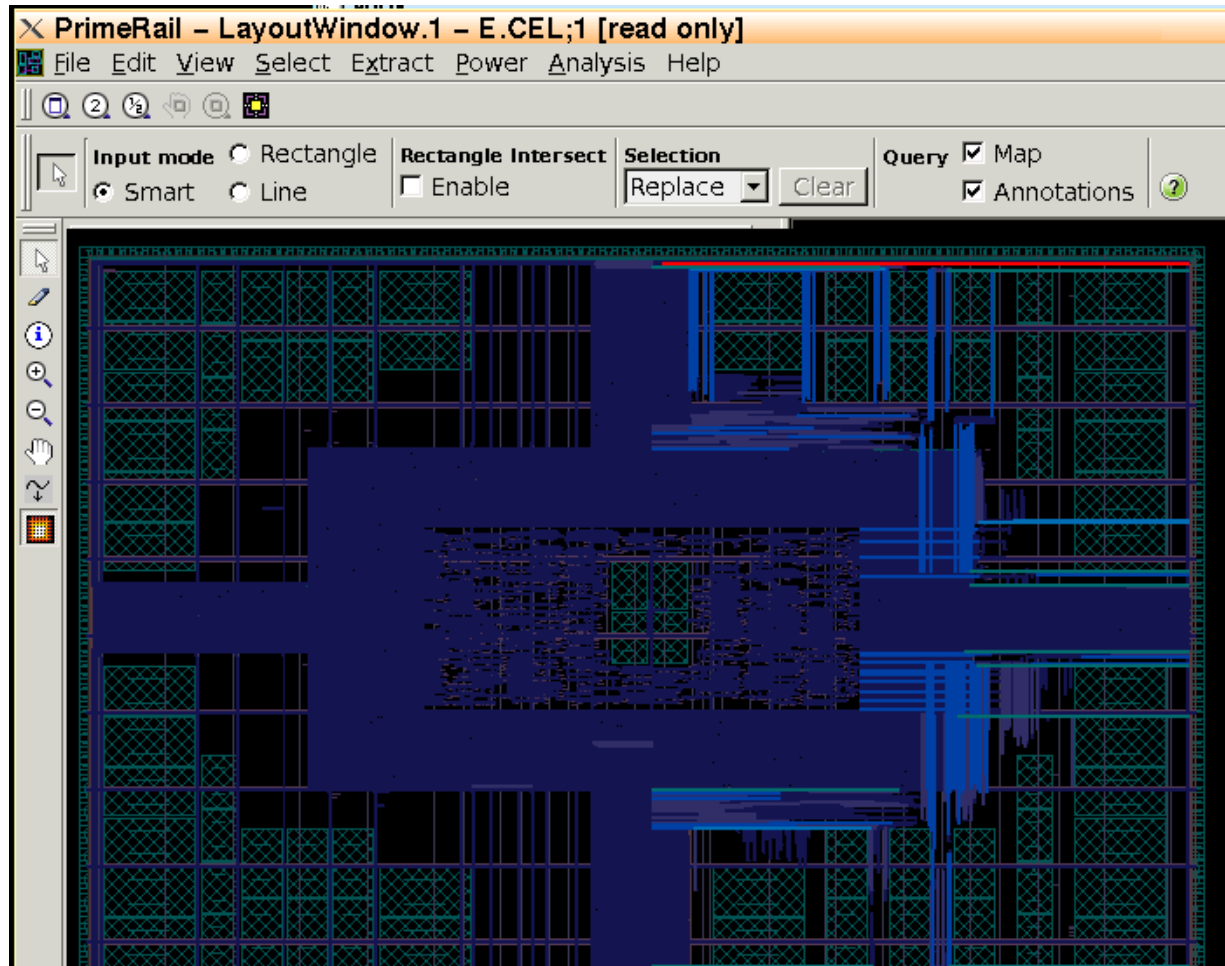
See Also

- [Extracting Supply Net Parasitics](#)

Viewing Parasitic Maps

When the extraction run is complete, you can display the parasitic map to view the parasitic data.

[Figure 7-7](#) shows an example of the parasitic map. In the map, the extracted geometries are highlighted in different colors. Use the info-tip tool to query the resistance and capacitance values of the geometries.

Figure 7-7 A Parasitic Map Example**See Also**

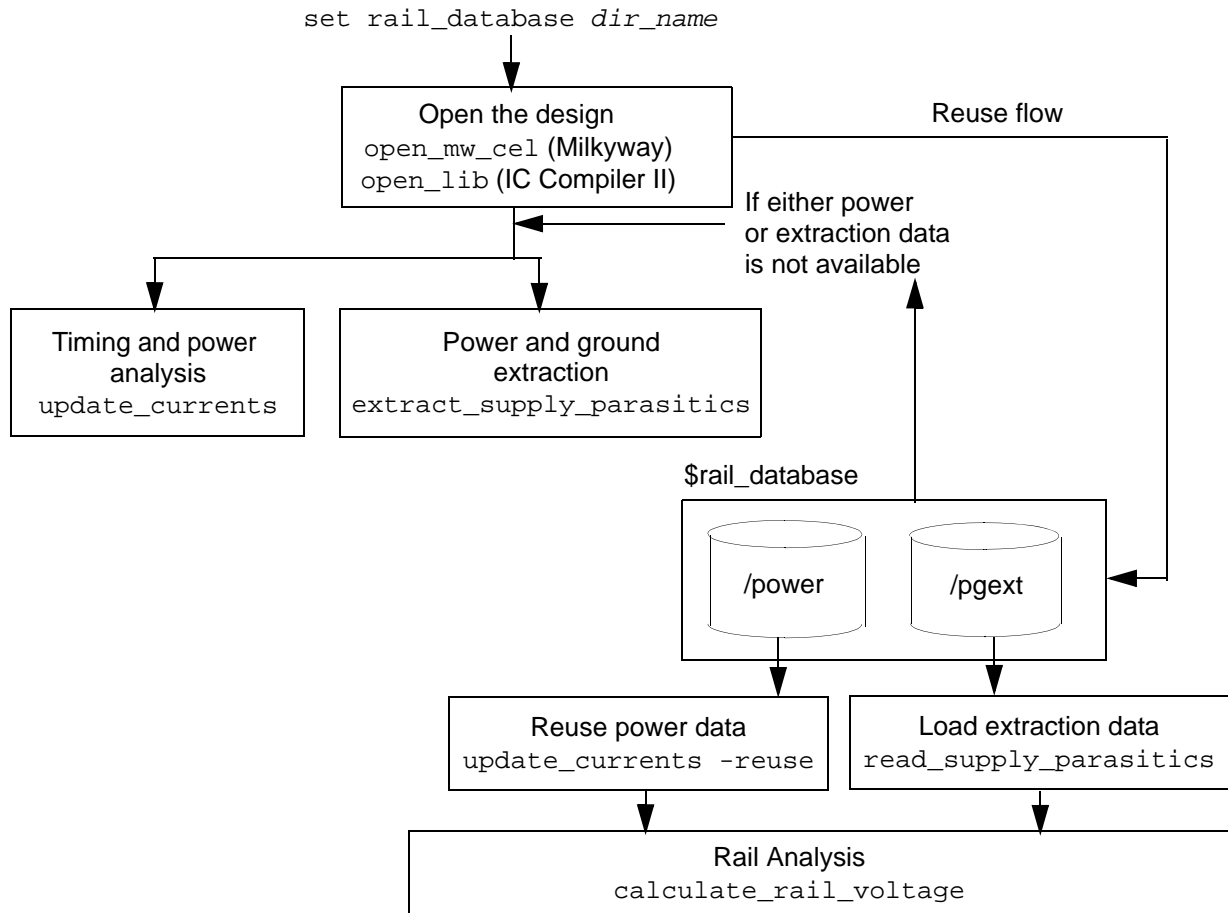
- [Extracting Supply Net Parasitics](#)

Reusing Power and Extraction Data

PrimeRail allows you to reuse the previously generated power and extraction data. This is useful if you have performed timing analysis and extraction on the design and want to reuse the data without rerunning the steps. Reusing power and extraction data saves the overall turnaround time.

[Figure 7-8](#) illustrates the data flow for the reusing previously generated power and extraction data.

Figure 7-8 Reusing Power and Extraction Data Flow



For example, to use the previously generated extraction data,

1. Run the `open_mw_cel` command to open the design.
2. Run the `read_supply_parasitics net_name` command to read in the existing extraction data.

PrimeRail reads the supply net extraction data that is saved in the `./RAIL_DATABASE/pgext` directory. Make sure that the `rail_database` variable is pointing to the `RAIL_DATABASE` directory from which you want to reuse the extraction data.

See Also

- [Setting Extraction Options](#)
- [Extracting Supply Net Parasitics](#)

8

Analyzing Power and Updating the Currents

PrimeRail provides the `update_currents` command, which allows you to perform timing and power analysis and generate instance-specific current waveforms. Some necessary design data, such as timing constraints, net switching activity and signal parasitics, are required for accurate power consumption values and current waveforms.

This chapter contains the following sections:

- [Introduction to Power and Current Waveform Generation](#)
- [Required Input Data and Data Flow for Current Calculation](#)
- [Analyzing Power and Generating Current Waveforms](#)
- [Debugging Power and Current Information](#)
- [Displaying Power and Current Maps](#)
- [Saving and Reusing Power Data](#)

Introduction to Power and Current Waveform Generation

Before analyzing voltage drop and rise in the power supply network, PrimeRail requires accurate power dissipation data of the designs. The power data is then used to calculate the current value for each power or ground pin that is connected to the nets being analyzed.

Run the `update_currents` command to perform gate-level power analysis and current waveform generation. The gate-level power analysis calculates timing and power information for the nets. With the timing and power reports generated by gate-level power analysis, the `update_currents` command then generates current waveforms for the design and loads additional parasitics of the power supply network for all the power and ground ports of each cell in the design.

In PrimeRail, you can run gate-level power analysis in either static, dynamic, or dynamic vectorfree mode, depending on the purpose of the rail analysis or the availability of the net switching information. With the generated power reports, you can choose to generate time-averaged or time-varying waveforms for static or dynamic rail analysis respectively.

PrimeRail uses the CCS power models to generate power and current waveforms. For macro cells, if no CCS power model is available, PrimeRail looks for the energy-scaled piecewise linear delay model (PWL). For standard cells, PrimeRail uses the NLPM models and creates trapezoid waveforms instead.

You need to specify additional settings in the CCS power library for generating current waveforms for macro cells. For more information, see [Settings for Generating Macro Cell Waveforms](#).

PrimeRail relies on the macro modeling technology or the CCS power library to generate PG pin current waveforms for macro cells. If none of these is available, PrimeRail uses timing data instead.

See Also

- [Required Input Data and Data Flow for Current Calculation](#)
- [Analyzing Power and Generating Current Waveforms](#)
- [Debugging Power and Current Information](#)
- [Displaying Power and Current Maps](#)
- [Saving and Reusing Power Data](#)

Required Input Data and Data Flow for Current Calculation

To accurately calculate power and timing values, PrimeRail requires some design and library data to be present in the database before you can execute the `update_currents` command. If any of the required files, such as SDC files, are missing, PrimeRail stops executing the command.

Since a large component of the power dissipated by a design depends on the circuit activity, PrimeRail uses the stimulus information that represents the activity for the event-based power analysis process. In PrimeRail, the stimulus is read through three different sources: user-provided simulation events, such as those recorded in an event file; user-provided statistical probabilities, such as those provided through a SAIF file; and tool-calculated statistical probabilities.

You do not need to provide the net switching activity information for the vectorfree power analysis flow.

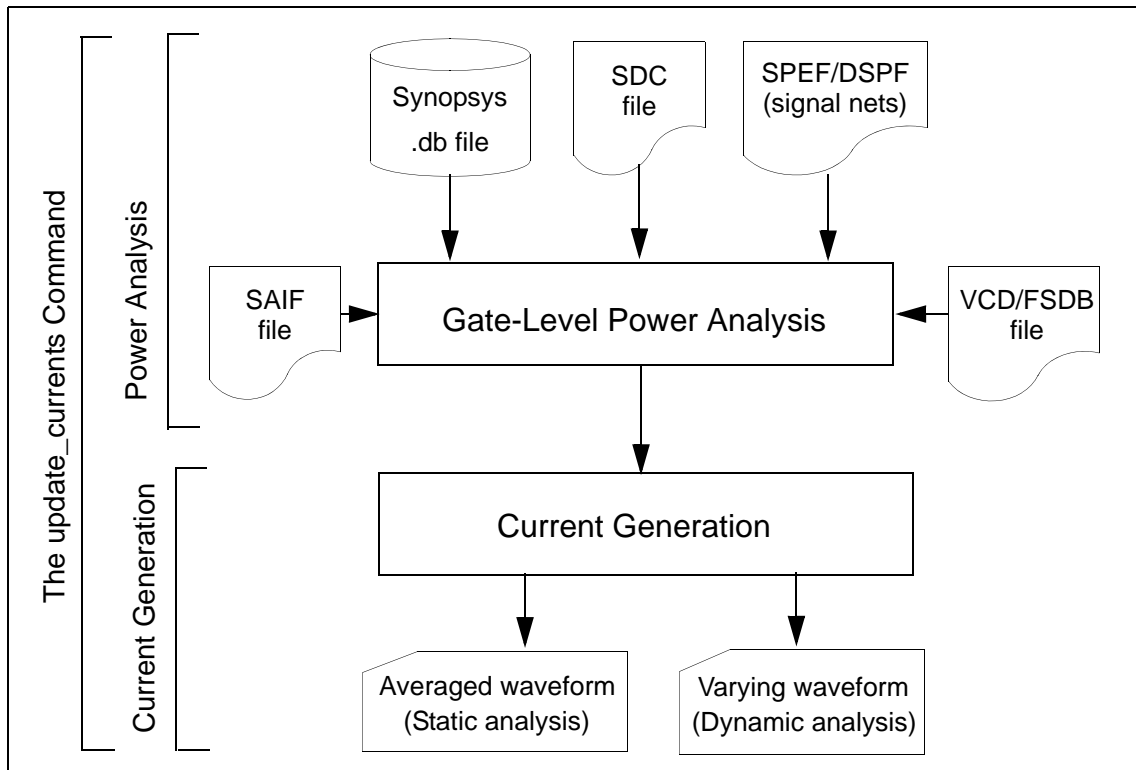
The following list contains the commands for reading in the required data and for running power calculation and current generation:

- `read_sdc`
- `read_parasitics`
- `set_voltage`
- `read_vcd` or `read_fsdb` for event-based power analysis
- `read_saif` or `read_vcd` for static power analysis
- `update_currents -static|-dynamic|-dynamic_vectorfree`

Depending upon the source of the stimulus, PrimeRail generates a VCD or FSDB binary report file (for event-based stimulus) or a SAIF binary report file (for statistical stimulus) upon the end of the power analysis. For event-based power analysis, both VCD (or FSDB) and SAIF binary report files are created, with the SAIF binary report file being driven by an implicit conversion of the events into switching probabilities.

[Figure 8-1](#) shows the data flow for the power and current generation in PrimeRail.

Figure 8-1 Data Flow for Calculating Power and Current With the `update_currents` Command



For more information about what data is required for running the `update_currents` command, see [Preparing and Checking Design Data](#).

See Also

- [Analyzing Power and Generating Current Waveforms](#)

Analyzing Power and Generating Current Waveforms

Use the `update_currents` command (or choose Power > Update Currents in the GUI) to calculate power consumption and current waveforms for the design. PrimeRail uses the generated power or ground pin electrical currents for rail analysis.

When running the `update_currents` command, PrimeRail first performs gate-level power analysis to analyze timing and power values of the design. With the generated timing and power information, the tool then calculates current waveforms of the design.

PrimeRail provides the following methods for power and current calculation:

- [Static Power and Current Generation](#)
- [Dynamic Event-Based Power and Current Calculation](#)
- [Dynamic Vectorfree Power and Current Calculation](#)
- [Power Analysis With Assigned Power](#)
- [Power Scaling](#)
- [Waveform Calculation With Assigned Current](#)

Static Power and Current Generation

Use the `update_currents -static` command to perform power analysis and current waveform generation in the averaged mode. The `update_currents` command first performs power analysis using the provided switching activity data, voltage settings, and other constraints. When power analysis is finished, PrimeRail generates instance-specific current waveforms using the calculated power data. When you use the `-static` option, the generated power and ground pin current for static analysis is a scalar value using the following equation:

$$I = \frac{P}{V}$$

where I stands for the current, P is the average power consumed by that pin, and V is the ideal voltage of the pin.

The tool generates current waveforms using the signal probabilities and timing windows that are created during power analysis. The timing windows are for each switching event of all the cells for a full clock period. Rising and falling switching events are considered separately.

The current waveform generated for a switching event also depends on the state of the inputs of the cell. In the static analysis, PrimeRail accounts for this dependency by generating a composite current waveform. This current waveform is a superposition of current waveforms for all input states of the cell and are scaled by the probability of a state according to the signal probabilities of the input states of the cell.

Annotation of Default Toggle Rates and Static Probability

By default, when calculating averaged power and current waveforms with the `update_currents -static` command, the tool propagates activities through design logics to assign activities on the unannotated design objects.

Use the `power_sequential_default_toggle_rate` and `power_sequential_default_static_probability` variables to annotate default `toggle_rate` and static probability values on all unannotated synthesis invariant points. When enabled, the engine uses the minimal effort level to propagate activities with less iterations.

Enable the `power_sequential_default_toggle_rate` and `power_sequential_default_static_probability` variables to reduce runtime when you calculate static power and current waveforms on designs with high annotation on synthesis invariant points.

See Also

- [PrimeRail Working Directories](#)
- [Reading Switching Activity Information](#)
- [Debugging Power and Current Information](#)
- [Saving and Reusing Power Data](#)

Dynamic Event-Based Power and Current Calculation

Use the `update_currents -dynamic` command to generate power and pin current waveforms for dynamic rail analysis with the switching activity information. The generated power and ground pin currents are represented as time-varying waveforms over the analysis window, and are derived from the instantaneous power consumed at each power and ground pin.

The VCD-based power analysis requires an event file that contains detailed switching event information. With the detailed switching events, the timing analysis results and the Liberty power models, PrimeRail derives the detailed switching, short-circuit and leakage power consumption values for every cell instance and for each event in the given simulation vector. Therefore, running a vector-based analysis generates the most accurate dynamic current waveforms for the design. The event-based analysis is also called event-based analysis.

By default, the generated currents are with respect to the full time span of the power analysis, either defined by the `-time` option of the `read_vcd` or `read_fsdb` command or the time window captured in the specified power file, such as a previously generated power file. To set a narrower set of time windows, use the `-time` option of the `update_currents` command. Narrowing the time range saves significant processing time if you are running the power analysis over a large time span.

See Also

- [PrimeRail Working Directories](#)

- [Reading Switching Activity Information](#)
- [Debugging Power and Current Information](#)
- [Saving and Reusing Power Data](#)

Dynamic Vectorfree Power and Current Calculation

Use the `update_currents -dynamic_vectorfree` command to generate power and pin current waveforms for dynamic rail analysis without detailed switching activity information. Unlike the dynamic vector-based power analysis flow where the maximum power is calculated based on the event information, PrimeRail derives the maximum power and its related circuit signal transition information by analyzing the circuit structure and design constraints during dynamic vectorfree power analysis flow.

See Also

- [PrimeRail Working Directories](#)
- [Debugging Power and Current Information](#)
- [Saving and Reusing Power Data](#)

Power Analysis With Assigned Power

By default, when you run the `update_currents` command, the tool calculates power consumption of the design and distributes the calculated power evenly to the corresponding current source file (CSF) locations. If you prefer using a power value other than the one generated by the `update_currents` command, or if no correct total power is available, like a black box without a detailed power model, use the `set_rail_power` command to manually define the power values for such cell instances. The power unit is watts.

Note:

You have to assign power before running the `update_currents` command. Otherwise, PrimeRail does not consider the assigned power values during current calculation and rail analysis.

The `set_rail_power` command assigns power values to power or ground pins. You can either directly specify the power or ground pins or indirectly specify them by referencing their associated cell instances, library cell references, or connected supply nets. The directly assigned power values have a higher precedence than the indirectly assigned values. A warning message is issued if you try to assign power to a power or ground pin indirectly, but another source of power information with higher power precedence was already set on the same power or ground pin.

To write out or remove the assigned power, use the `report_rail_power` or `remove_rail_power` commands, respectively.

The following list describes the ways of specifying power values that are supported in PrimeRail in an order of precedence from high to low.

- Power values that are directly assigned by the following command:

```
set_rail_power pg_pin_list
```

- Power values that are indirectly assigned by either of the following commands:

```
set_rail_power cell_list
set_rail_power lib_cell_list
set_rail_power -supply_net
set_rail_power -import
```

When multiple commands exist in the script, PrimeRail honors the latest one used.

- Power values generated by the `update_currents` command

In the assign-only flow, you can assign power in either of the following ways:

- [Assigning Power to PG Pins](#)
- [Assigning Power to Signal PG Nets or Pins](#)
- [Assigning Internal and Leakage Power](#)
- [Importing a Power File](#)

Assigning Power to PG Pins

Use the `set_rail_power` command to assign total power to the power or ground pins of the cells to be used in the rail analysis. The total power includes leakage power, internal power, and switching data. Values assigned by this command override all other sources of total power.

The `set_rail_power` command assigns power values to the specified power and ground pins. If no power or ground pin is specified, the command divides the power equally across all the power and ground pins of the specified cells or library cells. When power is applied to library cell power and ground pins, all instances of the library cell have the same power applied.

Examples

The following example shows how to set 12 uW power per rail for each SRAM32x64 instance.

```
pr_shell> set_rail_power .000012 [get_lib_cell SRAM32x64]
pr_shell> report_rail_power
```

Cells	PG_Pin_Name	Calculated_power	Assigned_Power
ram0_1_x0	VDD	8.38091e-06	1.2e-05
ram0_1_x0	VSS	8.38091e-06	1.2e-05
ram0_1_x1	VDD	1.17441e-05	1.2e-05
ram0_1_x1	VSS	1.17441e-05	1.2e-05

The following example shows how to set the power for all the power and ground pins that connect to supply net VDD:

```
pr_shell> set_rail_power 0.25 -supply_net VDD [get_cells -hier]
```

Assigning Power to Signal PG Nets or Pins

In PrimeRail, assigning power to signal nets or pins is not supported. You need to use the `change_pin_type` and `change_net_type` commands to change leaf cells' pin and net types from signal to either power or ground. Then run the `set_rail_power` command to assign power to the converted signal pins and nets. You need to run the `change_pin_type` and `change_net_type` commands before the `open_mw_cel` command.

The following example changes the pin type of the signal pin `test_a` to ground:

```
pr_shell> change_pin_type test_master -pin test_a -to ground
```

Note that the cell master of the pin must exist in a .db file and cannot be a macro cell. After conversion, the signal pin is not removed from the design and remains listed as a signal pin when you retrieve pin information using the `get_pins` command.

The following example changes the net type of the signal net `fi03_D` to power.

```
pr_shell> change_net_type -from signal -to power 03_D
pr_shell> open_mw_cel -library design dut19
pr_shell> get_supply_nets
```

All the pins connected to this net will be treated as PG pins, not signal pins. For a detailed description about how to change the pin type for signal pins, see the command man page.

Assigning Internal and Leakage Power

Run the `set_annotated_power` command to set internal power and leakage power for cell instances without library power models where accurate power values are not available. You must execute the `set_annotated_power` command before you perform power and current waveform generation with the `update_currents` command.

The power values assigned with the `set_annotated_power` command have a higher precedence than the power values calculated by the `update_currents` command.

Importing a Power File

If you have described the pin and power information in an ASCII file and want to use that information for rail analysis, run the `set_rail_power -import fileName` command. The *fileName* argument specifies the name of the ASCII file that describes the pin and power information.

In the ASCII file to be imported, describe the information for cells, pins, or library cells in the following format:

```
CELL cell_instance_name power_value [pattern_match]
PG_PIN cell_instance_name pg_pin_name power_value [pattern_match]
LIB_CELL lib_cell_name power_value [pattern_match]
```

Wildcard usage (`.*`, `?`, and `[]`) is supported. The power values are in watts.

Note:

If you import power information with the `set_rail_power -import` command, you can proceed to the rail analysis step without running the `update_currents` command. PrimeRail calculates the instance PG port current from the assigned power.

Example

The following example assigns power to all the power and ground pins specified in the `power_file.txt` file:

```
pr_shell> set_rail_power -import power_file.txt
```

Power Scaling

For what-if purposes, you can perform rail analysis with a scaling factor or value to estimate how much total power can be reduced without changing the actual power consumption result. Doing this temporarily affects the power that is used to calculate the voltage drop at power and ground pins when you run the `calculate_rail_voltage` command. The actual saved power remains unchanged.

When power and current waveforms are generated using the `update_currents` command, run the `scale_rail_power` command to scale power by assigning a power value or a scaling factor. You can perform scaling on all cell instances or those that are not in the clock network.

By default, the `scale_rail_power` command scales leakage power during power scaling. To exclude leakage power from being scaled, use the `-exclude_leakage` option.

The `scale_rail_power` command does not scale the instance power that is set by the `set_rail_power` command. To scale an assigned instance power, use the `-include_annotated_power` option instead.

The syntax is

```
scale_rail_power
[-target_power power_value | -scale_factor scale_value]
[-type non_clock_network | all ]
[-exclude_leakage]
[-include_annotated_power]
cell_list
```

When scaling is finished, the `has_scaled_power` attribute for the cells and the PG pins is set to `true`, and a scaling factor is stored for each effected instance. The same scaling factor is also used to scale current waveform during dynamic rail analysis.

Use the `total_power` attribute to list the scaled power value.

In PrimeRail, there are different ways for specifying power values. The following list describes these power values in a decreasing order of precedence.

- Power values that are directly assigned by the following commands:

```
set_rail_power pg_pin_list
set_rail_power cell_list
set_rail_power lib_cell_list
set_rail_power -supply_net
set_rail_power -import
```

When multiple commands exist in the script, PrimeRail honors the latest one used.

- Power values that are scaled by the `scale_rail_power` command
- Power values that are generated by the `update_currents` command

Example

In Design A, the power values for the `ci_d2` and `ci_clk` instances are 2.0 and 3.0 respectively. The `ci_d1` instance is assigned with power 1.0 with the `set_rail_power` command. To perform power scaling with a value 11.0 on all the instances, use the following command:

```
pr_shell> scale_rail_power -target_power 11.0 -type all
```

The power values of the `ci_d2` and `ci_clk` instances are scaled to 4.0 and 6.0 respectively. The power value of the `ci_d1` instance remains unchanged because the `scale_rail_power` command does not scale the power value assigned by the `set_rail_power` command.

Here is the summary of the scaling result:

```
ci_d1 = 1.0
ci_d2 = 4.0
ci_clk = 6.0
```

Waveform Calculation With Assigned Current

Typically, you calculate the power or ground pin current with the `update_currents` command. However, the `update_currents` command requires power models to be available for generating currents. If you do not have a power model for the cell or want to assign a current value directly, use the `set_rail_current` command. The assigned current unit is the same as the one specified in the library.

Use the `set_rail_current` command to assign electrical current to the power and ground pins of cells for rail analysis. You can assign current as an aggregated vector-free static current value with the `-static` option. The command overwrites any current that was previously generated by the `update_currents` command or that you assigned earlier.

When you assign static current to a power or ground pin, the corresponding power value for this power or ground pin is also assigned. This is to make the power value consistent with static current for the given power or ground pin. To query the assigned power value, use the `report_rail_power` command.

To determine the electrical currents of the power or ground pin, query the `static_current` or `current_waveform` attribute on that power or ground pin.

To remove the assigned current, use the `remove_rail_current` command.

Note:

If you set both the `set_rail_current` and `set_rail_power` commands in the script, PrimeRail honors the latest one shown.

You can assign user-defined current to a pin either before or after the `update_currents` command. The assigned current always takes precedence over the one calculated by the `update_currents` command.

Example

The following example applies a current of -5 mA to a power or ground pin.

```
pr_shell> report_units
...
Current_unit      : 0.001 Amp
pr_shell> set_rail_current -static -5 [get_pg_pins MyCell/VDDL]
```

Each successive pair of values in the waveform list represents time and the corresponding current value: {t1 i1 t2 i2 t3 i3 ... }.

Log Example

Here is an example of the power analysis result.

```
Warning: Neither event file or switching activity data present for power
estimation. The command will propagate switching activity values for
power calculation. (PWR-246)
```



```

Information: Running switching activity propagation with 4 threads!
Information: Running averaged power analysis... (PWR-601)
Information: dump power archive in background.
Information: 430.031 MB PG extraction memory is counted.
Information: Created saif binary report file '/power/
pr_rail_power.saif_rpt `

Total cell power :
total switching power = 45.2157 mW (52.4619 %)
total internal power = 12.9844 mW (15.0652 %)
total leakage power = 27.9876 mW (32.4728 %)
total power = 86.1876 mW
Total assigned rail power = 0.25176 mW
Total calculated power overwritten by assigned rail power = 0.234564 mW
Total final power = 86.2056 mW

```

See Also

- [Required Input Data and Data Flow for Current Calculation](#)
- [Debugging Power and Current Information](#)
- [Displaying Power and Current Maps](#)
- [Saving and Reusing Power Data](#)

Debugging Power and Current Information

To verify the power data calculated by the `update_currents` command, 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.

Use the `report_current_calculation` command to generate a report that provides detailed current and intrinsic RC calculation information for the power or ground pins of all the cells specified by the cell list. The report also details how PrimeRail derives the current information from power analysis and library data.

Note:

The `report_power_calculation` is available in the static and dynamic vectorfree flow.

The `report_current_calculation` command is available only in dynamic vector-based analysis flows.

For standard cells, the report includes the following information:

- A table reporting the units used throughout the report. The library units are used wherever applicable.
- A detailed report per specified cell, including

- Information for each signal pin of the cell, including
 - Rise transition time (slew)
 - Fall transition time (slew)
 - Probability of a '1' state being on the pin
 - Switching activity (toggles per time unit)
- Information for each power pin of the cell, including
 - Calculated power (P)
 - Ideal voltage (V)
 - Derived static current (P/V)
- Information for each ground pin of the cell, including
 - Contribution to the ground pin current from each power pin, including:
 - Percent contribution of the ground pin power and current from the power pin
 - Power contributed by the power pin
 - Current contributed by the power pin
 - Total power for the ground pin (as summed from the power pin contributions)
 - Total static current (as summed from the power pin contributions)
 - Ideal voltage (V)

For macro cells, the information reported is the same as for standard cells, except the detailed reports for the power and ground pins. PrimeRail calculates the currents for macro power or ground pins based upon the operating mode information provided with the macro model. This report includes the information for each power or ground pin of the cell, such as

- Information about each defined operating mode (by mode name):
 - Triggering signal pin
 - Defined rise or fall direction for the trigger
 - “When” condition
 - Switching activity for the trigger pin and its direction (toggles per time unit)
 - Probability of the “when” condition being met
 - Charge consumed per trigger
 - Derived static current at the power pin
- Calculated Power (P)

- Ideal voltage (V)
- Derived static current (summed from the operating mode information)

See Also

- [Analyzing Power and Generating Current Waveforms](#)

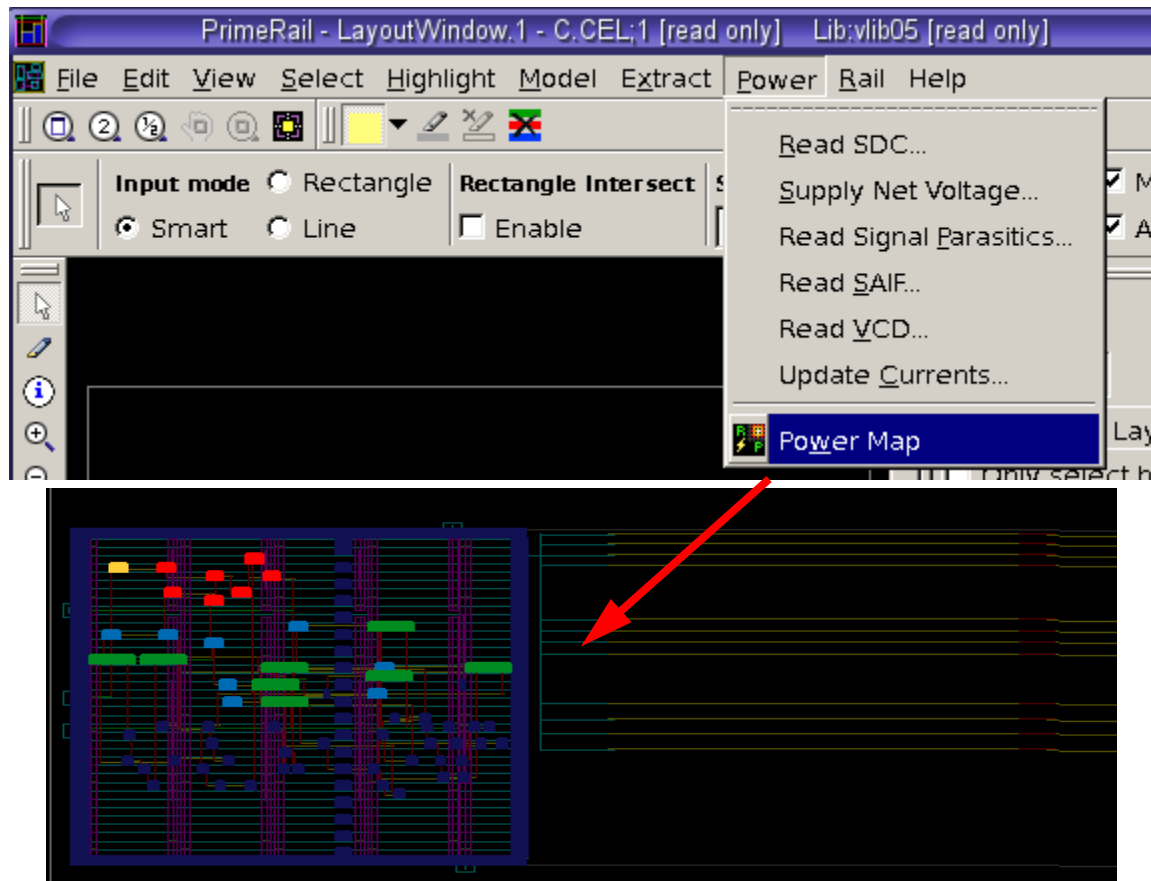
Displaying Power and Current Maps

When the `update_currents` command run is complete, you can display power and current distributions of your design in the power and current maps. Displaying the power map before you perform rail analysis helps you better understand where potential problem areas might occur.

If you select the Auto-load Power Maps option in the Update Currents dialog box, PrimeRail automatically displays the generated power map when power analysis is complete. Alternatively, you can choose Power > Power Map in the GUI to display the power map.

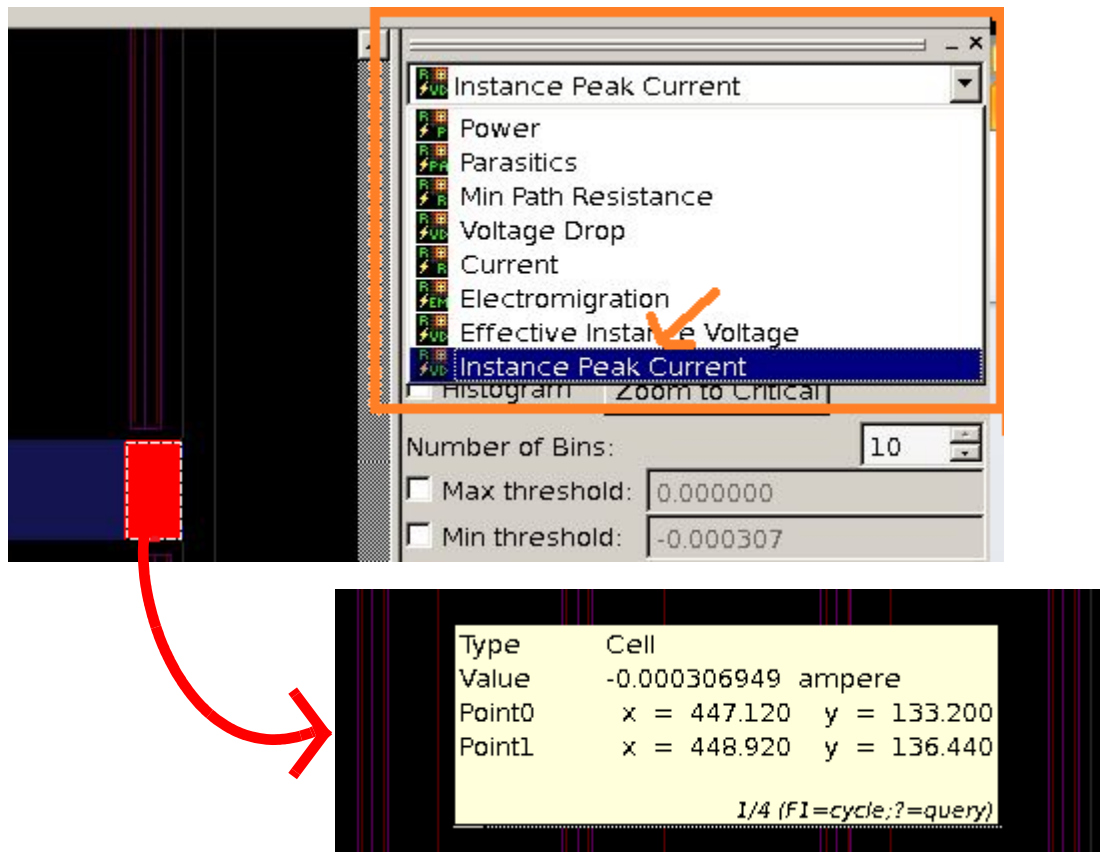
[Figure 8-2](#) shows an example of the power map, which you can use to check for the power consumption values of hotspots.

Figure 8-2 Displaying a Power Map to Investigate a Hotspot



When dynamic rail analysis is complete, you can examine peak current values for different instances or areas in the current map. Choose Instance Peak Current from the map selection pull-down menu and then click an instance on the map. The peak current values for the selected instance are reported in the yellow text box, as shown in [Figure 8-3](#).

Figure 8-3 Displaying Instance-Based Peak Current Maps

**See Also**

- [Analyzing Power and Generating Current Waveforms](#)

Saving and Reusing Power Data

By default, the `update_currents` command saves power data in the `$rail_database/power` directory. To specify a different location for saving the power data, set the `rail_database` variable.

To reuse the power data in another session, use the `-reuse` option with the `update_currents` command. Reusing power data allows you to run rail analysis across multiple sessions without rerunning power analysis, therefore reducing the overall turnaround time. See [Figure 7-8](#) for a data flow that illustrates how to reuse the previously generated power data.

For example, use the following script example to reuse the power data that was generated in the previous static power analysis runs, and was saved in the `./my_rail_database/power` directory:

In the static mode:

```
set rail_database ./my_rail_database
update_currents -static -reuse
```

See Also

- [Analyzing Power and Generating Current Waveforms](#)

9

Static Rail Analysis

When power analysis and supply net extraction have been completed for all cells in the design, you can perform static voltage drop and electromigration analyses on the design. You can also perform minimum path resistance analysis to find power and ground grid weaknesses.

For a detailed description about supply network extraction and power analysis, see [Extracting Supply Net Parasitics](#) and [Analyzing Power and Updating the Currents](#).

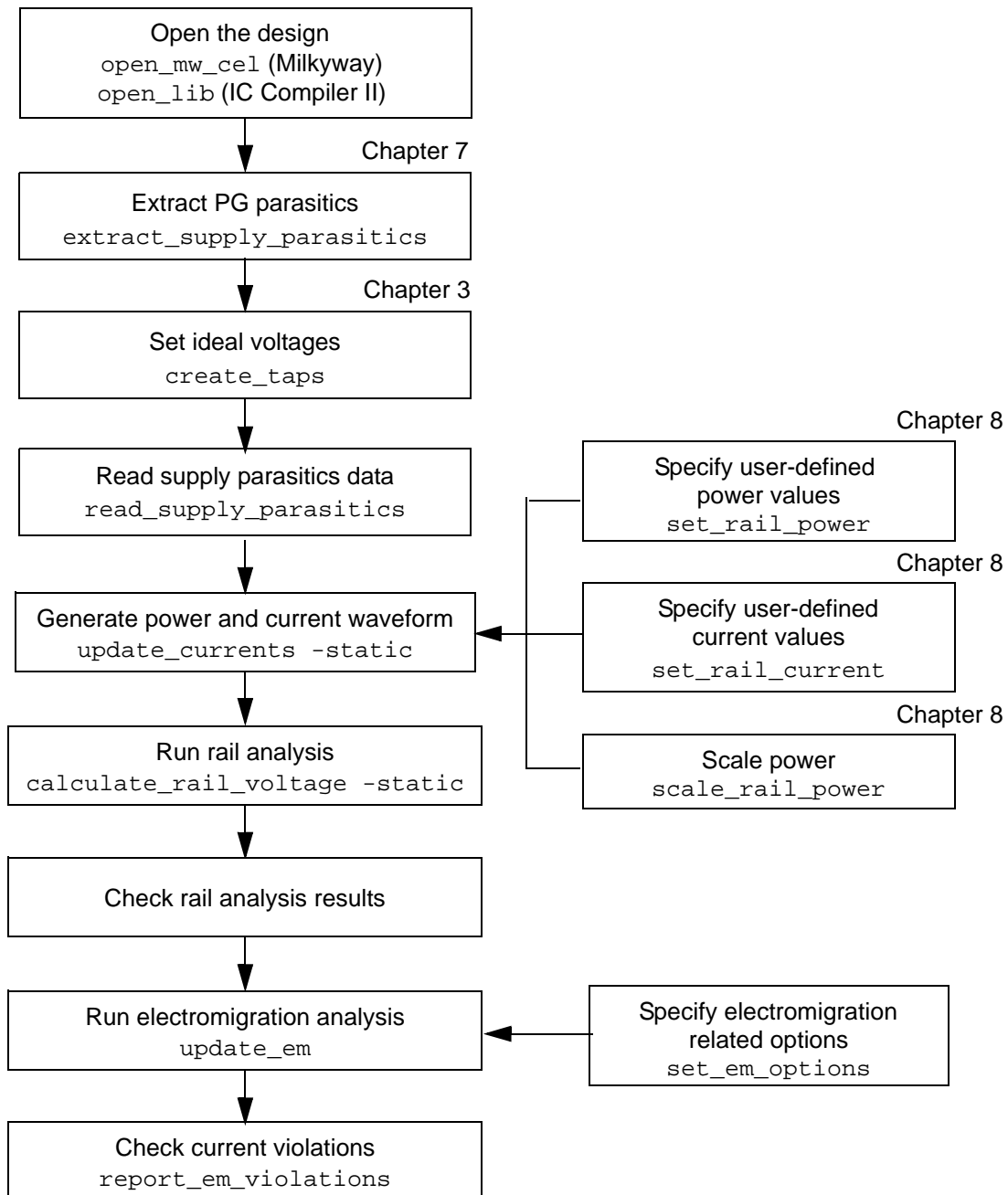
This chapter contains the following sections:

- [Static Rail Analysis Flow](#)
- [Setting Ideal Voltage Sources](#)
- [Reading Supply Parasitics](#)
- [Performing Static Voltage Drop Analysis](#)
- [Displaying Static Rail Analysis Results](#)
- [Performing Static Electromigration Analysis](#)
- [Displaying Static Electromigration Analysis Results](#)
- [Considering Body-Bias Effects](#)

Static Rail Analysis Flow

Figure 9-1 illustrates the steps in the static rail analysis flow.

Figure 9-1 Required Steps for the PrimeRail Static Analysis Flow



See Also

- [Performing Static Voltage Drop Analysis](#)

Setting Ideal Voltage Sources

During rail analysis, PrimeRail combines the power consumption results and the resistive network results to solve for the voltage drop values at each node in the resistive network. PrimeRail needs the location of the ideal voltage sources (taps) to form the boundary condition before solving for voltage drops on each node in the design.

To specify ideal voltage sources, use the `create_taps` command to create tap point objects in the current session. The taps are to be considered as part of supply network in the subsequent rail analyses.

For more information about creating ideal voltage drop sources, see [Creating Taps for Ideal Voltage Sources](#).

See Also

- [Performing Static Voltage Drop Analysis](#)

Reading Supply Parasitics

PrimeRail saves the parasitic data files (per master and per net) in the local working directory in binary format when the extraction process is complete. You must load the extraction data into memory with the `read_supply_parasitics` command before proceeding to the rail analysis step.

For more information about using the `read_supply_parasitics` command, see the command man page.

See Also

- [Performing Static Voltage Drop Analysis](#)

Performing Static Voltage Drop Analysis

Run the `calculate_rail_voltage` command to perform voltage drop analysis on the specified power nets. During voltage drop analysis, PrimeRail calculates voltage drop of the power and ground network and reports cell instances that have voltage drops.

For example,

```
pr_shell> calculate_rail_voltage [list VDD VSS] -static
```

Note:

You must complete the power analysis and supply net extraction, and then run the `read_supply_parasitics` command to load the data before you can perform rail analysis.

When the `calculate_rail_voltage` command run is complete, the tool automatically saves rail results in the cache. Each time you execute the command, the tool creates one result set with an associated name, either one that is automatically assigned by the command or the one that is specified with the `-result` option (see [Displaying Static Rail Analysis Results](#)).

The following table lists the most commonly used options of the `calculate_rail_voltage` command during static analysis flow. For a detailed description about each option, see the command man page.

Table 9-1 Commonly Used Options for the calculate_rail_voltage Command

Option	Description
<code>-noexpand</code>	Avoids expanding the listed supply nets to include all the supply nets that are connected by switching cells. By default, the <code>calculate_rail_voltage</code> command automatically expands the list of supply nets.
<code>-result rail_result_name</code>	Names the output rail result. When not specified, the command assigns a unique name to the result.
<code>-replace</code>	Replaces an existing rail result. When used with the <code>-result</code> option, the command names the output result as <code>rail_result_name</code> . When used without the <code>-result</code> option, the command names the output result based on what is specified by the <code>current_rail_result</code> command.
<code>-error_cell</code>	Assigns a name to the output error view. If the <code>-error_cell</code> option is not specified, no error view is generated.

Table 9-1 Commonly Used Options for the calculate_rail_voltage Command

Option	Description
<code>-voltage_threshold</code>	Specifies the voltage threshold value for the error view. If no voltage threshold is specified with the <code>-voltage_threshold</code> option, the default voltage value 0 (zero) is applied. The value is in volts and is an absolute number for both power and ground nets.
<code>-static</code>	Chooses the rail analysis mode to be static.

Example

The following example creates tap points from top-level power and ground pins, reads in the supply net parasitics, calculates power and current values, and performs voltage drop analysis. Run the `report_rail_result` command to generate a verbose report on supply nets and a snapshots of taps in the result.

The following is an example of the log file created when rail analysis is complete. The log file includes a detailed voltage drop report for all the supply nets being analyzed. For each net, the log reports the maximum and minimum voltage drop values, and five instances that have the highest voltage drop values. This information is useful to detect and analyze the large voltage drop areas. To understand the complete network's voltage drop, display voltage drop maps and use the query tools in the GUI.

```
pr_shell> create_taps -top_pg
pr_shell> read_supply_parasitics VDD
pr_shell> update_currents
pr_shell> calculate_rail_voltage VDD

*****
Power domain VSS network voltage report

## Maximum and minimum (averaged) voltage drop for the power and ##
## ground net##

AVG voltage:
  Min voltage rise (mV) = 0
  Max voltage rise (mV) = 80.48

Power domain VSS instance voltage report

## List instance PG pins with top 5 voltage drop values. Use this list ##
## to locate the instances with large voltage drops ##

Top 5 AVG voltage drop values at instance ports:
1: 80.48 (mV) at UPF/VSS (X8_LVT)
2: 80.475 (mV) at DATA_1_UPF/VSS (X1)
```

```
3: 80.474 (mV) at DATA_28_UPF/VSS (X1_HVT)
4: 80.466 (mV) at clock25/VSS (X2_LVT)
5: 80.465 (mV) at DATA_UPF/VSS (X1_HTA)
```

```
pr_shell> report_rail_results -verbose
*****
RAIL_RESULT has rail result
  Supply nets in the rail result: VDD
  Taps in the result: TAP_1 T
```

See Also

- [Extracting Supply Net Parasitics](#)
- [Analyzing Power and Generating Current Waveforms](#)
- [Displaying Static Rail Analysis Results](#)

Displaying Static Rail Analysis Results

When rail analysis is complete, PrimeRail saves the rail analysis results and the related logs on the disk in the RAIL_DATABASE directory. You can view areas with large voltage drops or resistances graphically by using voltage drop or minimum path resistance map.

In the map, the problematic areas are highlighted in different colors. Alternatively, you can generate error views for locations where limits are violated, or view the rail analysis results in the ASCII format.

With the rail analysis results, you can perform the following tasks:

- [Displaying Voltage Drop Map](#)
- [Writing Rail Data](#)
- [Debugging Rail Results](#)
- [Toggling Between Rail Results](#)

To save the voltage drop violations or resistance calculations to an error view, use the `-error_cell` option with the `calculate_rail_voltage` command. You can then open the generated error view in the error browser.

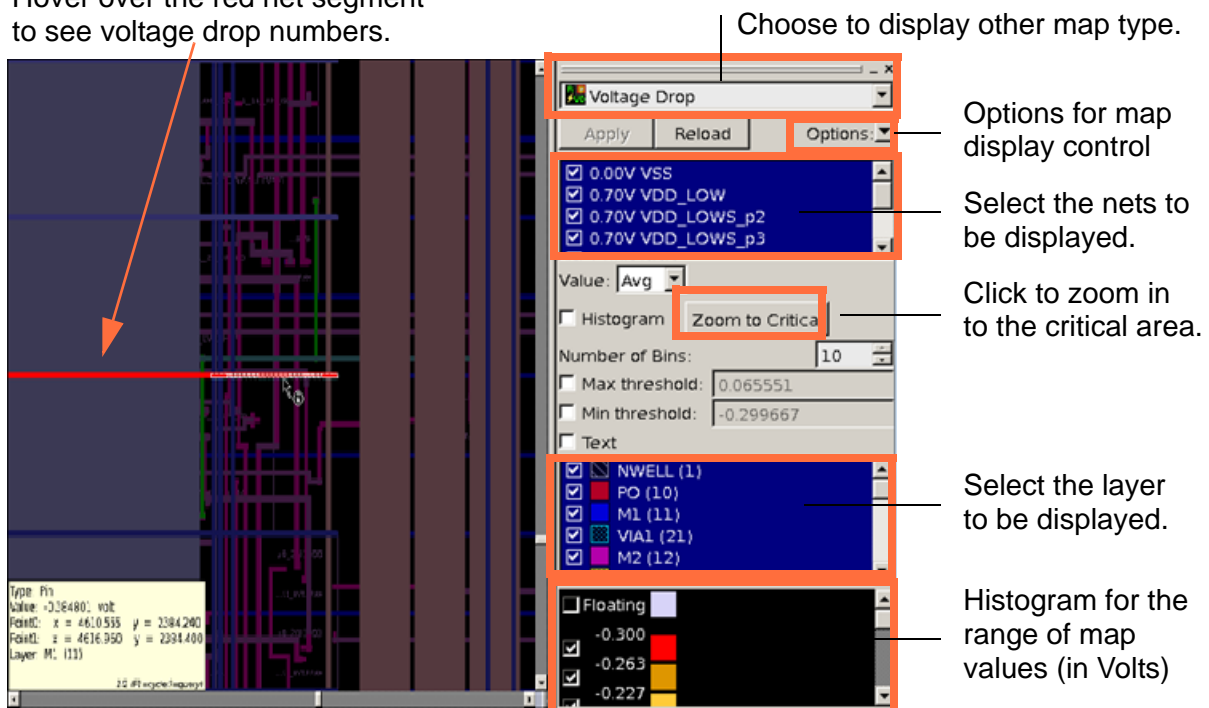
Displaying Voltage Drop Map

To display a voltage drop map,

1. Choose Rail > Voltage Drop Map from the GUI.
2. Click Apply. The voltage drop map is shown in the layout window, as shown in [Figure 9-2](#).

Figure 9-2 Displaying a Voltage Drop Map

Hover over the red net segment to see voltage drop numbers.



To examine the voltage of one specific net, click Options and then select Objects > Hide all. Then select the net to be displayed from the list. To analyze only the shapes on a specific layer, select the layer from the list.

Use the histogram to determine the range of values to be displayed. This is useful if you want to display only the most critical drop ranges in the map. To inspect the voltage drop of the critical area in the map, click Zoom to Critical. Then use the InfoTip tool to examine the value.

To compare the voltage drop map with the results from another rail analysis run, run the `current_rail_result result_name` command to specify a rail content and then redisplay the map. This is useful in a what-if scenario when you assign a power or current value with the `set_rail_power` or `set_rail_current` command and want to compare the voltage effect in a snapshot.

For a detailed description about minimum path resistance maps, see [Minimum Path Resistance Analysis](#).

See Also

- [Toggling Between Rail Results](#)
- [Writing Rail Data](#)

Writing Rail Data

When rail analysis is complete, run the `write_rail_data` command to write the rail analysis result from the current session to a text file. Use this output file to set the relevant rail data in another session with the `set_rail_power -import` command. The power unit in the report file is watt; the current unit is ampere; and the voltage unit is volt.

You can specify which data type to write by using the `-type` option. The supported data types are

- `instance_power`: The power value on each instance
- `effective_voltage_drop_or_rise`: The dynamic effective voltage violations
- `minimum_path_resistance`: The minimum path resistance on each instance
- `pg_pin_peak_current`: The peak current value on each power or ground pin
- `pg_pin_power`: The power value on each power or ground pin
- `voltage_drop_or_rise`: The dynamic voltage violations

To filter the data to be written, use the options listed in [Table 9-2](#).

Table 9-2 Options for Filtering Out Rail Data

Option	Description
<code>-threshold</code>	Writes only the rail data that is smaller than the specified threshold. Note: This option does not support the <code>minimum_path_resistance</code> data type.
<code>-limit</code>	Limits the elements to be written. The default is 0, meaning all the elements are reported in a descending order.
<code>-cell</code>	Specifies the names of the cell instances whose data is to be written. Note: This option does not support the <code>voltage_drop_or_rise</code> and <code>effective_voltage_drop_or_rise</code> data type.
<code>-supply_net</code>	Specifies the names of the supply nets whose data is to be written. Note: This option does not support the <code>instance_power</code> data type.

Example 1

The following example writes the cell instance power values to a file, called `set_power_file`, and uses the exported instance power values for timing and power analysis at a later run:

```
pr_shell> write_rail_data -type instance_power set_power_file
```

```
Information: total 188 of cells written to file "set_power_file" with
total power: 0.0001 (W)
```

```
pr_shell> set_rail_power -import set_power_file
```

```
Information: 188 of power values assigned to cells from the file
set_power_file
```

```
pr_shell> update_currents
```

Example 2

The following example writes the top five peak PG pin current values to an output file, called peak.rpt:

```
pr_shell> write_rail_data -type pg_pin_peak_current -limit 5 peak.rpt
pr_shell> sh cat peak.rpt
FI2/vss 4.81004e-06
FI4/vss 4.79684e-06
FI1/vss 4.79594e-06
```

Example 3

The following example writes the calculated minimum path resistances for all instances in the design to an output file called minres.rpt:

```
pr_shell> write_rail_data -type minimum_path_resistance minres.rpt
pr_shell> sh cat minres.rpt
FEED_14/vbp 518.668
V61/vdd 158.268
CR4_1/vbp 145.582
FEED/vdd 156.228
V11/vbp 518.669
...
```

Example 4

The following example writes the calculated minimum path resistances for the cell instance INV61 to an output file, called minres_inv.rpt. In the report, the command prints the minimum path resistance values only for nodes, but not edges. The minimum path resistance value for the tap is 0.

```
pr_shell> write_rail_data -type minimum_path_resistance -cell \
INV1 minres_inv.rpt
pr_shell> sh cat minres_inv.rpt
```

pin_name	instance_name	XY location	minimum path resistance
PG Pin : INV1/vss			
M1 (447.060, 135.990)		(447.180, 136.350)	158.268
M1 (441.420, 135.990)		(447.060, 136.350)	
M1 (441.300, 135.990)		(441.420, 136.350)	156.228
M1 (439.620, 135.990)		(441.300, 136.350)	
...			
M1 (409.260, 135.990)		(409.380, 136.350)	144.881
M1 (407.940, 135.990)		(409.260, 136.350)	
M1 (407.820, 135.990)		(407.940, 136.350)	144.37
...			
M1 (0.360, 135.990)		(407.820, 136.350)	
TAP M1 (0.000, 135.990)		(0.360, 136.350)	

Debugging Rail Results

Use the `report_rail_voltage` command to write peak voltage drops and rises at the PG pins when the `calculate_rail_voltage` command run is complete. By default, the `report_rail_voltage` command reports on all the supply nets based on the current rail result. To examine voltages that are calculated in another session, use the `current_rail_result` command to load another rail result and set it as current.

By default, the command reports five instances that have the highest voltage drop or rise values. To control the number of instances to be reported, use the `-limit num` option. To list all the instances, set the `-limit` option to zero.

PrimeRail names the output report file `rail_result.out`. By default, it writes the file in ASCII format. To write the report in the format of the `set_voltage` command, use the `-primetime` option.

During full-chip analysis with a macro model, PrimeRail reports voltage drops at the hard macro boundary points (that is, the external pins of the hard macro). To examine voltages at the internal pins (that is, the current source points), set the `report_macro_internal_pin` variable to `true` before running the `report_rail_voltage` command.

To query voltage drops and rises of the PG pins, use the `get_attribute` command.

Example

The following example reports the voltages of the VDD and VSS supply nets based on the current rail analysis result. The voltage values are reported in a descending order, from high to low.

```
pr_shell> report_rail_voltage -limit 5
```

Top 5 static voltage drop values for supply net VDD at instance ports:

```
1: 693.088e-3 (mV) at buf_2I2/VDD (buf7)
2: 676.534e-3 (mV) at buf_2I6/VDD (bufa)
3: 670.242e-3 (mV) at 211d1_W24/VDD (211d1)
4: 656.981e-3 (mV) at buf_2I11/VDD (buf7)
5: 648.552e-3 (mV) at instruction_reg_29_/VDD (dfr1)
```

Top 5 static voltage rise values for supply net VSS at instance ports:

```
1: 791.158e-3 (mV) at buf_2I2/VSS (buf7)
2: 791.025e-3 (mV) at 22d1_I1783/VSS (22d1)
3: 791.025e-3 (mV) at abc02d0_T/VSS (abc02d1)
4: 784.504e-3 (mV) at ao_221d1_I253/VSS (ao_221d1)
5: 780.852e-3 (mV) at oa_211d1_W252/VSS (oa_211d1)
```

The following example reports the internal pin voltages of the VSS net based on the current rail analysis result. It reports only the top three voltage drop values.

```
pr_shell> set report_macro_internal_pin TRUE
pr_shell> report_rail_voltage VSS -limit 3
```

```
Top 3 static voltage rise values for supply net VSS at instance ports:
1: 23.33e-3 (mV) at inst1/cel_inst_VSS/10221 (sram1056x12)
2: 23.33e-3 (mV) at inst1/cel_inst_VSS/10222 (sram1056x12)
3: 23.327e-3 (mV) at inst1/cel_inst_VSS/11271 (sram1056x12)
```

In the previous example, the `_mm_dev_cel_inst` prefix (in bold) is added to each instance name, indicating that the voltage at an internal port is reported.

Toggleing Between Rail Results

Run the `current_rail_result` command (or choose Rail > Manage Rail Results in the GUI) to load and set the previously generated rail analysis results as the one to be used in the current session. This allows you to compare data from two or more result sets for what-if analysis. Use the `report_rail_results` command to see which results already exist in the current session for subsequent toggling. After assigning a result as current, use the `get_attribute` attribute to query the voltage values with the result context, which shows different values as expected.

Run the `current_rail_result -clear` command to clear the current rail analysis result without replacing it with another rail result. If you use the `-clear` option, the result is still available in the session and you can reload it back later with the `current_rail_result` command. Use the `remove_rail_result` command to completely remove the rail result from the current session.

Example

The following example switches between different rail results. In this example, there are two runs of the `calculate_rail_voltage` command under different setup conditions. Therefore, two named rail results are created for performing voltage drop analysis on the same net.

```
pr_shell> calculate_rail_voltage -static -result MyResult_1 VDD
pr_shell> create_taps -of_objects [get_cell MyTop/U235]
pr_shell> calculate_rail_voltage -static -result MyResult_2 VDD
pr_shell> current_rail_result
MyResult_2
pr_shell> get_attribute [get_pg_pins U1/VDD] static_voltage -2.14354e-04
pr_shell> current_rail_result MyResult_1
MyResult_1
pr_shell> get_attribute [get_pg_pins U1/VDD] static_voltage -1.41374e-05
```

The following example removes the current rail result without replacing it with a new rail result:

```
pr_shell> current_rail_result myrailresult
myrailresult
pr_shell> get_attribute [get_pg_pins U123/VDD] static_voltage \
-3.19933e-05
pr_shell> current_rail_result -clear
none
pr_shell> current_rail_result
none
pr_shell> get_attribute [get_pg_pins U123/VDD] static_voltage
UNINIT
```

Performing Static Electromigration Analysis

Electromigration (EM) is the gradual displacement of metal atoms in a metal conductor. It occurs when the current through the conductor is high enough to cause the displacement of metal atoms. During electromigration analysis, PrimeRail checks for nets whose currents exceed the limits defined in the electromigration rules. The rules include width- and temperature-dependent current limits.

Important:

You must complete power analysis, supply net extraction, and voltage drop analysis before you can perform electromigration analysis.

For a detailed description about power analysis, supply network extraction, and voltage drop analysis, see [Analyzing Power and Updating the Currents](#), [Extracting Supply Net Parasitics](#), and [Performing Static Voltage Drop Analysis](#), respectively.

See [Figure 9-1](#) for the steps that are required in the static analysis flow.

To calculate current violations when voltage drop analysis is complete, perform the following tasks:

1. Before analysis, make sure all the required input data, including electromigration rule information is ready.

See [Required Input Data for Electromigration Analysis](#) for details.

2. Run `set_em_options` command to specify configuration information for calculating current violations.

See [Setting Electromigration Options](#) for details.

3. Run `update_em` command to perform electromigration analysis.

See [Performing Electromigration Analysis](#) for details.

4. When electromigration analysis is completed, run the `report_em_violations` command to generate an ASCII violation report.

For more information about how to write a current report, display an electromigration map, or open an error view in the error browser, see [Displaying Static Electromigration Analysis Results](#).

Required Input Data for Electromigration Analysis

Other than the design and library data, PrimeRail also needs the information from the electromigration rule for calculating electromigration violations.

Use the `read_em_rules` command to read in an electromigration rule in the technology file, or ALF format. Run the `report_em_rules` command to list and examine the derived electromigration rule.

Open the `PR_CHECKING_DIR/em.missing_rules` file to check which layers do not have the electromigration rule defined. This file is generated when the `report_rail_checks` command run is complete.

The following is an example of the output `PR_CHECKING_DIR/em.missing_rules` file:

```
# This file contains the layers which are missing EM rules.
# Format : layer_name missing_em_rule_name
metall static
metal2 static
...
via1 static
Via2 static
...
```

See Also

- [Reading Electromigration Rules](#)
- [Reporting Electromigration Rules](#)
- [Checking Design Readiness](#)

Setting Electromigration Options

Run the `set_em_options` command to specify configuration information for electromigration analysis. You can run the `set_em_options` command multiple times to set the options incrementally. To check the settings of the current electromigration options, run the `report_em_options` command.

For what-if purposes, you can change a setting, such as setting a higher temperature or a different scaling factor, and rerun the `update_em` command to compare the different electromigration results.

The following table lists the options that are most commonly used. For a detailed description of the options, see the command man page.

Table 9-3 Commonly Used Options for the `set_em_options` Command

Option Name	Description
<code>-temperature temp_value</code>	Sets the temperature value, in degrees Celsius, for evaluating electromigration constraints. The allowed values are -300.0 through +500.0.
<code>-snap_temperature_to_rule snap_value</code>	<p>Snaps temp setting to the nearest value in the electromigration rule.</p> <p>Set <code>snap_value</code> to 1 to specify the nearest temperature value in the electromigration rule to be used when the user-specified <code>temperature_value</code> does not match the defined temperatures in the electromigration rule table. By default, <code>snap_value</code> is 0 and interpolation is performed to derive approximate rules for the specified <code>temperature_value</code>.</p>
<code>-scale_factor scale_value</code>	Specifies a uniform scale factor to be applied to the current threshold in the electromigration rule before electromigration violation checking. The default is 1.0.
<code>-width_limit width_value</code>	<p>Sets an absolute minimum width, in microns, for polygons to be included in the electromigration violation checking. If the width of a polygon carrying current that violates an electromigration rule is smaller than the <code>width_value</code>, the command does not report it as a violation.</p> <p>By default, there is no absolute minimum width for polygons to be included during electromigration violation checking.</p>
<code>-relative_width_limit value</code>	<p>Sets a minimum width, as a percentage of the layer minimum width defined in the technology file, for polygons to be included in the electromigration violation checking. If the width of a polygon carrying current that violates an electromigration rule is smaller than the defined percent of the minimum width for its layer, the command does not report it as a violation.</p> <p>By default, the <code>percentage_value</code> is set to 80 if the <code>relative_width_limit</code> option is not specified.</p>

Example

The following example sets configuration options for electromigration violation checking:

```
pr_shell> set_em_options -temperature 125 -snap_temperature_to_rule 1 \  
           -scale_factor 0.81  
pr_shell> report_em_options  
  
*****  
Report : Options for EM checking  
Design : test  
Version: <Version>  
Date   : <Date>  
*****  
Temperature: 125.00 C  
Snap Temperature to Rule: ON  
Scale Factor: 0.800  
Width Limit: NONE  
Relative Width Limit: 80.00 %  
1
```

See Also

- [Performing Static Electromigration Analysis](#)

Performing Electromigration Analysis

Run the `update_em` command to perform electromigration analysis based on the configuration options specified by the `set_em_options` command.

When electromigration analysis is completed, run the `report_em_violations` command to generate an ASCII violation report or to write the errors to an error view. You can also check for the violations graphically either in the electromigration map and the error browser.

Note:

If PrimeRail GUI is open when executing the `update_em` command, PrimeRail automatically displays the electromigration map when the analysis is complete.

See Also

- [Performing Static Electromigration Analysis](#)
- [Setting Electromigration Options](#)
- [Displaying Static Electromigration Analysis Results](#)

Displaying Static Electromigration Analysis Results

When electromigration analysis is complete, you can write the calculated electromigration violations into an ASCII report file or examine the violations that were found in an electromigration map. Alternatively, you can generate an error view during electromigration analysis and display the generated error in the error browser.

- [Viewing the Electromigration Violation Report](#)
- [Displaying Electromigration Maps](#)
- [Opening Error Data in the Error Browser](#)

Viewing the Electromigration Violation Report

Run the `report_em_violations` command (or choose Rail > EM Analysis in the GUI) to generate an ASCII log file that lists all the violations that are found during electromigration analysis.

The `report_em_violations` command reports electromigration violations for the nets that are included in the current rail result, using per-shape electrical currents calculated as a side effect of rail analysis. You must run the command after the `calculate_rail_voltage` command is executed.

To specify a temperature when evaluating temperature dependent electromigration constraints, use the `-temperature` option. By default, the command uses the temperature of the default operating condition in the first library in the link path.

To display the current and threshold ratio value in the violation report, either in a descending or an ascending order, enable the `pr_em_report_sort_by_ratio` variable before the `report_em_violations` command.

To keep electromigration checking data in memory for debugging purposes, use the `-keep_data` option. These checking data is used by the `report_em_threshold_calculation` command to write out the threshold-related data. See [Checking Threshold Information](#) for more information.

To remove the electromigration checking data from memory, use the `remove_em_data` command.

Here are the commonly used options to tailor the electromigration analysis results to identify the location where violations occur.

Option Name	Description
<code>-layer</code>	Specifies the layers to be included in the electromigration violation report.
<code>-scale_factor</code>	Specifies a scale factor to apply to the current threshold values in the EM rule when displaying EM violations
<code>-temperature</code>	Specifies a global temperature for EM analysis, in Celsius.
<code>-width_limit</code>	Specifies an absolute minimum width, in microns, for polygons to be included in the electromigration violation report. The default <code>width_value</code> is set by the <code>set_em_options</code> command.
<code>-relative_width_limit</code>	Specifies a minimum width, as a percentage of the layer minimum width defined in the technology file, for polygons to be included in the electromigration violation report. The default <code>percentage_value</code> is set by the <code>set_em_options</code> command.
<code>-advanced_em</code>	Invokes the advanced electromigration analysis using the advanced electromigration rules loaded with the <code>read_em_rules</code> command.
<code>-error_cell</code>	Creates an error cell to be viewed in the IC Compiler or IC Compiler II error browser.

Example

```
pr_shell> set pr_em_report_sort_by_ratio ascending
pr_shell> report_em_violations -scale_factor 1.000 -temperature 125
```


The following is an example of the output report file:

 Section: EM Violations

Scale Factor: 1.000

Temperature : 125 C

Units:

WIDTH : 1e-06 Meter

AREA : 1e-12 Meter**2

CURRENT : 1e-06 Amp

Keys:

L_NAME : Layer name

L_NUM : Layer number

THRESHOLD : EM threshold for current

RATIO : Ratio of current to threshold

W|A : Width or area (rule-dependent)

REC W|A : Recommended width or area (rule-dependent) needed to resolve violations

 * NET: VDD **Electromigration violations errors for VDD**

L_NAME	L_NUM	MODE	CURRENT	(>THRESHOLD)	RATIO	W A	REC	W A
(LEFT, BOTTOM) (RIGHT, TOP)								
metall1	11	average	52.233	(>52.200)	1.001	0.060		0.060
(157.716, 830.794)(160.100, 830.854)								
metall1	11	average	52.322	(>52.200)	1.002	0.060		0.060
(453.280, 559.930)(454.600, 559.990)								
metall1	11	average	52.323	(>52.200)	1.002	0.060		0.060
(453.104, 559.930)(453.280, 559.990)								
metall1	11	average	52.365	(>52.200)	1.003	0.060		0.060
(563.608, 820.762)(563.660, 820.822)								
metall1	11	average	52.361	(>52.200)	1.003	0.060		0.060
(563.100, 820.762)(563.176, 820.822)								
metall1	11	average	52.363	(>52.200)	1.003	0.060		0.060
(563.176, 820.762)(563.456, 820.822)								
.....								

Summary for NET: VDD

of total violations: 7299

* metall1 violations: 7258

* via1 violations: 7

* via2 violations: 7

* via3 violations: 7

* via4 violations: 7

* via5 violations: 7

* via6 violations: 3

* via7 violations: 3

**Electromigration violation
summary for VDD**

See Also

- [Performing Electromigration Analysis](#)

- [Displaying Static Electromigration Analysis Results](#)

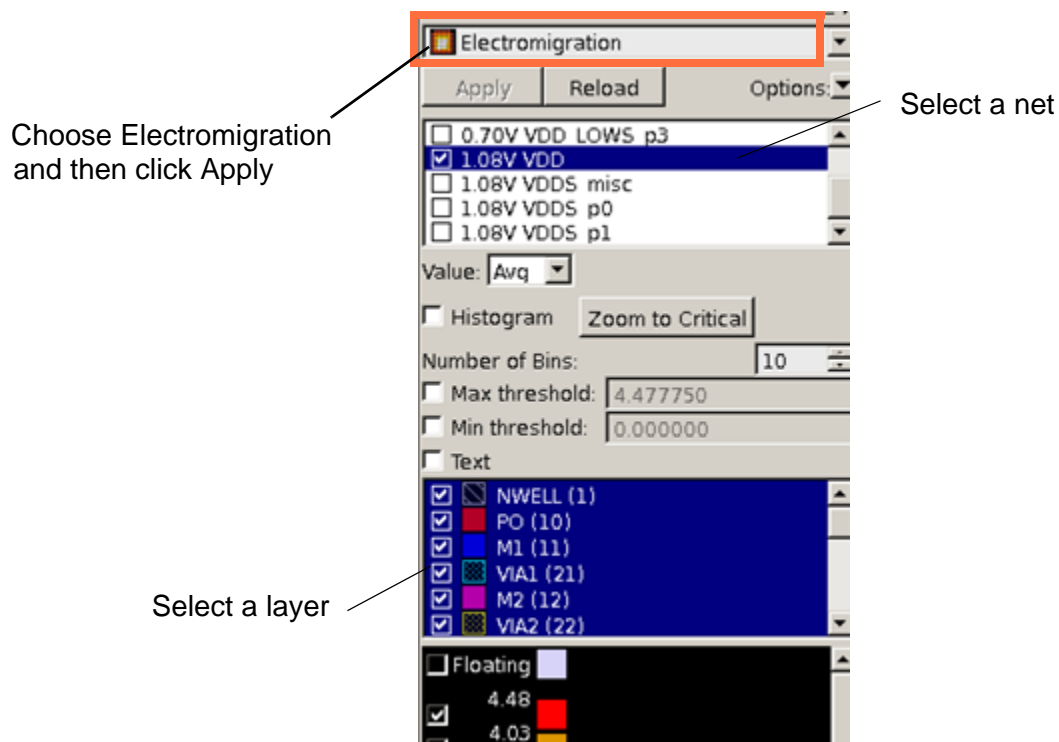
Displaying Electromigration Maps

When electromigration analysis is complete, choose Rail > Electromigration Map in the GUI to display the electromigration map for examining the electromigration violations graphically.

In the electromigration map, choose the layout layers and the design level for which you want to investigate possible violation problems.

Figure 9-3 shows an example of the electromigration map.

Figure 9-3 Displaying Electromigration Maps



To analyze only the shapes on a specific layer, select the layers from the list.

Use the histogram to determine the range of values to be displayed. This is useful if you want to display only the most critical area in the map.

To compare the electromigration map with the results from another rail analysis run, run the `current_rail_result result_name` command to specify a rail content, and then redisplay the map. This is useful in a what-if scenario when you assign a power or current value with the `set_rail_current` command and want to compare the current in a snapshot.

See Also

- [Performing Electromigration Analysis](#)
- [Displaying Static Electromigration Analysis Results](#)

Checking Threshold Information

When electromigration analysis is complete, run the `report_em_threshold_calculation` command to query the geometry that touches the specified rectangle area and check how the threshold is derived from electromigration rules for calculating current violations. Use the query result for debugging when complicated electromigration rules are used for the current violation calculation.

Use the `-mode` option to write the threshold information in either average, rms or peak mode.

Use the `-layer` and `-supply_net` options to specify the layer or the supply net for querying the geometry.

Note:

You must run the `report_em_violations` command with the `-keep_data` option before running the `report_em_threshold_calculation` command (see [Viewing the Electromigration Violation Report](#)).

Examples

The following example runs electromigration analysis and writes electromigration threshold information in the average mode.

```
# Read EM rule into PrimeRail
read_em_rules -format t1plus em_rule.itfem -design2itf_map itfem.map

# Enable EM debugging feature and specify which temperature corner
# of EM debugging information to be reported
report_em_violations -keep_data -temperature 25

# Print out EM threshold calculation information with the specified
# options
report_em_threshold_calculation -mode average -touching \
    {11.589 0.019 11.625 4.055}

# Release EM data for debugging from memory
remove_em_data
```

The following is the output example:

```
*****
Report : EM Detail Calculation Report:
Design : top
Version: <version>
Date   : <Date>
*****
NET: VDD
LOCATION : (9.964      3.810) (11.662      3.870)
EM_TYPE : AVERAGE
TEMPERATURE : 25.000
LAYER : M0 (metall)
WIDTH : 0.06
LENGTH : 650
THRESHOLD_CURRENT : 0.42 (mA)

NET: VSS
LOCATION : (11.589      0.019) (11.625      4.055)
EM_TYPE : AVERAGE
TEMPERATURE : 25.000
LAYER : M1(metal2)
WIDTH : 0.037
LENGTH : 625
THRESHOLD_CURRENT : 0.347 (mA)
```

The following example runs electromigration analysis and writes out threshold information in the rms mode.

```
# Set delta temperature for rms EM threshold calculation
set pr_aem_rms_deltat $deltat
# Read in EM rule into PrimeRail
read_em_rules -enable_comments -format tlua plus em_rule.itfem
               -design2itf_map itfem.map

# Enable EM debugging feature and specify which temperature corner
# of EM debugging information to be reported
report_em_violations -keep_data

# Print out EM threshold information with the specified options
report_em_threshold_calculation -mode rms -touching \
    {11.589      0.019      11.625      4.055}
```

The following is the output example.

```
*****
Report : EM Detail Calculation Report:
Design : top
Version: <Version>
Date   : <Date>
*****
NET: VDD
LOCATION : (9.964      3.810) (11.662      3.870)
```

```

EM_TYPE :      RMS
TEMPERATURE : 125.000
DELTA_TEMPERATURE : 5.000
LAYER : M0(metal1)
WIDTH : 0.06
LENGTH : 650
THRESHOLD_CURRENT : 0.501 (mA)
NET: VSS
LOCATION : (11.589      0.019) (11.625      4.055)
EM_TYPE :      RMS
TEMPERATURE : 125.000
DELTA_TEMPERATURE : 5.000
LAYER : M1(metal2)
WIDTH : 0.037
LENGTH : 625
THRESHOLD_CURRENT : 0.393 (mA)

```

The following example queries the average threshold of geometry within the range which is on layer m1 and the supply net vdd.

```

pr_shell> report_em_threshold_calculation -mode average \
        -touching { 0 0 10 10 } -layer m1 -supply_net [get_supply vdd]

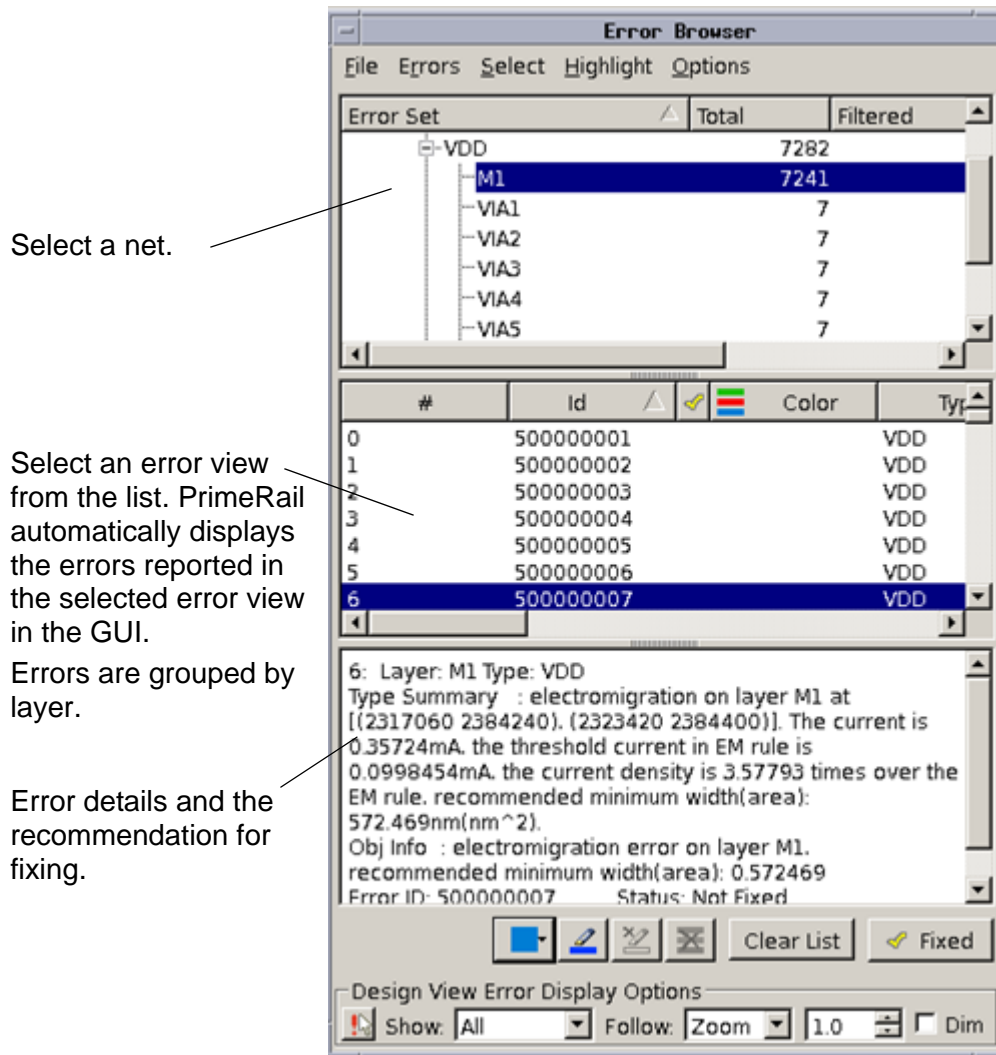
```

Opening Error Data in the Error Browser

In addition to electromigration maps or violation reports, you can generate error data during electromigration analysis and open the generated error views in the error browser.

To generate error views during electromigration analysis, use the `-error_cell` option of the `report_em_violations` or `set_em_options` command. When electromigration analysis is complete, choose Rail > Error Browser to open the Open Error View dialog box. Configure the settings for displaying only electromigration violations, and click Apply. The Error Browser dialog box then opens, as shown in [Figure 9-4](#).

Figure 9-4 Viewing Error Views in the Error Browser

**See Also**

- [Performing Electromigration Analysis](#)
- [Displaying Static Electromigration Analysis Results](#)

Considering Body-Bias Effects

PrimeRail considers body-bias effects during parasitic extraction and rail analysis. You can extract the parasitics on the nwell, pwell and contact mask layers, and generate current sources at the center of the body-bias pins.

Before you run rail analysis with body-bias effects, verify that you have the following information:

- The technology file that contains diffusion contact layer mask names, such as NWELL for the nwell layer mask name, PWELL for the pwell layer mask name, and CONTACT for the contact layer mask name.
- The FRAM view that contains bias, nwell pins and pwell pins for both standard and macro cells. For a tap cell, the diffusion contact shape belongs to the bias pin.
- The mapping file that contains the technical information for the nwell, pwell, and contact layers.
- The .lib or .db file that contains bias leakage power for both standard and macro cells.

When the input data is ready, execute the following steps:

1. Set the `extract_enable_body_bias` variable.

```
pr_shell> set extract_enable_body_bias 1
```

2. Specify the sheet resistance values for the pwell, nwell, and contact pins in units of ohms.

For example,

```
pr_shell> set extract_nwell 300.0
pr_shell> set extract_pwell 400.0
pr_shell> set extract_contact 2.0
```

3. Specify the extraction options, such as the settings for the TLUPlus and layer mapping files.

For example,

```
pr_shell> set_extract_supply_parasitics_options \
          -max_tluplus max.tlup -tech2itf_map layer.map
```

For more information about how to set extraction options, see [Setting Extraction Options](#).

4. Run the parasitic extraction.

For example,

```
pr_shell> extract_supply_parasitics [list VSS VNW VPW VDE]
```

When the extraction is complete, check the parasitic data that is saved in the RAIL_DATABASE/pgext directory. You can also view the extraction result via the parasitic map in the GUI.

For more information about power and ground net extraction, see [Extracting Supply Net Parasitics](#).

5. Specify the ideal voltage source locations and voltage settings.

For example,

```
pr_shell> create_taps -import $designidir/VR.src
pr_shell> set_voltage 0 -min 0 -object_list VSS
```

For more information about setting ideal voltage sources and voltage settings, see [Creating Taps for Ideal Voltage Sources](#) and [Setting Voltages](#).

6. Set the `power_exclude_ground_leakage_power` variable.

```
pr_shell> set power_exclude_ground_leakage_power true
```

7. Calculate power and current waveforms for the power and ground network, including the leakage power on the body bias pins.

```
pr_shell> update_currents -static
```

After you calculate pin power and current using the `update_currents` command, you can assign power to the body bias pins using the `set_rail_power` command for what-if purposes.

For more information about running power and timing analysis, see [Extracting Supply Net Parasitics](#).

8. Load the extraction data into memory using the `read_supply_parasitics` command.

For example,

```
pr_shell> read_supply_parasitics [list VSS VNW VPW VDE]
```

9. Perform rail analysis using the `calculate_rail_voltage` command.

For example,

```
pr_shell> calculate_rail_voltage [list VSS VNW VPW VDE] \
    -static -resistivity
```

For more information about running rail analysis, see [Performing Static Voltage Drop Analysis](#).

Displaying Results

When the extraction or rail analysis is complete, you can display the results via the parasitic map or the voltage drop map in the GUI.

10

Dynamic Rail Analysis

During static rail analysis, PrimeRail evaluates the voltage drop caused by the time average current flowing through a design. To evaluate the voltage drops caused when a large amount of cells switch simultaneously, perform dynamic rail analysis with the time-varying current waveforms that represent how the power and pin currents change with time.

Note:

You must complete power analysis and PG extraction before proceeding to the rail analysis stage. For more information, see [Extracting Supply Net Parasitics](#) and [Extracting Supply Net Parasitics](#).

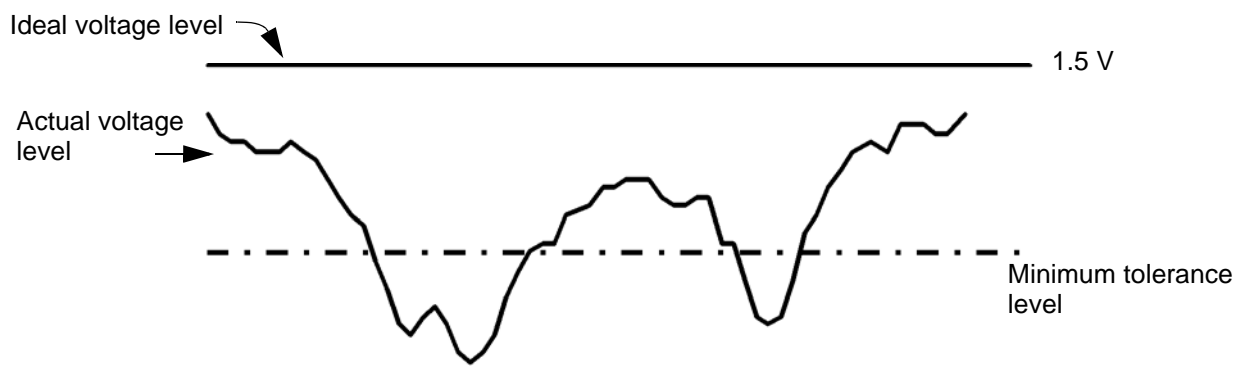
This chapter contains the following sections:

- [About Dynamic Rail Analysis](#)
- [Dynamic Rail Analysis Flow](#)
- [Required Input Data for Dynamic Analysis](#)
- [Library Characterization for Dynamic Analysis](#)
- [Dynamic Voltage Drop Analysis](#)
- [Dynamic Rail Analysis With Multiple Time Windows](#)
- [Dynamic Rail Analysis Results](#)
- [Dynamic Electromigration Analysis](#)
- [Dynamic Electromigration Analysis Results](#)

About Dynamic Rail Analysis

The purpose of rail analysis is to ensure that the power supply network in the design works as expected. One of the tasks of rail analysis is to analyze the voltage drop and rise in the power supply network. There are two types of rail analysis, static and dynamic. Static rail analysis models the power supply network as a resistive circuit and tries to capture the DC behavior of the power supply network. Dynamic rail analysis models power supply network as an RC (L) circuit, and tries to capture the transient behavior of the power supply network.

For most designs that use 90- and 65-nanometer process technologies, using a static approach is no longer sufficient and it becomes mandatory to analyze the actual variation of the supply voltage with respect to time for detecting potential chip failure conditions.



- Increasing power density and leakage power

In the advanced process technologies, power density values reach a level that is comparable to those that trigger the replacement of bipolar by the CMOS technology. Over-designing the power interconnect is no longer a viable option due to the higher power density.

Exponentially worsening leakage power consumption and tight full-chip power budgets also disallow deploying large amounts of decoupling cells. As a result, an accurate dynamic analysis tool is required to identify the problematic areas of the design and to provide suggestions on where to add additional chip resources for power grid integrity improvement.

- Power management techniques aggravate the problem

To address the increasing power densities and pressure from the consumer electronics markets for mobile applications, supply voltage are scaled to levels where the noise margin can no longer provide a safeguard against dynamic voltage drop fluctuations.

Active power management techniques such as shutting down portions of the chip and bringing it back up worsens the gradients of the on-chip current distribution, eventually leading to di/dt noises.

- Reducing static voltage drop does not solve the problem

During static rail analysis, decreasing the interconnect resistance is one way to improve the efficiency of the design. However, when the power supply network includes the dynamic properties, decreasing the interconnect resistance actually reduces the damping of unintended oscillations induced by inductive noises.

See Also

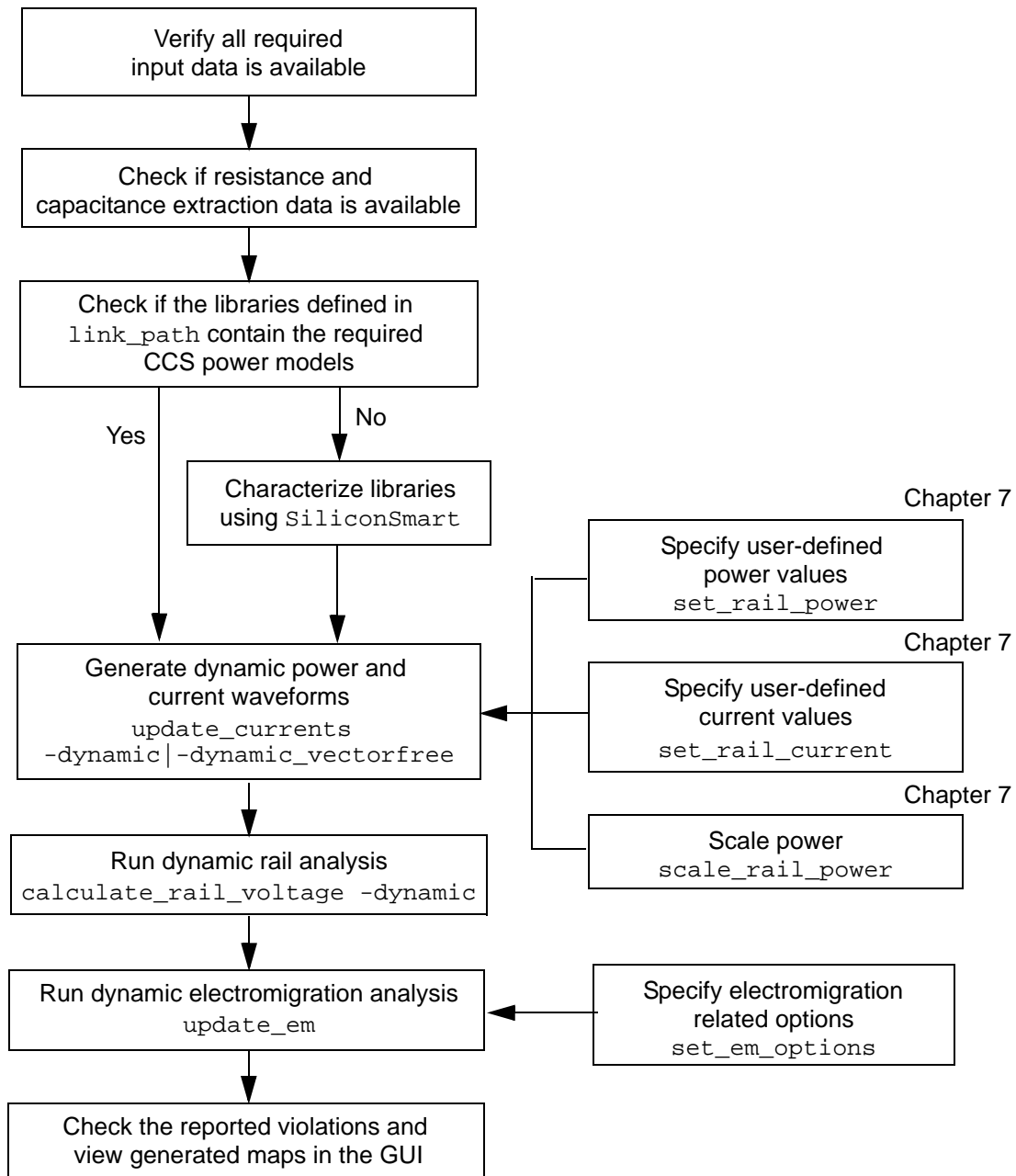
- [Dynamic Voltage Drop Analysis](#)

Dynamic Rail Analysis Flow

Similar to the static rail analysis flow, running dynamic rail analysis includes PG network extraction, power and current waveform generation and circuit simulation. Library power characterization is needed if the library does not contain the required CCS power models for running dynamic rail analysis. You can run dynamic rail analysis on the design either with or without detailed net switching activity information.

[Figure 10-1](#) illustrates the steps of a dynamic rail analysis flow.

Figure 10-1 PrimeRail Dynamic Analysis Flow

**See Also**

- [Dynamic Voltage Drop Analysis](#)

Required Input Data for Dynamic Analysis

PrimeRail requires a set of input data before analyzing maximum and minimum voltages in the design. In addition to the required data, CCS power model libraries are needed to analyze the dynamic voltage and current effects in the design.

The required input data includes

- The ideal voltage source locations

Use the `create_taps` command to create tap point objects in the current session. The taps are considered as part of the supply network in the subsequent rail analysis.

For more information about creating ideal voltage drop sources, see [Creating Taps for Ideal Voltage Sources](#).

- Both resistance and capacitance data for supply nets

Use the `extract_supply_parasitics` command to perform extraction on the specified power or ground nets. When the extraction process is complete, load the extraction data into memory with the `read_supply_parasitics` command before proceeding to rail analysis.

For more information about power and ground net extraction, see [Extracting Supply Net Parasitics](#).

- CCS power libraries

PrimeRail needs the waveform-related information in the CCS power libraries for generating the time-varying waveform at each PG port, including

- State-dependent leakage current
- Intrinsic parasitic models with voltage-independent or steady-state intrinsic resistances and capacitances
- Current waveforms for each power or ground pin defined in a dense format with all combinations of the output load, represented as the `dynamic_current` group
- Switch cell IV curves represented as the `dc_current` group
- Voltage-dependent intrinsic parasitics for all cells and IV curves for multithreshold-CMOS switch cells during an inrush current analysis flow.

If your design libraries do not have the necessary CCS power models, use the SiliconSmart tool to characterize these models. For more information, see [Library Characterization for Dynamic Analysis](#).

See Also

- [Running Library Characterization](#)
- [Dynamic Voltage Drop Analysis](#)

Library Characterization for Dynamic Analysis

Use the SiliconSmart tool to characterize libraries and to create CCS power models for dynamic rail analysis. Characterization is performed only once for each library in the design.

The characterization results are saved in the Liberty format (.lib). You need to convert the generated.lib file into the .db format before using it in the dynamic rail analysis flow. For a detailed description about converting a .lib file into the .db format, see the Library Compiler documentation.

This section includes the following topics:

- [Preparing Data for Library Characterization](#)
- [Editing the configure.tcl File](#)
- [Running Library Characterization](#)
- [Troubleshooting Library Characterization](#)

Preparing Data for Library Characterization

To characterize libraries and create CCS power models using the SiliconSmart tool, you must have

- The following required input data:
 - Reference NLDM, NLPM, or CCS timing library (.lib) for all cells in the design
 - SPICE transistor models for circuit elements
 - SPICE netlists for all cells in the design
- A run script that contains Tcl commands to automate the process of configuring, characterizing, and modeling a whole library of cells using the SiliconSmart tool.

For more information, see [SiliconSmart Run Script Example](#).

- A configuration file that contains Tcl commands to define parameter and definition settings for all cells of the characterization.

The `configure.tcl` file includes the following major parameter blocks:

- Global Configuration Parameters: Includes the global parameters that control high-level characterization settings and integration with third-party tools, such as SPICE.
- Default Pin Configuration Parameters: Includes the parameters that control the characterization settings for a class of pins—for example, setting the output load range for a set of pins.
- Operating Conditions: Includes the parameters that specify the process, voltage, and temperature for the characterization.

For more information, see [Editing the `configure.tcl` File](#).

- Access to the SiliconSmart, HSPICE and Library Compiler binaries. You must set path to the latest available HSPICE and SiliconSmart executables.

See Also

- [Running Library Characterization](#)
- [Troubleshooting Library Characterization](#)

Editing the `configure.tcl` File

Before running characterization with the SiliconSmart tool, you must set up the operating environment by editing the `configure.tcl` file. The SiliconSmart tool provides a default `configure.tcl` file in the `siliconsmart_install_path/etc` directory.

If you do not have a `configure.tcl` file, you run the `create -legacy mycharpoint` command to copy this default `configure.tcl` file to your `mycharpoint/config/` directory. Or you can create a `configure.tcl` file from scratch.

The parameter and definition settings in the configuration file are global and affect all cells during characterization. To define settings for a specific cell only, you set parameters in that cell's instance file. For more information, see the Editing Instance Files section in *SiliconSmart Online Help*.

[Table 10-1](#) lists the commonly used Tcl commands to modify the parameters in the configuration file. For a complete list of parameters and their usage, see the related chapters in *SiliconSmart Online Help*.

Table 10-1 Commonly Used Commands in SiliconSmart configure.tcl File

Commands	To ...
add_opc_supplies, add_opc_grounds	<p>Add all supply and ground nets, including primary, backup and internal nets.</p> <p>Make sure all supplies are in one <code>add_opc_supplies</code> statement, and all grounds are in one <code>add_opc_grounds</code> statement. If multiple <code>add_opc_supplies</code> or <code>add_opc_grounds</code> statements exist, only the last one is honored.</p> <p>Example:</p> <pre>add_opc_supplies ULVL_125C VDD 0.945\ VDDI 0.945 VDDR 0.945 VBP 0.945 add_opc_grounds ULVL_125C VSS 0.000\ VBN 0.000</pre>
power_meas_supplies, power_meas_grounds	<p>Control which supply or ground nets are measured for power consumption. Use this parameter to specify the list of power and ground supply names that should be used for the entire library of cells, except <code>internal_power</code> nets.</p> <p>Example:</p> <pre>set power_meas_supplies {VDD VDDI VDDR VBP} set power_meas_grounds {VSS VBN}</pre>

See Also

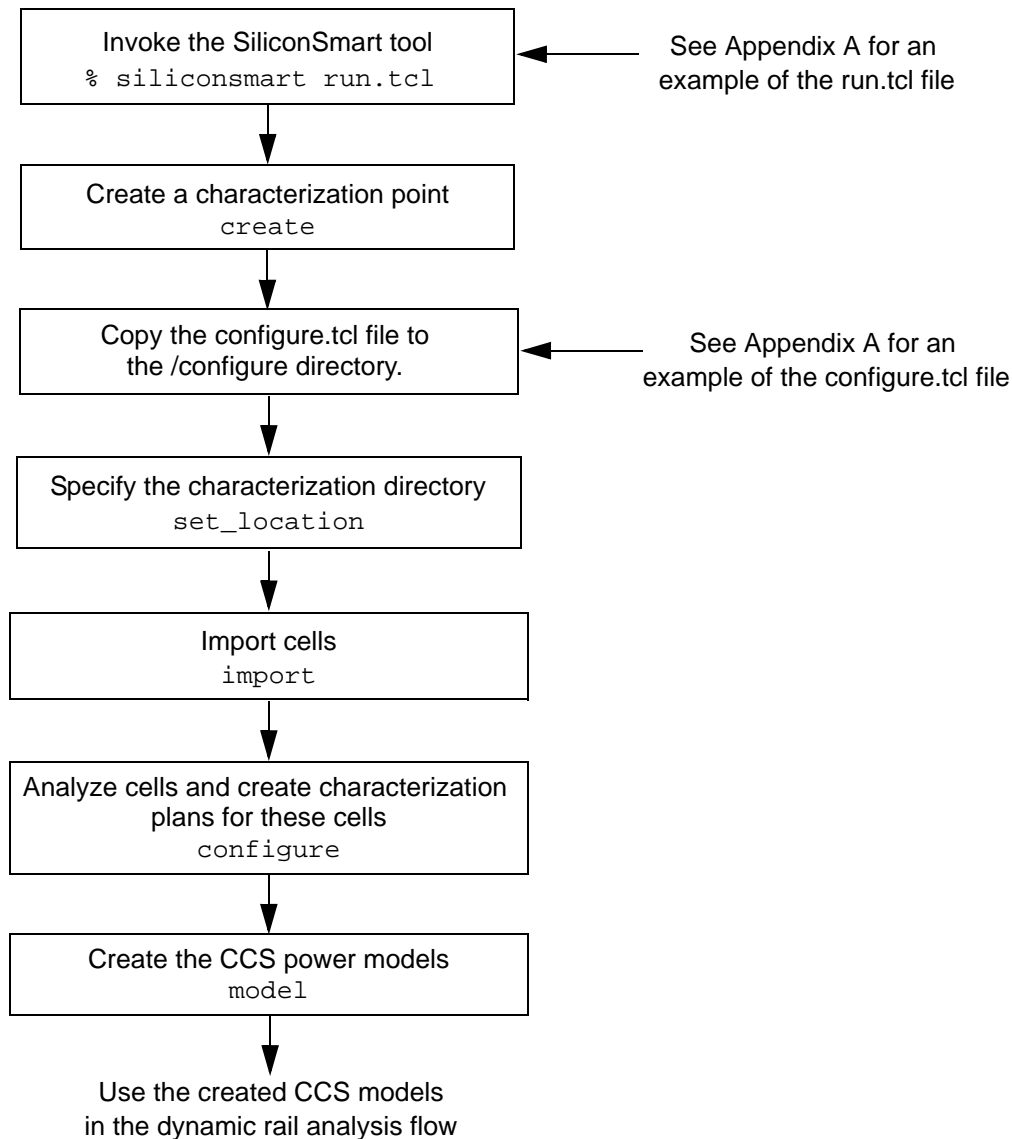
- [Preparing Data for Library Characterization](#)
- [Running Library Characterization](#)
- [Troubleshooting Library Characterization](#)
- [SiliconSmart configure.tcl Script Example](#)

Running Library Characterization

If your design libraries do not have the necessary CCS power models, use the SiliconSmart tool to characterize these models.

Figure 10-2 illustrates the steps for running library characterization with the SiliconSmart tool.

Figure 10-2 Running Library Characterization With SiliconSmart



To characterize libraries and generate models with the characterization data, do the following steps:

1. Invoke the SiliconSmart tool by using a run.tcl script file.

```
% siliconsmart run.tcl
```

2. Run the `create` command to create a characterization point with a directory structure for saving the related characterization data.

The SiliconSmart tool works within a predefined characterization directory structure. Relevant files are expected to reside within this structure and are generated in specific subdirectories. If you are not working in an existing characterization directory (`char_dir`), you must create the directory.

3. Copy the `configure.tcl` file to the `/config` directory in the SiliconSmart directory created at step 2.

4. Run the `set_location` command to specify the characterization directory location for all subsequent operations and to reload the `configure.tcl` file.

You must reload the `configure.tcl` file if you edited the `configure.tcl` to update the modified parameter settings.

5. Run the `import` command to import cells from a Liberty file (`.lib`) and a SPICE netlist into an existing characterization directory.

When a cell is imported from a netlist, an instance file (`.inst`) is created for that cell. This instance file defines the behavior of the cell, including information, function, characterization and modeling configuration options. You can edit the instance file when needed.

6. Run the `configure` command to analyze the cells and generate a characterization plan for each cell.

7. Run the `characterize` command to execute the characterization plan generated by the `configure` command.

8. Run the `model` command to create the specified new Liberty models. Characterized Liberty models are created in the `models/liberty` directory.

For more information about SiliconSmart commands and characterization flows, see the related chapters in *SiliconSmart Online Help*.

See Also

- [Preparing Data for Library Characterization](#)
- [Troubleshooting Library Characterization](#)
- [Dynamic Voltage Drop Analysis](#)

Troubleshooting Library Characterization

This section describes some of the typical errors you encounter during library characterization.

- Libraries With Multiple Power Nets

When a library contains multiple power nets, you need to establish the power-net-to-pin association for these power nets. Otherwise, Library Compiler issues the LBDB-27 error message to indicate that incorrect pins are placed with the `related_power_pin` attribute.

To set up the power-net-to-pin association for power nets, run the following steps:

1. Generate instance files for the cells by re-importing cells from a Liberty (.lib) file into an existing characterization directory.

Example:

```
import -liberty test.lib -netlist_dir netl_dir_name -ext .ext $cells
```

The Liberty model of each cell is saved and an instance file is generated for each cell. Instance files are stored in the `char_dir/control` directory.

2. Define the `pintype` statement in the `configure.tcl` file. You can obtain the `logic_high_threshold` and `logic_low_threshold` information from the PrimeRail threshold information section in the .lib file.

Do not define internal power in the `pintype` statement.

Example:

```
pintype default {
  set logic_high_name VDDI
  set logic_high_threshold 0.700
  set logic_low_name VSS
  set logic_low_threshold 0.300
  ...
}

pintype shifted1 -> default {
  set logic_high_name VDDR
}

pintype shifted2 -> default {
  set logic_high_name VDD }
```

3. Modify the pin definition of each instance file based on the power-net-to-pin association defined in the .lib file.

Example:

```
# pin definition
add_pin A default -input
add_pin EN shifted1 -input
add_pin X shifted1 -output
```

- **Modify Leakage Current Settings in the configure.tcl File**

In some cases, Library Compiler issues the LBDB-796 error message during compilation due to the total of absolute leakage current values does not add up to zero with a relative tolerance of 1.0e-6.

To ensure a correct compilation of the library file, remove the leakage acquisition warning from the SiliconSmart log file by including the following commands in the configure.tcl file:

```
Simulator option
If HSPICE is used, use simulator option
method=gear

gate_leakage_time_scaling_factor
set gate_leakage_time_scaling_factor 500

mode_input_leakage_current
set mode_input_leakage_current 1

liberty flavor
set liberty_flavor "2010.03"
```

- **Libraries With Preset Timing Arc Information**

If a library has preset timing arc information, Library Compiler issues the LBDB-267 compilation error. To resolve this, include the following commands in the configure.tcl file:

```
set liberty_multi_rail_format to v2 in configure.tcl
set liberty_multi_rail_format v2
```

In the run.tcl file, include the `-create_new_model` option for the `model` command. This instructs SiliconSmart to create a new library with the parameters that are specified in the `liberty_model` parameter block of the configure.tcl file. The resulting model is formatted such that the order of the cells, pin, arcs, and when conditions remains consistent across product releases.

Last, rename the newly generated library with your original library name.

See Also

- [Library Characterization for Dynamic Analysis](#)
- [Dynamic Voltage Drop Analysis](#)

Dynamic Voltage Drop Analysis

When time-varying waveforms are generated using the `update_currents` command, run the `calculate_rail_voltage -dynamic` command to analyze the actual variation of the supply voltage with respect to time for detecting chip failures.

Note:

You must complete power analysis and supply net extraction before performing rail analysis. For more details, see [Analyzing Power and Updating the Currents](#) and [Extracting Supply Net Parasitics](#).

After the tool executes the `calculate_rail_voltage` command, it automatically saves rail results in the cache. Each time you execute the command, the tool creates a result set with an associated name that is automatically assigned by the command or one that is specified with the `-result` option. For more information about results files, see [Displaying Static Rail Analysis Results](#).

Table 10-2 Commonly Used Options of the calculate_rail_voltage Command in Dynamic Rail Analysis Flow

Option	Description
<code>supply_nets</code>	Lists the supply nets to perform voltage drop analysis. Each element of the list can be a supply net collection or a name pattern. The list of supply nets is automatically expanded to include the supply nets that are connected to those listed by switch cells. To avoid the expansion, use the <code>-noexpand</code> option.
<code>-dynamic</code>	Selects the rail analysis mode as dynamic.
<code>-start_time</code> <code>-end_time</code>	Specifies the start and end time for the dynamic analysis, using .db time units. The start time is the time relative to the reference time (or the VCD start time) reported by the <code>update_currents</code> command.
<code>-step_size time</code>	Specifies the simulation time step size for performing dynamic analysis, using .db time units.
<code>-total_steps count</code>	Specifies the total number of time steps for performing dynamic analysis.

Table 10-2 *Commonly Used Options of the calculate_rail_voltage Command in Dynamic Rail Analysis Flow (Continued)*

Option	Description
<code>-result rail_result_name</code>	Names the output rail result. When not specified, the command assigns a unique name to the result.
<code>-replace</code>	Replaces an existing rail result. When used with the <code>-result</code> option, the command names the output result as <code>rail_result_name</code> . When used without the <code>-result</code> option, the command names the output result based on what is specified by the <code>current_rail_result</code> command.
<code>-error_cell</code>	Assigns the name to the output error cell. If no voltage threshold is specified with the <code>-voltage_threshold</code> option, the default voltage value 0 (zero) is applied.
<code>-fsdb_file</code>	Captures current and voltage drop waveforms in an output Fast Signal Database (FSDB) file. This FSDB current waveform file contains the information for each power and ground port instance. You can open the FSDB file and view the current waveforms in a standard waveform viewer, like nWave.
<code>-voltage_threshold</code>	Specifies the voltage threshold value for the error cell. The value is in volts and is an absolute number for both power and ground nets. If the <code>-error_cell</code> option is not specified, no error cell is generated.

Dynamic Rail Analysis With Multiple Time Windows

During VCD-based analysis, when more than one time windows are specified by the `read_vcd -time` command, PrimeRail automatically detects the windows and runs dynamic rail analysis on each of time windows. The power grid voltage and current values within the time gaps between adjacent simulation windows are assigned with the nominal supply voltage 0. The average and rms values are calculated only within the simulation windows; that is, the time duration of the taps are excluded during rail analysis to minimize the overall analysis time.

When analysis with multiple simulation windows is complete, the tool generates one consolidated rail result and one FSDB waveform file which encompass all simulation windows. The generated rail result contains power grid node voltage (including maximum,

minimum, and average) and resistor current (including maximum, average, and rms) values which are calculated over the given windows.

PrimeRail reads the time window settings from the `read_vcd -time` command and then runs rail analysis with the `calculate_rail_voltage` command. PrimeRail issues a message during simulation to indicate the presence of multiple simulation windows.

For example,

```
pr_shell> read_vcd des.vcd -strip_path {tb/aa} -time {{t1 t2} {t3 t4}}
pr_shell> calculate_rail_voltage [list VDD VSS] -dynamic
```

Information: There are 2 simulation windows.
Simulating window 1 of 2:start_time=2.0ns end_time=3.5ns, step_size=10ps

Each of the time windows has to be at least one simulation step size long and cannot overlap with each other (in time).

Note:

Using multiple simulation windows is not supported in the following analysis flows:

- Combined inrush and dynamic analysis
- Rail analysis with voltage regulators
- Reduced core model generation

See Also

- [Required Input Data for Dynamic Analysis](#)
- [Reading Switching Activity Information](#)
- [Dynamic Voltage Drop Analysis](#)
- [Dynamic Rail Analysis Results](#)

Dynamic Rail Analysis Without Transient Noises

Package inductances might generate oscillatory transient voltage waveforms at power grid nodes due to $L di/dt$ noises. The oscillatory waveforms at the onset of simulation can be large and are often reported as peak voltage drop or ground bounce values during dynamic rail analysis.

Use the `calculate_rail_voltage` command is enhanced with the `-report_start_offset` option to specify an interval in which the calculated power grid voltage and current values are excluded from the analysis report and GUI map generation. This allows you to ignore these oscillatory waveforms and focus on the transient circuit behavior of the power grids when the noise reduces to an insignificant level.

The following is an example:

```
pr_shell> calculate_rail_voltage -report_start_offset 1.0 \  
                                -dynamic {VDD VSS}
```

Note:

Specifying the `-report_start_offset` option does not have an impact on the voltage and current waveforms for taps, package ports and probed PG pins that are saved in the Fast Signal Database (FSDB) file.

See Also

- [Dynamic Voltage Drop Analysis](#)
- [Dynamic Rail Analysis Results](#)

Dynamic Rail Analysis Results

When rail analysis is complete, the tool saves the analysis results and a log file in the / RAIL_DATABASE/design_name/rail directory. The log file contains a detailed voltage drop report for all the supply nets. For each net, the log reports both the maximum and minimum voltage drop values, and five instances with the highest voltage drop values. This information is useful to detect and analyze areas with high voltage drops.

For more information about rail analysis results, see [Displaying Static Rail Analysis Results](#).

With the rail analysis results, you can

- Display the voltage drop map and use the query tool in the GUI. You can view areas with high voltage drops or resistances graphically, by using the voltage drop or the minimum path resistance map. The problem areas are highlighted in different colors.

For a detailed description of how to display a voltage drop map, see [Displaying Voltage Drop Map](#).

- Calculate effective voltage drops and rises across the entire simulation period or within timing windows of each cell.

For more information, see [Reporting Effective Voltage Drops](#) and [Reporting Switching-Window-Based Effective Voltage Drops](#).

- Save the voltage drop violations or resistance calculations to an error cell by using the `-error_cell` option with the `calculate_rail_voltage` command. You can then open the generated error cell in the error browser.

For more information about how to open error views in the error browser, see [Opening Error Data in the Error Browser](#).

- Save the dynamic rail analysis result in the FSDB format by using the `-fsdb_file` option with the `calculate_rail_voltage` command.

If you want to probe a set of PG pins and save their voltage and current waveforms to the output FSDB file, first run the `set_probes pg_pins` command and then rail analysis with the `calculate_rail_voltage -fsdb_file` command.

Here is an example:

```
pr_shell> set_probes [get_pg_pins -of VDD]
pr_shell> calculate_rail_voltage [list VDD VSS] -dynamic -top_pin \
    -fsdb_file output.fsdb
```

To report or remove probes, use the `report_probes` or `remove_probes` command.

- Examine peak voltage drops at the PG pins for the supply nets with the `report_rail_voltage` command. For more information, see [Debugging Rail Results](#).
- Write the calculated rail data from the current session to a text file with the `write_rail_data` command. You can then import this output file in another session with the `set_rail_power -import` command, saving time from setting related data when rerunning rail analysis. For more information, see [Writing Rail Data](#).
- Compare the voltage drop map with the results from another rail analysis run by running the `current_rail_result result_name` command to specify the rail content, and then redisplaying the map. This is useful in a what-if scenario when you assign a different power or current value with the `set_rail_power` or `set_rail_current` command and want to compare the voltage effects. For more information, see [Toggling Between Rail Results](#).
- Display a minimum path resistance map for quickly detecting power and ground network weaknesses in the design. For more information, see [Minimum Path Resistance Analysis](#).

Dynamic Electromigration Analysis

After you perform dynamic rail analysis with the `calculate_rail_voltage -dynamic` command, you can check and report peak, average, and root mean square (rms) current violations by running the `update_em` and `report_em_violations` commands. The tool checks for the nets whose current exceeds the peak and rms limits defined in the electromigration rule file and reports the analysis results in the log file.

To check which layers do not have the defined electromigration rule, open the `PR_CHECKING_DIR/em.missing_rules` file. This file is generated when the `report_rail_checks` command run is complete.

To check for dynamic electromigration violations,

1. Verify the peak and rms rules are defined in the electromigration file. Load the electromigration rule file using the `read_em_rules` command.

For more information about reading in the electromigration rules, see [Reading Electromigration Rules](#).

2. Complete parasitic extraction, power and current calculation, and dynamic voltage drop analysis.

For a detailed description about power analysis, supply network extraction, and voltage drop analysis, see [Analyzing Power and Updating the Currents](#), [Extracting Supply Net Parasitics](#), and [Performing Static Voltage Drop Analysis](#), respectively.

3. Run the `set_em_options` command with the following options to specify the configuration information for electromigration analysis. For instance,
 - Use the `-temperature` option to set a temperature value, in degrees Celsius, for evaluating electromigration constraints. By default, the command uses the operating condition in the first library in the link path for checking.
 - Use the `-scale_factor` option to specify a uniform scale factor for the current threshold in the electromigration rule before electromigration violation checking. For what-if purposes, you can change the setting to a higher temperature or a different scaling factor, and rerun the `update_em` command to compare the different electromigration results.
 - Use the `-width_limit` option to set an absolute minimum width rule for polygons that is used during electromigration violation checking. Use the `-relative_width_limit` option to set a minimum width for polygons, as a percentage of the layer minimum width that is defined in the technology file. The default is 80%.

For more information about how to specify options for electromigration analysis, see [Setting Electromigration Options](#).

4. Run the `update_em` command to perform electromigration analysis based on the configuration options specified by the `set_em_options` command. If the PrimeRail GUI is open when executing the `update_em` command, the tool automatically displays the electromigration map or error view when the analysis is complete.

When analysis is complete, you can check for current violations with the analysis results. For more information about how to examine the analysis results, see [Dynamic Electromigration Analysis Results](#).

See Also

- [Checking Design Readiness](#)

- [Dynamic Voltage Drop Analysis](#)

Dynamic Electromigration Analysis Results

Run the `report_em_violations` command (or choose Rail > EM Analysis in the GUI) to report the calculated electromigration violations in an ASCII file.

The `report_em_violations` command only generates and updates the violation reports and error cells. The command does not update in-memory electromigration results, such as electromigration maps. To update the electromigration map, run the `update_em` command.

You can

- Generate error cells during electromigration analysis and open the generated error cells in the error browser using the `-error_cell` option.
- Choose which layers to be reported in the electromigration violation report with the `-layer` option, and specify layer names or ID numbers. By default, all the layers are included in the electromigration violation report. If the polygons are not on the specified layers and they violate the defined electromigration rule, these polygons are not reported.
- Customize the violation report without rerunning the `update_em` command by using the `-scale_factor`, `-width_limit`, and `-relative_width_limit` options. For example, to verify the design does not have currents exceeding 90% of the allowed value, set the scaling factor to 0.9 using the `report_em_violations -scale_factor` command.

Note:

Setting the `-scale_factor`, `-width_limit`, and `-relative_width_limit` options with the `report_em_violations` command overrides the settings defined by the `set_em_options` command.

The following is an example of the dynamic electromigration violation report:

```
unix> vi pr_work_directory/em_report1.txt
```

```
.....      Electromigration rms violations errors for VSS
Dynamic EM violation report (mode rms):
* NET: VSS
L_NAME L_NUM MODE CURRENT (>THRESHOLD) W|A REC W|A (LEFT,BOTTOM) (RIGHT,TOP)
-----
metall 3 rms 0.135 (>0.001) 0.360 48.425 (0.360,135.990) (406.520,136.350)
metall 3 rms 0.135 (>0.001) 0.360 48.692 (406.720,135.990) (408.500,136.350)
metall 3 rms 0.135 (>0.001) 0.360 48.695 (408.700,135.990) (410.120,136.350)
metall 3 rms 0.113 (>0.001) 0.360 40.536 (410.320,135.990) (413.900,136.350)
metall 3 rms 0.113 (>0.001) 0.360 40.542 (414.100,135.990) (421.680,136.350)
metall 3 rms 0.090 (>0.001) 0.360 32.392 (417.880,135.990) (421.460,136.350)
metall 3 rms 0.090 (>0.001) 0.360 32.398 (421.660,135.990) (425.240,136.350)
.....

Summary for NET: VSS
# of total violations: 13
* metall violations: 13
```

Current width

Recommended width for fixing electromigration violations

Electromigration violation summary for VDD

See Also

- [Dynamic Voltage Drop Analysis](#)
- [Displaying Static Electromigration Analysis Results](#)

11

Rail Macro Models for Milkyway Designs

This chapter discusses how to create rail macro models for hard macro cells for Milkyway designs.

This chapter has the following topics:

- [What Is a Rail Macro Model?](#)
- [Using a Rail Macro Model](#)
- [Creating Rail Macro Models](#)
- [Creating Macro Models With On-Resistances](#)
- [Creating Simple Rail Macro Models](#)
- [Updating Macro Models](#)
- [Setting Mode Sequences for Macro Models](#)
- [Checking the Macro Model Content](#)
- [Reporting Unmodeled Macro Cells](#)
- [Displaying Current Source Information](#)
- [Viewing Macro Models in the GUI](#)

What Is a Rail Macro Model?

A rail macro model is an abstract representation of the internal physical model (including PG structure and currents) and the transistor-level electrical model of a hard macro cell. By making a soft macro into a hard macro, many data objects inside the soft macro are no longer needed for analysis and thus freed from memory. This greatly reduces memory usage and improves performance of the rail analysis.

For instance, you need to create a macro model for your design to determine power budget and floorplan of the macro if your design:

- Is a hard macro that has only a behavior model
- Uses IP modules and has no cell-level data
- Is a GDSII macro that has only layout data and has only one cell-level model
- Is too large or is an analog block

The power consumption of these macro cells or the power and ground routing that exists inside them can affect the voltage and current supplied to other regions of the chip. Therefore, you should consider the internal power and ground network and power consumption of the cells during rail analysis.

See Also

- [Using a Rail Macro Model](#)
- [Power Management Cells](#)
- [Voltage Drop Analysis With Regulators](#)

Using a Rail Macro Model

The rail macro model feature allows you to capture the analog effects in a hard macro during full-chip analysis with reusability. This hard macro can be a memory or an analog block. Sometimes cells, like pad cells, switch cells or soft macros, can be modeled as rail macro models.

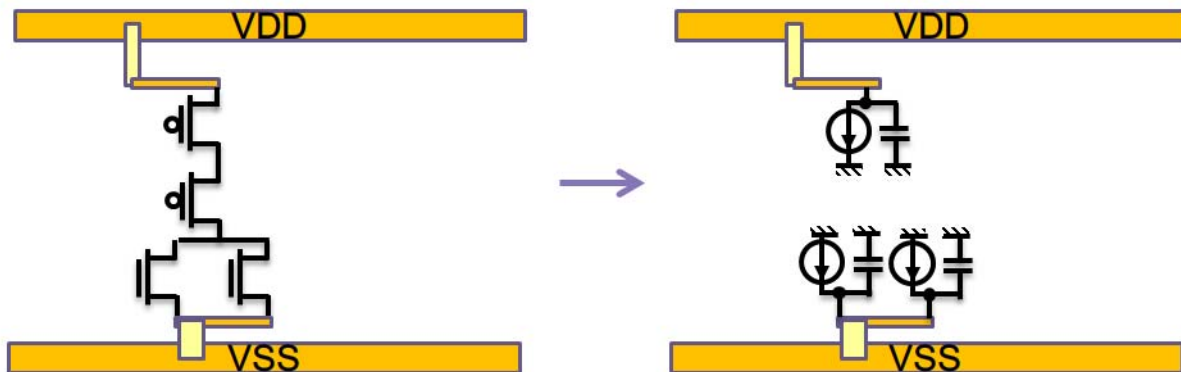
You can create different types of rail macro models, depending on the type of analysis flow, the level of accuracy desired or the availability of the input data. For example, during static analysis, you create a rail macro model using the geometry information from the CEL view or the GDSII file. This is the simplest form of the rail macro model in which current is evenly distributed over all device locations in the CONN view with the power value that is generated by the power analysis.

To use a rail macro model with more accurate data or in the dynamic analysis flow, update the rail macro model with the SPICE or simulation information. As shown in [Figure 11-1](#), with the SPICE information, PrimeRail models the transistors as current source and intrinsic capacitance points in parallel and scales the capacitance and current waveforms to each device location.

When using SPICE information and GDSII data as electrical and physical inputs, PrimeRail simulates the circuit based on the device netlist and records the device current waveforms in a binary file. The generated simulation-based rail macro model contains a CONN view with internal-pin current waveforms and weighted capacitances.

For more information about CONN views and current source information, see [Connectivity Views](#) and [Current Source Locations](#).

Figure 11-1 Modeling a Rail Macro Model With Transistor-Level Information



Each rail macro model contains the following information: power and ground net capacitance, intrinsic capacitance, current source locations, and one or multiple modes. Because the power consumption of a building block is dominated by control signals, you can categorize the current waveforms of the circuit into several different waveform signatures based on different operation modes, such as read mode, write mode or stand-by mode.

The rail macro model flow also provides the following capabilities:

- Support of the operation modes

When generating a macro model, the tool determines the operation modes for the model according to the macro manufacturer's datasheet or timing and power libraries. Each mode in the macro model contains current waveform information, which is automatically triggered and applied to the rail analysis according to the VCD input. Similarly, it can be automatically adopted into the vectorfree dynamic flow, or assigned manually. PrimeRail allows you to store different operation modes (for example, read or write mode for an SRAM) in the macro model. The operation mode is determined by the state of I/O ports of the design, and each mode results in different current waveforms. When performing a full-chip rail analysis, PrimeRail automatically picks up the waveforms of the chosen mode based on the given VCD input.

For each macro model, the operation mode is similar to the WHEN condition in the Liberty format. Like CCS power libraries, each operation mode has a current waveform for the power and ground port and is distributed to each current source of the port.

- Support of both synchronized and asynchronous modes

PrimeRail also allows you to specify multiple trigger signals (a bus or a bundle of signals) for the mode created. Creating asynchronous modes is also supported. A synchronized circuit design relies on a single control signal (usually known as a clock) to trigger the operation. Asynchronous designs do not require such a signal but are active whenever the input signal changes. For example, an SRAM contains one synchronized write port and one asynchronous read port.

- Support of multiple PVT designs

PrimeRail also supports handling multiple PVT corners in the macro models. You can generate macro models based on different PVT settings by specifying a corner in the model description file. The tool then runs power and rail analysis by obtaining the corner information from the power report and automatically picks up the closest corner from the macro model. Note that unlike with standard cells, no scaling (interpolation and extrapolation) is performed for macro models.

Each macro model is identified by cell names and power and ground net names, and contains parasitics and current waveform information. The rail macro model is stored in the CONN view. You can display the generated CONN view with current source and on-state resistance locations in the GUI.

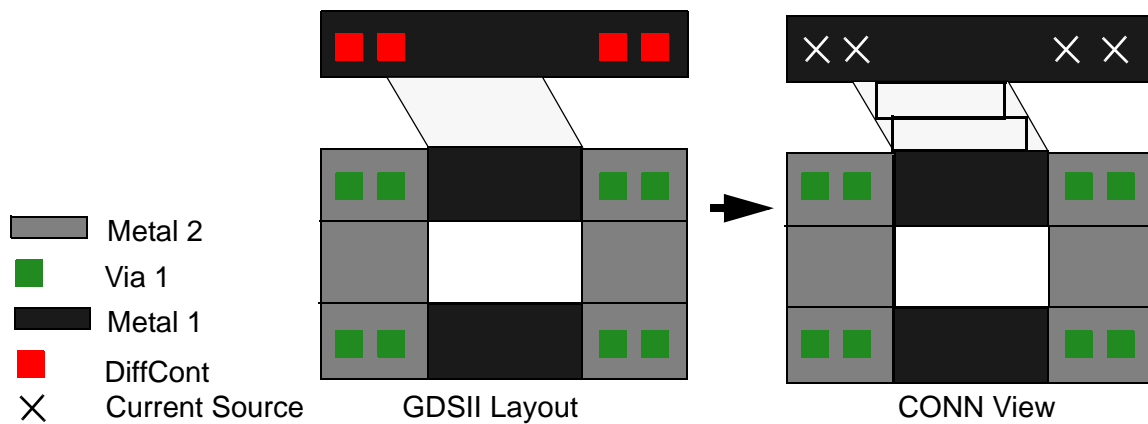
See Also

- [What Is a Rail Macro Model?](#)
- [Creating Rail Macro Models](#)
- [Updating Macro Models](#)
- [Viewing Macro Models in the GUI](#)

Connectivity Views

A CONN view is very similar to a CEL view; it represents the power and ground networks inside the cell in a viewable manner. All metal shapes are pin objects and all vias are rectangles. As is depicted in [Figure 11-2](#), there is one particular design where polygons (trapezoid for example) are broken down into rectangles. Those rectangle shapes are for PrimeRail to perform extraction. However, the original polygon is still created as a user shape and is used only by the GUI. The GUI draws both the polygon and the rectangles.

Figure 11-2 Comparing CEL View with CONN View



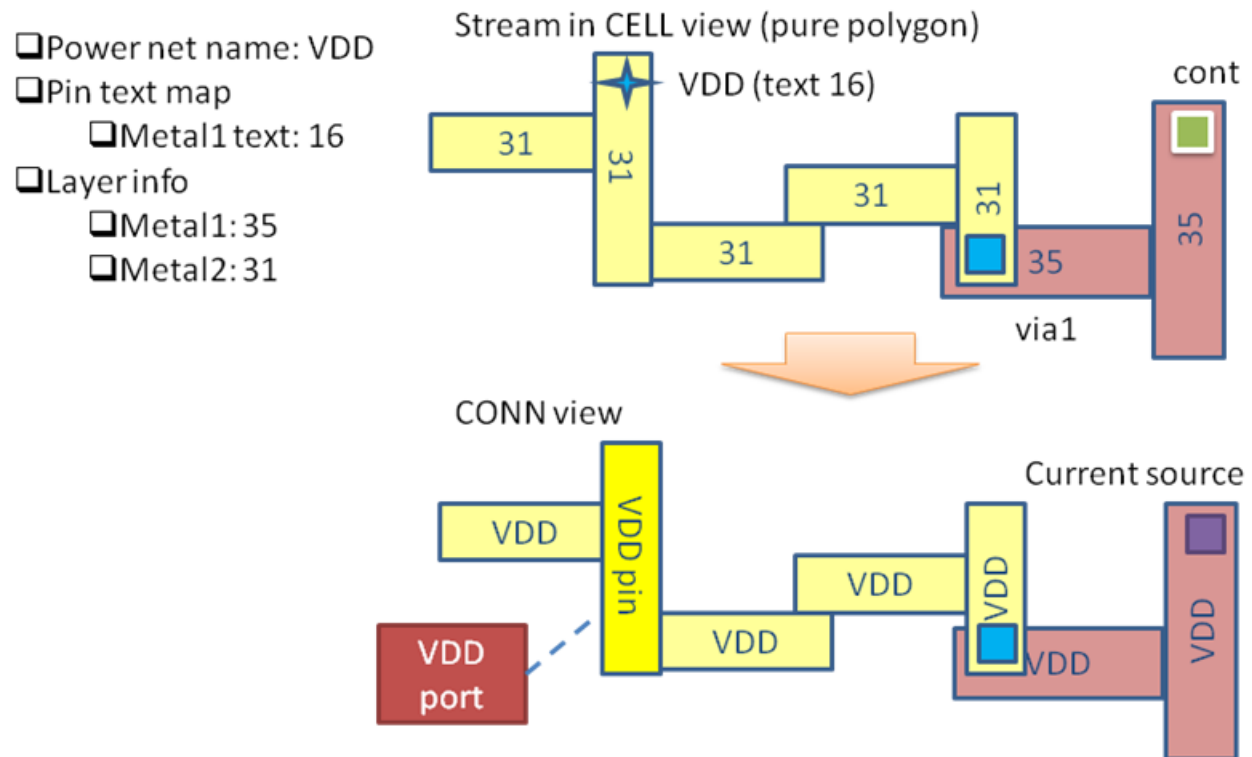
The following cells require CONN views to correctly re-create the resistance network:

- Power cells
- Cover cells
- Hard macro cells
- I/O pad cells

Figure 11-3 shows how PrimeRail builds a VDD net by finding the identifying text and by tracing connections through touching and overlapping rectangles. The upper example is the unprocessed stream-in data. The route geometries are scattered and the net property is unknown. PrimeRail identifies the power or ground pin text by the given power net name VDD and then uses the VDD text as a source to tag all touching and overlapping rectangles with VDD, as shown in the lower example. Next, PrimeRail defines the VDD text point as a VDD pin rectangle and defines the lowest contact layer as a current source. In this case, PrimeRail uses diffcont as the current source tap point.

Note that when the pin text is unknown and only net text is available, PrimeRail cannot determine the pin location and thus defines all VDD metals as pins.

Figure 11-3 Tracing Pin Net Texts for Creating CONN View and Current Source File



See Also

- [Using a Rail Macro Model](#)
- [Current Source Locations](#)

Current Source Locations

Running the `create_rail_macro_model` command automatically generates the information of how current sources are distributed inside the macro model. The current source information is attached to the CONN views; PrimeRail uses the information to improve rail analysis accuracy by placing current sinks of hard macros into the current source locations.

See Also

- [Using a Rail Macro Model](#)
- [Connectivity Views](#)

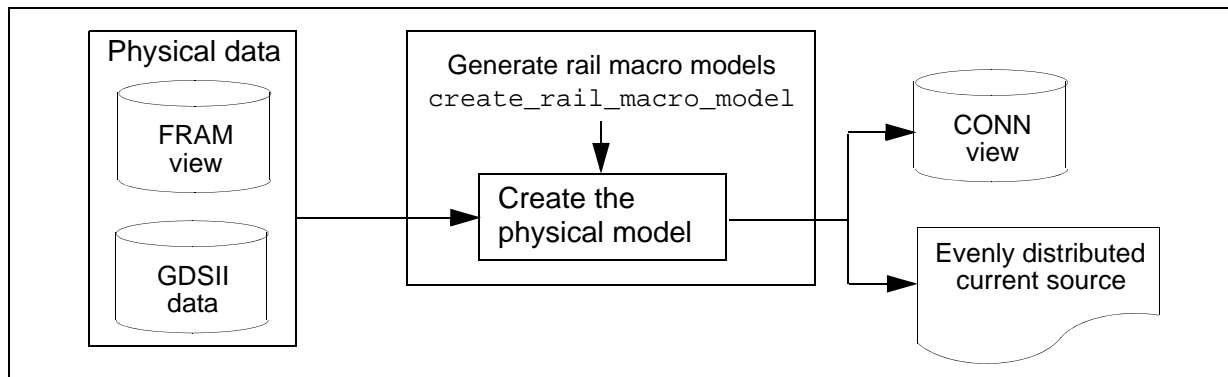
Creating Rail Macro Models

To create a rail macro model using the geometry information from the CEL view or the GDSII file, run the `create_rail_macro_model` command. If the macro cell is created by the Embed-It! Integrator tool, you can model this macro cell using the Embed-It! Integrator compiling results.

The generated rail macro model contains a CONN view and the current source information. In the CONN view, the current is evenly distributed over all device locations with the power value that is generated by the power analysis.

To use the generated rail macro model in the dynamic analysis flow, you need to update the rail macro model with more accurate information. For a detailed description about how to update a rail macro model, see [Updating Macro Models](#).

Figure 11-4 The Data Flow for Creating a Macro Model With Milkyway or GDSII Data



In PrimeRail, you create rail macro models in one of the following ways:

- [Creating Macro Models With Milkyway Data](#)
- [Creating Macro Models With GDSII Data](#)
- [Creating Macro Models With Synopsys Embed-It! Integrator Data](#)

Creating Macro Models With Milkyway Data

When the FRAM and CEL views of the hard macro are available, run the `create_rail_macro_model` command to create a rail macro model for the hard macro cells. Specify the name of the cell and the library in which the cell resides. If the power and ground nets are not defined in the FRAM view, specify either power or ground nets whose data is to be extracted.

The following example generates a rail macro model for the macro cell named `sram64x64` and saves the generated macro model in the output library named `test_lib`. The power and ground nets are `VDD` and `VSS` respectively.

```
pr_shell> create_rail_macro_model -library test_lib -cell sram64x64 \
        -power_nets {"VDD"} -ground_nets {"VSS"}
```

See Also

- [Creating Rail Macro Models](#)

Creating Macro Models With GDSII Data

In the GDSII flow, you create rail macro models for the macro cells by using the GDSII data as physical data input, with layer mapping and technology files. The command traces connectivity through PG pin shapes and determines the locations of resistors based on the design layout and the process-specific parameters, such as layer thicknesses and resistances for each layer. The layer mapping file describes the layer relationship between the GDSII file and the technology file.

In cases when the lower layers have incorrect texts, you can extract the power and ground net information from the FRAM view, but not the GDSII file. To do this, run the `create_rail_macro_model` command with the `-extract_by FRAM` option.

Before you run the `create_rail_macro_model` command, verify that you have the following input data:

- A GDSII file that is LVS-clean and without DRC or design errors. The GDSII file must contain the complete design layout and text for the ports and nets. Do not use a GDSII file after optical proximity correction (OPC), or incorrect CONN views might be created.
- A technology file that is used to create a Milkyway library. Use the same technology file as the one for the parent design of the macro, or the one that is aligned with the design library of the parent design. If the technology file is unavailable, generate one from the parent design library or from the process library of the macro model in the IC Compiler tool.

For more information, see [Setting Up the Technology File](#).

- A layer mapping file that describes the mapping between the GDSII and Milkyway layers. Unlike a GDSII file that contains only the XY coordinate information, the layer mapping file describes the layer mapping between the technology and GDSII files. If a layer mapping file is unavailable, you can either manually create one or use the `-layer_translation_table` option with the `create_rail_macro_model` command to generate a layer translation table file (*.ltx) by using the Synopsys Embed-It! Integrator tool.

For more information, see [Layer Mapping Files](#).

Note:

You need to have the related license for performing layer translation with the Synopsys Embed-It! Integrator tool.

- (Optional) A tracing file that defines the tracing sources for CONN view generation. Use this file when the input GDS file does not have layer texts for PAD cells.

For more information, see [Layer Text Files](#).

Example 1

The following example creates rail macro models for hard macro cells with a GDSII file, a manually-created layer mapping file, and a technology file:

```
pr_shell> create_rail_macro_model \
        -cell sram_test \
        -library library_test \
        -gds_file gds_test \
        -layer_map_file map_test\
        -extract_by FRAM \
        -tech_file test
```

In this example, PrimeRail creates a rail macro model for the macro cell named sram_test using the input data from the specified GDSII file, the layer mapping file, and the technology file. The `-extract_by FRAM` option is specified to indicate the geometry information is extracted starting from the FRAM view layer.

Example 2

The following example invokes the Synopsys Embed-It! Integrator tool to create a layer translation table file (*.ltd) based on the GDSII and technology files. The output test.ltd file is then used to create a rail macro model for the macro cell named sram8x8.

```
pr_shell> create_rail_macro_model
        -library test \
        -cell sram8x8 \
        -gds_file sram8x8.gds \
        -layer_translation_table test.ltd \
        -tech_file mw.tf \
        -power_nets {VDD} \
        -ground_nets {VSS}
```

Layer Mapping Files

A layer mapping file contains a translation table that describes the mapping between the foundry layers in the GDSII file and the design layers in the technology file. Usually a layer mapping file is provided by the foundry. If a layer mapping file is unavailable, you either manually create one or use the Synopsys Embed-It! Integrator Online tool to perform layer translation for creating a layer translation table file (*.ltd).

Manually Creating a Layer Mapping File

In a text file, define the mapping between foundry layers and design layers. The following is an example of a layer mapping file:

```
; MilkywayLayer:MilkywayDataType   GDSIILayer:GDSIIDataType
        6:0                           6:0
        10                           6:7
; The above line maps GDSIILayer 6 of GDSIIDataType 7 to
; MilkywayLayer 10 of MilkywayDataType 0
        17:0                           17:0
        17:7                           17:7
        18:5                           17:11
; The above line maps GDSIILayer 17 of GDSIIDataType 11 to
; MilkywayLayer 18 of MilkywayDataType 5
        40:0                           40
; The above line maps GDSIILayer 40 of any GDSIIDataType to
; MilkywayLayer 40 of MilkywayDataType 0
```

Layer Text Files

A layer text file defines the tracing sources for CONN view extraction when the input GDSII file does not have layer texts for some cells, such as pad cells.

In the layer text file, define the tracing sources in the following syntax:

```
pg_net_name mask_name x_in_Milkyway_unit y_in_Milkyway_unit
```

When the layer text file is available, use the `-tracing_file` option of the `create_rail_macro_model` command to specify the tracing file to be used.

Note that when a tracing file is used, the tracing sources from the GDSII text or the pin shapes from the FRAM view will be ignored.

See Also

- [Creating Rail Macro Models](#)

Layer Translation Table Files

Use the `-layer_translation_table` option of the `create_rail_macro_model` command to create a layer translation table file for layer translation. Use this option when the memory block is created by the Synopsys Embed-It! Integrator tool.

The following example shows the generated layer translation table file:

```
# LAYOUT_TECH DATABASE FOR <name> #
# CSR No. : <number>#
# LAYERMAP_DOC_FILENAME : <name> #
# LAYERMAP_DOC_VERSION : <version> #
# LAYERMAP_DOC_DATE :<date>#
# Synopsys CVS $Revision: 1.1 $ (VSKP) #
# Synopsys Release $Date: 2012/05/17 04:33:26 $ (GMT) #
# ----- #
# layer-mapping file for: XXX-XXX
# created on: Tue Dec 06 10:17:09 IST 2011
#-----#
# design layer foundry layer
0:0 D # cell_text
0:1 222:0 # std_text
0:2 127:0 # user_text
10:0 4:0 # nwell
2:0 6:0 # ndiff
3:0 7:0 # pdiff
4:0 16:0 # poly
5:0 40:0 # contact
6:0 41:0 # metall
17:0 55:0 # via2
```

See Also

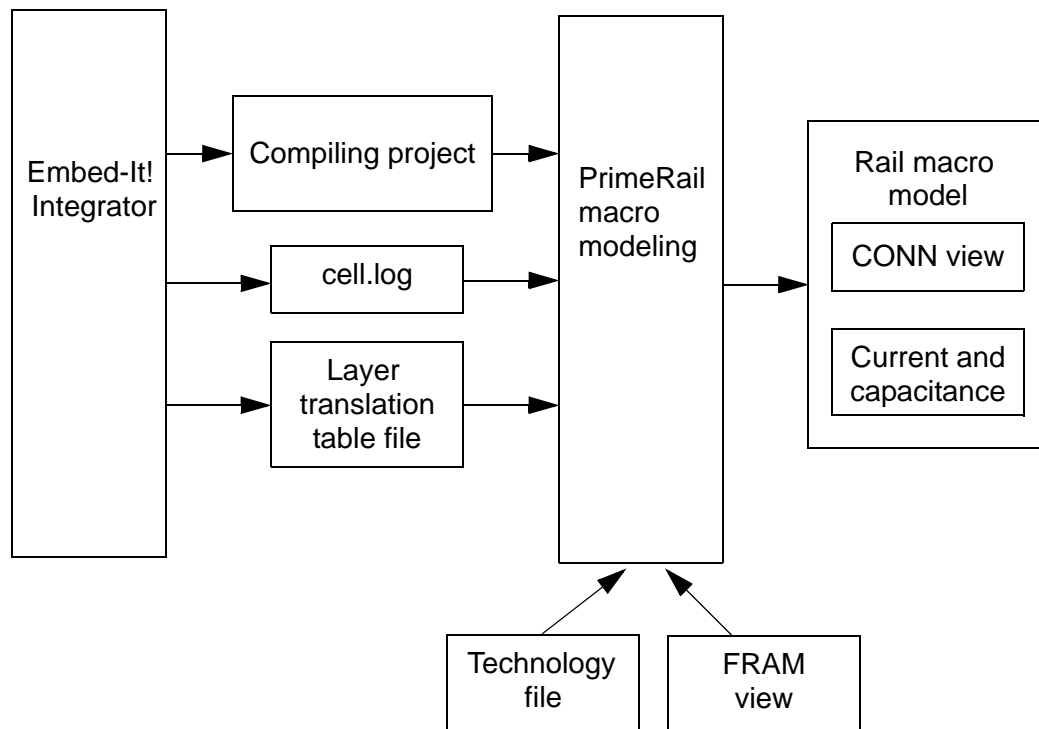
- [Creating Rail Macro Models](#)

Creating Macro Models With Synopsys Embed-It! Integrator Data

The Synopsys Embed-It! Integrator tool is a memory compiler tool that creates components in a System-on-Chip (SoC) design. If a macro cell is created by the Embed-It! Integrator tool, you can create a rail macro model for this macro cell using the compiling results which contain the IC Validator plug-in results.

[Figure 11-5](#) illustrates how PrimeRail retrieves data from the Embed-It! Integrator tool compiling results for creating a rail macro model.

Figure 11-5 Data Flow Between the PrimeRail and Embed-It! Integrator Tools



To use the compiling results from the Embed-It! Integrator tool for creating rail macro models, run the following command:

```
pr_shell> create_rail_macro_model \
        -si_ware_project_path project_path cell_name
```

The `project_path` keyword defines the path to where the compiling result is saved. By default, the `cell_name.log` file and the layer translation table file are stored in the `project_path/compout/views/cell_name` directory. The tool reads the layer information from the layer translation table file and finds the power and ground supply nets by the `pg_pin` keyword in the `cell.lib` file.

Example 1

The following example generates a rail macro model that contains a CONN view with evenly distributed currents and capacitances:

```
pr_shell> create_rail_macro_model \
        -si_ware_project_path {/home/project/ sram8x8}\
        -tech_file mw.tf
```


Example 2

The following example generates a rail macro model that contains a CONN view with weighted current and capacitance distributions:

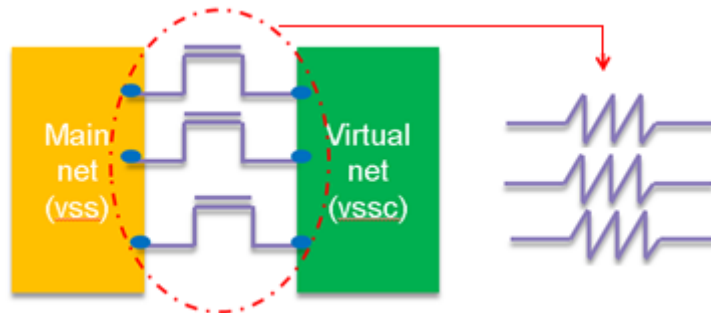
```
pr_shell> create_rail_macro_model
        -si_ware_project_path {/home/project/ sram8x8} \
        -tech_file mw.tf \
        -hspice_exec {/bin/hspice} \
        -spice_model_config {config.sp} \
        -spice_model_condition {ss 1.0 0.7 125}
```

Creating Macro Models With On-Resistances

When creating rail macro models for power-gating designs, PrimeRail also considers the behavior of the virtual nets that are in the power-gating design. Therefore, generating macro models for fine-grain memory cells requires the on-resistance information.

With the internal net information, the `create_rail_macro_model` command first extracts the resistance and capacitance network for the internal nets; the command then characterizes these switch cells to generate on-resistance values and ON/OFF states for the switch cells by using the device location information from the SPICE file. The calculated on-resistance is applied to the closest external and internal nets.

Figure 11-6 An Example of Creating a Macro Model With Internal On-Resistances



When the nets are at the ON state, the current distributes to both the external and internal nets; the on-resistance values connect to the closest external and internal nets. You can display the parasitic values for the nets in the parasitic and voltage drop maps.

When the nets are at the OFF state, the current distributes to the external nets only, meaning that the virtual net becomes floating. The internal net shape locations are covered by one big gray area in the parasitic or voltage drop map. The weighting values are different for the ON state and the OFF state.

Analysis Flow for Creating Macro Models With On-Resistances

To create a macro model with on-resistances,

1. Run the `create_rail_macro_model` command with the `-switch_supply_net_pairs` option. Specify other options as necessary.

For example,

```
pr_shell> create_rail_macro_model \
        -spice_top_netlist sram8x8.sp \
        -switch_supply_net_pairs {{vssc vss}} \
        -hspice_exec { /bin/hspice } \
        -spice_model_config { config.sp } \
        -spice_model_condition { ss 1.0 0.7 125 }
```

In this example, the names for the internal and external nets are `vssc` and `VSS` respectively. The switch cells in the design are characterized, and the generated model has different current distributions for the ON and OFF states.

If no SPICE netlist is specified, the information for the internal net is stored in the generated model but the switch cells remain uncharacterized.

2. Open the generated macro model in the PrimeRail GUI and check the attribute, the switch cell location, and the internal net shape location.
3. Check the ON and OFF states of the switch cell using the `is_mtcmos_switch_cell_on` attribute. When the value is `TRUE`, all on-state resistance values in the macro are connected to the external and internal nets. When the value is `FALSE`, all on-state resistances are disconnected.

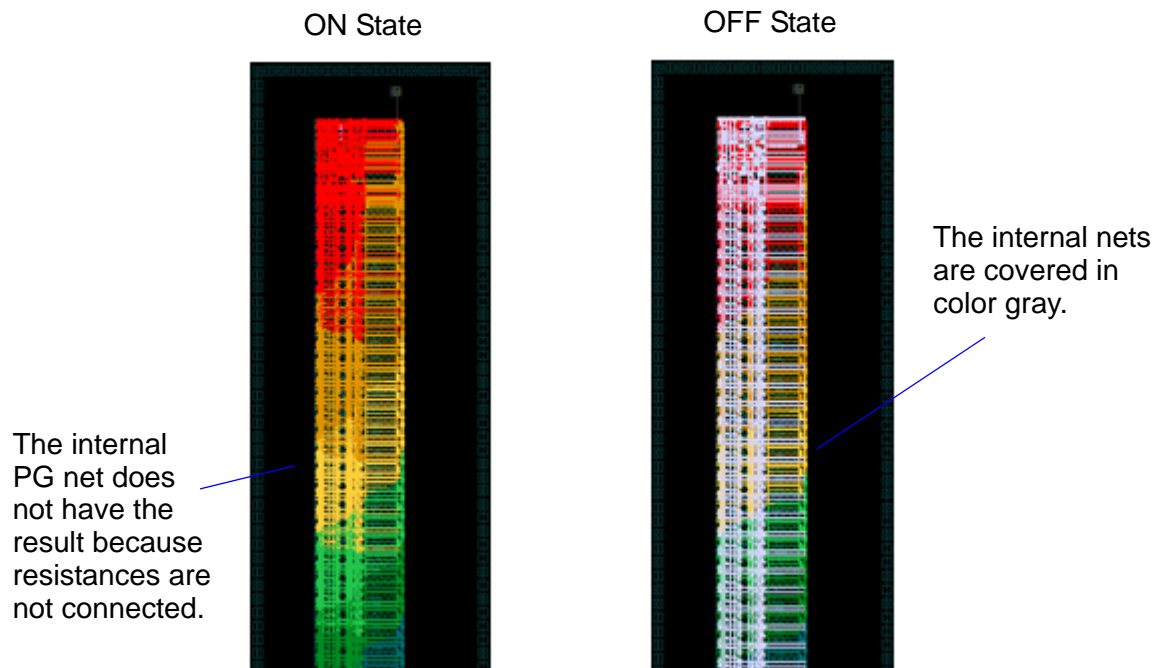
If the ON or OFF state of the switch cell is incorrect, use the

`set_rail_macro_model_cells -internal_net_state ON|OFF` command to reset the state. You must set the state of the hard macro before the `update_currents` and `extract_supply_parasitics` commands.

4. Run top-level rail analysis to examine the analysis results by switching between different states via the parasitic map or the voltage drop map (see [Figure 11-7](#)).

For more information about top-level rail analysis, see [Analyzing Top-Level Designs](#).

Figure 11-7 Checking the Generated Rail Macro Models in the Voltage Drop Map



See Also

- [Creating Rail Macro Models](#)
- [Power Management Cells](#)

Creating Simple Rail Macro Models

By default, the `create_rail_macro_model` command simulates all the child cells in the specified hard macro cell when generating macro models. To reduce data size and improve performance of the subsequent voltage drop analysis, you can specify the child cells or vias to exclude from the macro model generation using the options listed in [Table 11-1](#).

To create a simple rail macro model without manual settings, set the `-auto_exclude_bit_cells` option to `true`. For more information, see [Automated Simple Rail Macro Generation](#).

Note:

The `-auto_exclude_bit_cells` option supports only large memory designs.

Table 11-1 *The Options of the `create_rail_macro_model` Command to Exclude Cells*

Option name	Description
<code>-auto_exclude_bit_cells true false</code>	Automatically excludes bitcells during macro model generation. The default is <code>true</code> .
<code>-skip_conn_view_by_cell cell_names</code>	Specifies the child cells to exclude during macro model generation. By default, all child cells are included.
<code>-skip_via via_mask_names</code>	Specifies the vias that are not included in the macro model. By default, the command extracts the connectivity from via1 to via14 (metal1 to metal15).
<code>-skip_current_source_by_cell cell_names</code>	Specifies which child cells to be excluded from the current source generation. By default, all child cells are included.
<code>-stop_at_mask_layer mask_layer</code>	Stops extraction at the specified mask layer.

Automated Simple Rail Macro Generation

When analyzing a chip design with hard macros, the content of the rail macro models might have a great impact on the rail analysis results. To improve performance and capacity of the rail analysis with minimal loss of accuracy, you can create simple rail macro models by excluding bitcells. If you are not sure which bitcells to exclude, run the `create_rail_macro_model` command with the `-auto_exclude_bit_cells` option. The tool automatically creates simple rail macro models by identifying bitcells to be skipped based on the following criteria.

The tool skips a bitcell of a master cell if

- Its instance count is more than 1e5, and
- Its instances occupy more than 50% of the hard macro area

When multiple master cells meet both two criteria, the tool skips the one with the highest instantiation. The tool reports no bitcell is identified if the hard macro is not of the memory type or the size of the hard macro is not large enough.

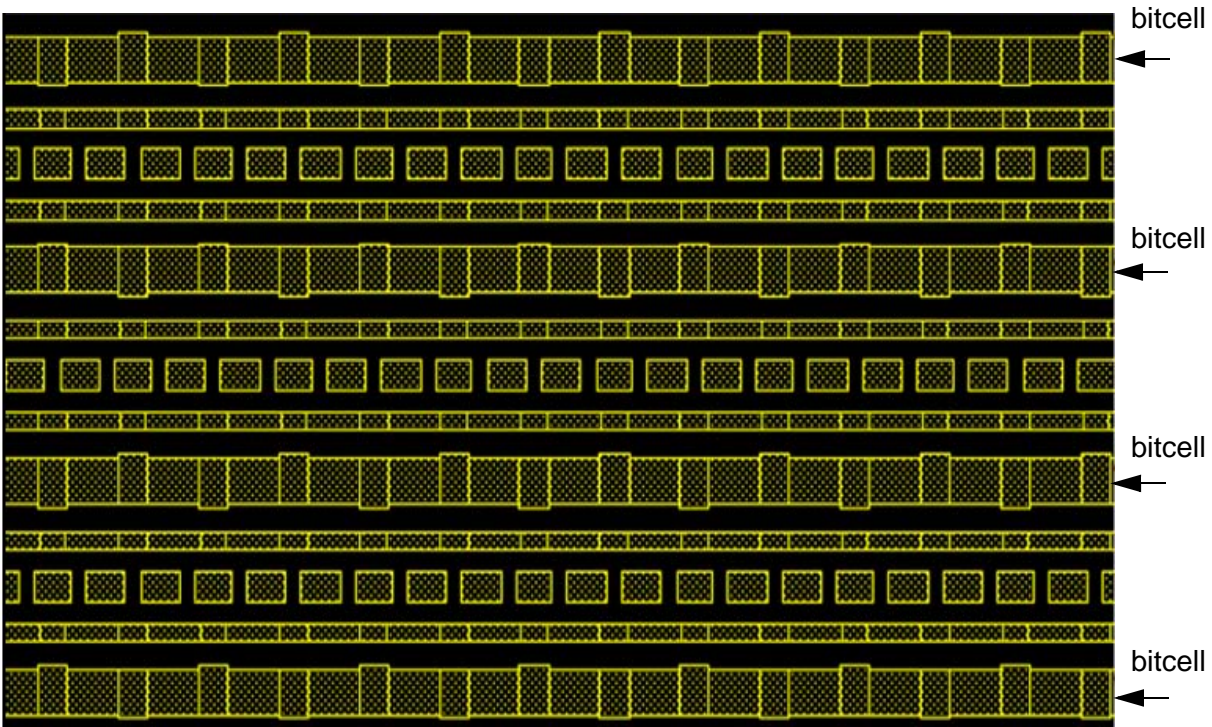
bitcells are good candidates to be skipped in rail macro model generation because:

- In design layouts, bitcells often appear as continuous segments with different heights. In [Figure 11-8](#), the arrows show cell rows with height differences. These differences result

in high node and edge counts in the extraction process and eventually affects the performance of the subsequent rail analysis.

- bitcells dissipate less current, compared with other memory components

Figure 11-8 Bitcells in the Design Layout



Example

The following example generates a macro model for the MyDesign cell in the MyLibrary library, using the VDD power net. During macro model generation, the tool skips via2 and via3, and the child cells SoftBlockA and SoftBlockB.

```
pr_shell> create_rail_macro_model -library MyLibrary -cell MyDesign \
        -power_nets VDD -skip_via {via2 via3} \
        -skip_conn_view_by_cell {SoftBlockA SoftBlockB}
```

See Also

- [Creating Rail Macro Models](#)

Updating Macro Models

When analyzing a macro cell design in the dynamic analysis flow, PrimeRail requires the rail macro models with both the physical and electrical information. To meet this requirement, you update the generated rail macro models either with SPICE or simulation data. By reusing the CONN view that was generated in the previous run, the `create_rail_macro_model` command regenerates the macro model by updating the electrical data, the distribution of the current source, and the intrinsic capacitance based on the specified SPICE or simulation information. The geometry information remains unchanged during the updating process.

To improve the accuracy of the generated rail macro models, update the rail macro models in either of the following ways:

- [Updating Rail Macro Models With SPICE Data](#)
- [Updating Rail Macro Models With Simulation Data](#)

Updating Rail Macro Models With SPICE Data

To update a generated rail macro model with SPICE information, prepare:

- A SPICE netlist file that specifies the device locations for the top design. A SPICE netlist is generated using any LVS tools, such as the IC Validator or Calibre tool.

The following example is a SPICE netlist file generated by the IC Validator tool. The \$X and \$Y keyword represent the coordinates of the devices, while the \$T keyword represents the transformation of the device. For more information about the SPICE netlists, see the documentation of the related LVS tool.

```
* ICV Netlist example
.subckt top
X1 n0 n1 n2 test $T=0 0 0 0 $X=0 $Y=0
X2 n3 n4 n5 test $T=70 0 0 90 $X=300 $Y=90
.ends
.subckt test
M0 a b c 1 nch $X=20 $Y=30 $D=0
.ends
```

- A SPICE configuration file that specifies the SPICE netlist model.

```
.LIB SS1
.lib 'device_model.sp' slow_ram.lib 'device_model.sp' slow_mos_cap
.lib 'device_model.sp' slow_res
.ENDL
.LIB FF1
...
.ENDL
```

- (Optional) A SPICE mapping file that specifies the mapping between layers in the geometry and the SPICE netlist. The supply voltages of each power and ground net in the SPICE file are also specified. When not specified, the tool assumes the geometry and SPICE layer names are the same. The supply voltage is specified with the corner voltage.

The syntax is:

```
net_type name_in_gds name_in_spice voltage_value
```

Example:

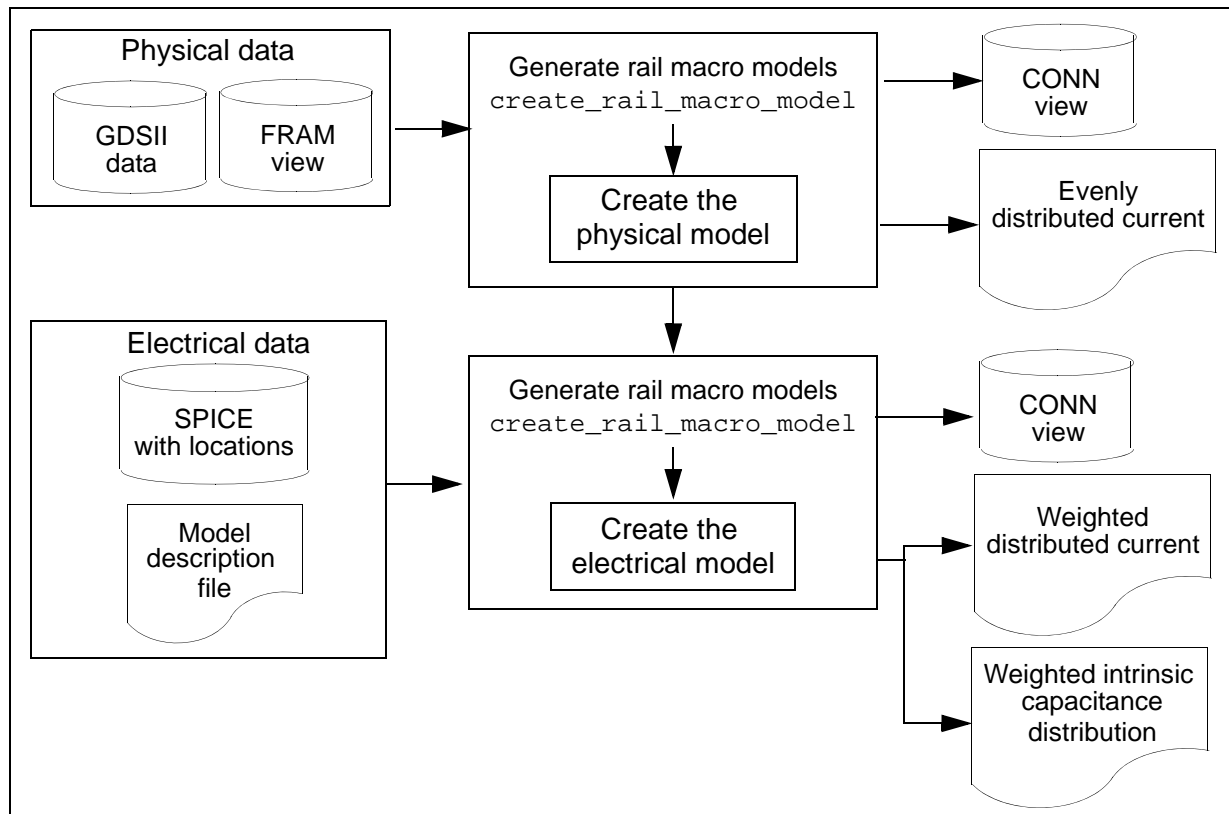
```
POWER VDD VDD 0.765  
GROUND VSS VSS 0
```

Note:

Creating rail macro models with SPICE netlists includes the process of the circuit simulation. You must have the license to run the simulation tool, such as HSPICE, and specify the path to the simulation tool before creating rail macro models.

As shown in [Figure 11-9](#), when updating a rail macro model with SPICE information, the command uses the SPICE netlist information to create the electrical models. Weighted current and intrinsic capacitance are distributed to the device locations. The generated macro model is saved in the CONN view.

Figure 11-9 The Data Flow for Updating Rail Macro Models With SPICE Information



The following example updates the existing rail macro model for the sram64x64 cell by modifying the SPICE information:

```
pr_shell> create_rail_macro_model -library test_lib \
    -cell sram64x64 \
    -hspice_exec /tools/hspice \
    -spice_top_netlist {sram64x64.sp} \
    -spice_model_config model_config.sp \
    -spice_model_condition { abc 1.0 1.2 25 } \
    -reuse_conn_view
```


[Table 11-2](#) lists the options of the `create_rail_macro_model` command for updating a rail macro model with SPICE input data.

Table 11-2 Options for Creating Hard Macro Models

Option name	Description
<code>-library library_name</code>	Specifies the name of the Milkyway library that contains the cell being analyzed.
<code>-cell cell_name</code>	Specifies the name of the cell to be created in the Milkyway library.
<code>-spice_top_netlist design_name top_design_name</code>	Specifies the SPICE file that contains the top design.
<code>-spice_model_config file_name</code>	Specifies the configuration file for running the SPICE simulation.
<code>-spice_model_condition {corner_name P V T}</code>	Specifies the characterization condition of the current and the capacitance weights. If the specified condition does not exist in the model configure file, the first model condition in the model configure file is used for characterization.
<code>-hspice_exec</code>	Specifies the path to the HSPICE binary.
<code>-spice_include_path {paths}</code>	(Optional) Specifies the include paths of the related SPICE netlist.
<code>-reuse_conn_view</code>	(Optional) Uses the CONN view that is generated in the previous run for updating the rail macro model

See Also

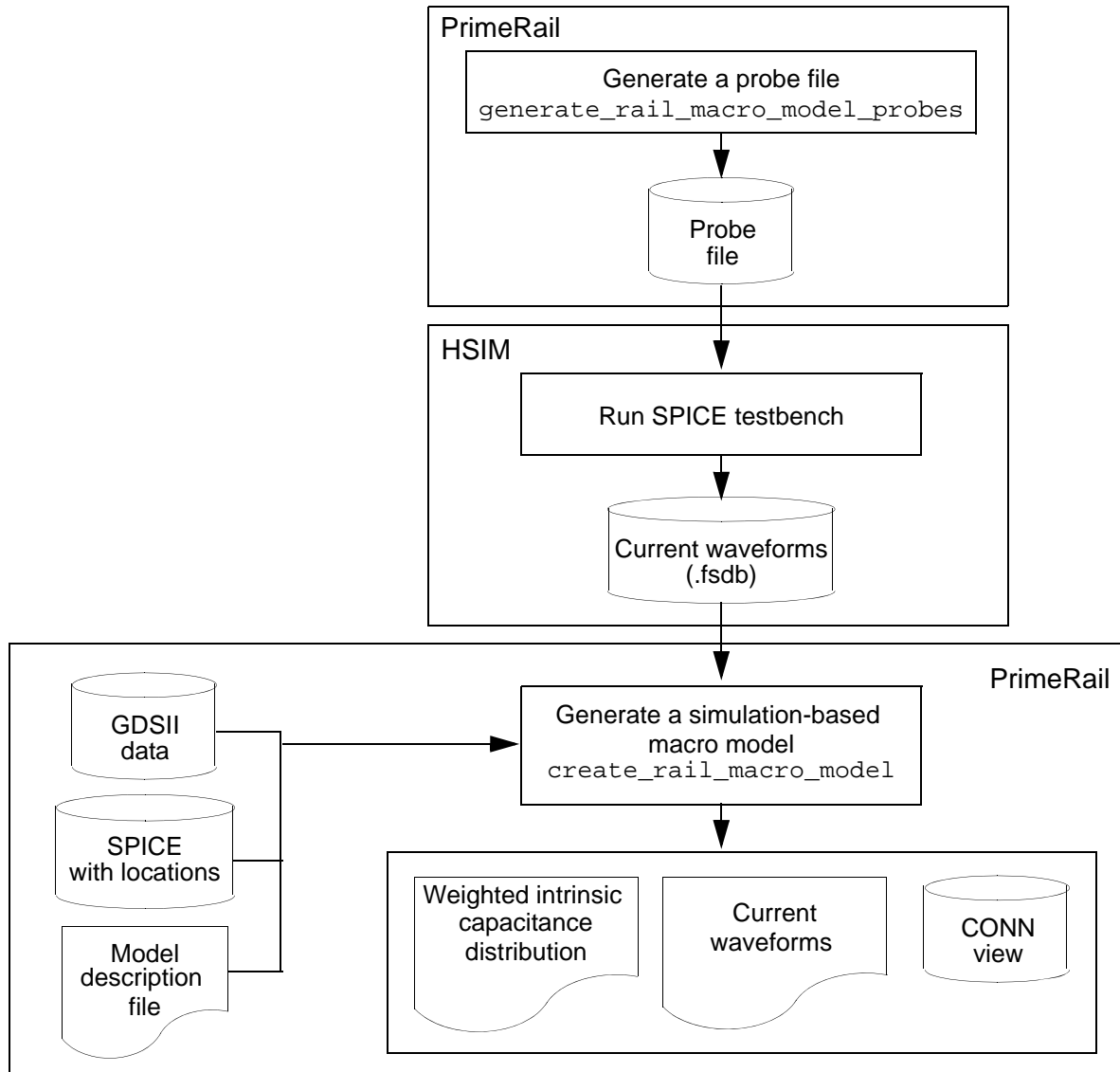
- [Creating Rail Macro Models](#)

Updating Rail Macro Models With Simulation Data

Current dissipation inside a hard macro might vary significantly with the status of the macro. To accurately calculate how current flows in the power and ground network, update an existing rail macro model with the circuit simulation results. The device current waveforms are then recorded in an FSDB file. The updated rail macro model contains a CONN view with current waveforms and weighted capacitances.

Figure 11-10 illustrates the analysis flow for calculating device current waveforms in a hard macro.

Figure 11-10 Steps for Updating a Macro Model With Simulation Data



Creating a simulation-based macro model consists of the following processes:

- **Node recording process:** In PrimeRail, run the `generate_rail_macro_model_probes` command to generate a probe file that records how the node status changes.
- **Simulation process:** In HSIM, run the HSPICE simulation to derive the current waveforms for all device terminals that are connected to the power supply network.

When the process is finished, an FSDB file is generated for recording device current waveforms.

- Rail macro model creation process: Using the GDSII data, the SPICE netlist, the SPICE simulation model, and the FSDB file, generate rail macro models by extracting current waveforms for each current source location in the macro cell.

Required Input Data

- The GDSII file that contains the physical layout information
- The SPICE netlist with the device location information

You can use LVS tools, like the IC Validator tool, to generate a SPICE netlist with device locations. The LVS scripts are provided by foundries.

Here is a SPICE netlist example with device locations. The \$X and \$Y keywords represent device coordinates. The \$T keyword represents the transformation of this device.

```
* ICV Netlist example
.subckt top
X1 n0 n1 n2 test $T=0 0 0 0 $X=0 $Y=0
X2 n3 n4 n5 test $T=70 0 0 90 $X=300 $Y=90
.ends
.subckt test
M0 a b c 1 nch $X=20 $Y=30 $D=0
.ends
```

- The SPICE package file that describes the device models along with condition configurations

The SPICE syntax, the .LIB and .ENDL keywords, are used to conclude all device models used in the specified macro cells.

Here is an example of the SPICE package file:

```
* Model configure file example
.LIB SSP
.lib 'device_model.sp' slow_sram
.lib 'device_model.sp' slow_macro_moscap
.lib 'device_model.sp' slow_res_disres
.ENDL
.LIB FFP
...
.ENDL
```

To capture the current waveform for each current source location and the intrinsic capacitance distribution in the power and ground network,

1. In PrimeRail, run the `generate_rail_macro_model_probes` command to generate a probe file that records how the node status changes.

Here is an example of the output probe file:

```
pr_shell> generate_rail_macro_model_probes -spice_netlist test.sp \
        -output mm.probes -power_nets {VDD} -ground_nets {VSS}
```

```
###mm.probe###
```

```
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M15)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M16)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M17)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M18)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M19)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M20)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M21)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M22)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M23)
.probe i(XI1.XI73.XI275.XI484.XI905.XI594.M24)
.probe i(XI1.XI73.XI275.XI485.XI904.XI594.M1)
```

2. In HSIM, add the following option to the SPICE testbench file:

```
.option xa_cmd = "set_waveform_option -format fsdb_4.3"
```

Next, add the testbench prefix to the probe file generated at Step 1, like

```
.probe i(x_testbench.XI1.XI73.XI275.XI484.XI905.XI594.M15)
.probe i(x_testbench.XI1.XI73.XI275.XI484.XI905.XI594.M16)
.probe i(x_testbench.XI1.XI73.XI275.XI484.XI905.XI594.M17)
.probe i(x_testbench.XI1.XI73.XI275.XI484.XI905.XI594.M18)
```

Now, run the testbench XA tool with the probe file (see the text in **bold** in the example) to calculate current waveforms in FSDB format.

Here is a syntax example:

```
* Testbench to simulate 64-time write+read operation
.param VH=0.765
.param VL=0
VDD VDD 0 VH
VSS VSS 0 VL
.global VDD VSS
.temp 125
.tran 10p 5n
.option MACMOD=1
...
* probing nodes from "generate_rail_macro_model_probes"
.inc mm.probes
...
.end
```

3. In PrimeRail, run the `create_rail_macro_model` command to generate a simulation-based rail macro model with the FSDB probe file generated at Step 2.

Here is an example:

```
pr_shell> create_rail_macro_model \
    -library mw_lib \
    -cell test \
    -reuse_conn_view \
    -power_nets { VDD } \
    -ground_nets { VSS } \
    -mode_configure mode_cfg \
    -spice_top_netlist test.sp \
    -spice_model_config spice/config.sp \
    -spice_model_condition { abc 1.0 0.765 25.0 } \
    -hspice_exec { /test/hspice/bin/hspice }
```

The `mode_configuration_file` keyword specifies the file that describes the necessary configuration for signal net settings as listed in [Table 11-3](#) and mode settings in [Table 11-4](#).

Table 11-3 Options for Setting Signal Configurations

Argument	Description
signal:signal_name	Specifies the name of the signals
dir:direction	Specifies the direction of the signals

Table 11-4 Options for Setting Modes

Argument	Description
name:mode_name	Specifies the mode name.
waveform_source:file_name	Specifies the waveform file that contains current waveforms in the FSDB format.
pg_pin:pin_name	Specifies the names of the power and ground pins.
related_pin:pin_name	Specifies the name of the pin that triggers the status change. This is used in rail analysis.
start_time:time	Specifies the time when the current waveform is to be captured.
end_time:time	Specifies time when the current waveform stops being captured.

Table 11-4 Options for Setting Modes (Continued)

Argument	Description
<code>strip:"stripped_prefix"</code>	Specifies the path prefix to be removed from the names of the objects in the <code>waveform_source:file_name</code> file.

The following is an example of the mode configuration file used for generating a simulation-based rail macro model:

```

### Comment# and concatenated line via '\' is supported
### No space is allowed in argument assignment
### Ex. "name:VDD" is GOOD, but "name: VDD" will cause problem/error

### Multiple lines concatenated by '\' is supported
signal:MEA dir:i
signal:WEA dir:i
signal:RMEA dir:i
signal:LS2 dir:i
signal:CLKA dir:i
signal:MEB dir:i
signal:RMEB dir:i
signal:LS dir:i
signal:CLKB dir:i

### "Multiple modes but different time periods inside one FSDB" is
### supported
name:W1 \
start_time:1125 end_time:1625 \
waveform_source:1.fsdb pg_pin:VDD \
when:"A&B" related_pin:"CLK1"\
name:W1 start_time:1125 end_time:1625 \
waveform_source:1.fsdb pg_pin:VSS \
when:"A&B" related_pin:"CLK1"

name:W2 start_time:4125 end_time:4625 \
waveform_source:1.fsdb \
pg_pins:VDD \
strip:"TEST_STRIP/" \
when:"C&D" related_pin:"CLK2"

### "Multiple FSDB files for different modes" is supported
name:R1
start_time:2375 end_time:2875 \
waveform_source:2.fsdb pg_pin:VSS \
when:"B&C" \
related_pin:"CK" \

name:R2 \
start_time:3375 end_time:3875 \
waveform_source:3.fsdb pg_pin:VSS \

```

```

when:"B&C" related_pin:"CK"

### Any duplicate name/pg_pin combination is illegal and not allowed.
### Later one will be ignored
name:R3 \
start_time:2375 end_time:2875 \
waveform_source:4.fsdb \
pg_pin:VSS \
when:"B&C" \
related_pin:"CK"

name:R3 \
start_time:3375 end_time:3875 \
waveform_source:5.fsdb \
pg_pin:VSS \
when:"B&C" \
related_pin:"CK"

```

During the generation process, PrimeRail prints the node mapping summary in the log file, including the mapping rate and the mismatched node list. The rate of back-annotating the FSDb file to the SPICE netlist decides the quality of the generated macro model. The higher the mapping rate, the more accurate the model is generated. The mismatched nodes are also listed following the hierarchy in the SPICE netlist.

Here is an example of the log message:

```

INFO: Processing FSDb file "testbench_64_WR_only_1_/test.fsdb"...
=====
Mapping Rate Summary
=====
Mapping Rate: 0.00%
Mode
-----
W0
W1
R0
R1
-----
Mis-matched Node(s)
-----
XI1.X8939
XI1.X8941
XI1.X8943
XI1.X8945
XI1.X8947
XI1.X8949
XI1.X8951

```

See Also

- [Creating Rail Macro Models](#)

Setting Mode Sequences for Macro Models

Use the `set_rail_macro_mode_sequence` command to specify mode sequence related options for analyzing dynamic current and voltage effects of the specified macro models. The options include the pair of time and operation mode, the associated clocks, and the user-specified mode sequence file for both vector-based and vector free rail analysis. You can also specify the operation modes for vector free rail analysis only.

When setting is complete, run the `update_currents` and `calculate_rail_voltage` commands to calculate the current waveforms and voltage drops for the macro models based on the specified model sequence settings.

During vector-based rail analysis, PrimeRail applies the corresponding current waveforms from the mode at the specified time. These time points are triggering clock edges to the macro model.

During vector free rail analysis, you can specify the number of clock cycles that do not need to match the length of mode sequence. If a mode sequence is shorter than the number of cycles, PrimeRail applies the sequence to the subsequent triggering clock edges recursively. For example, if the mode sequence is defined as “read read write read write” and the number of clock cycles is 10, the mode sequence is “read read write read write read read write read write” for 10 clock cycles. If the clock cycle number is 2, the mode sequence is “read read.”

Different macro models might have different triggering clock edges. By default, the macro models are only triggered by their own clock edges. For example, if the clock triggering a macro model has zero switching activity on its source pin, the clock will be inactive during vector free rail analysis; the macro model that is associated with that clock will not be triggered, even if you have specified a mode sequence for it. To activate the mode sequence for a macro model, regardless of the switching activity, use the `-clock` option of the `set_rail_macro_mode_sequence` command to specify a clock that triggers the macro model.

For more information about how to set mode sequence for macro models with the `set_rail_macro_mode_sequence` command, see the command man pages.

Checking the Macro Model Content

To check the information inside a macro model, run the `report_rail_macro_model` command. The report includes the information about cell master names, the number of external and internal PG pins, electrical models, macro sources and their locations, as well as current distribution data. For each cell master, the report also lists all the cell instances associated with this cell master.

Here are the commonly used options when checking the macro model content with the `report_rail_macro_model` command.

Table 11-5 *Frequently Used Options With the `report_rail_macro_model` Command*

Option	Description
<code>-pin_detail</code>	Reports detailed information about the internal pins of the specified hard macro, such as pin weighting and location. No weighting value is reported if the port is evenly distributed.
<code>-dump_mode_waveform</code>	Writes the mode-based current waveforms of the specified hard macro into a waveform file.
<code>-waveform_filename</code>	Saves the output total PG or internal-pin waveforms in the Fast Signal Database (FSDB) or ASCII format.

Example 11-1 *Examples of Running the `report_rail_macro_model` Command*

```
pr_shell> report_rail_macro_model -library ram64x8 -cell ram64x8

*****
* Hard macro info
Macro cell name: ram64x8
Number of ports: 2
Physical model: CONN
Electrical model: DWM-lite
Model pvt: process = 1, voltage = 0.86V, temperature = 25
* Hard macro port info:
Port name: VSS
Port number: 0
Number of pins: 74375
* Port electrical model data
Port power type: Mode-based power
Port power distribution: Weighted
Mode CS&write average -0.000136A/max -0.002A/duration 7.36e-09s
Mode CS&!write average -0.000136A/max -0.002A/duration 7.36e-09s
* Hard macro port info:
Port name: VDD
Port number: 1
Number of pins: 74375
* Port electrical model data
Port power type: Mode-based power
Port power distribution: Weighted
Mode CS&write average 0.000136A/max 0.002A/duration 7.36e-09s
Mode CS&!write average 0.000136A/max 0.002A/duration 7.36e-09s

pr_shell> report_rail_macro_model -library ram64x8 -cell ram64x8 \
      -pin_detail

*****
```

Report : Hard Macro
 Design: ram64x8
 Version: <Version>
 Date : <Date>

=====

Hard Macro ram64x8 Report

=====

Hard Macro Model :

Macro Name	Ports
ram64x8	2

Model Operating Condition :

Process	Voltage(V)	Temperature(Celsius)
1.000	0.800	25

Model Database Information

Milkyway Library : ram64x8

Port VSS Content :

Port Name	Number of Pins
VSS	16889

Electrical Model Data :

Power Type	Distribution	Total Capacitance
Mode-based power Weighted		12.58

Pin Weight List :

Pin ID	Weight	Pin Capacitance
0	3.50584e-09	0.00074486
1	3.50584e-09	0.00074486
.....		
24400	1.67952e-09	0.00051555

Mode Current Summary :

Mode Name	Average(A)	Max(A)	Duration(s)
CLKB_read	-0.000156	-0.0297	1.42e-09
CLKA_WRITE	-0.00418	-0.02	1.15e-10

See Also

- [Creating Rail Macro Models](#)

Reporting Unmodeled Macro Cells

A top design usually contains a great number of memory or macro cells. To check if the top design has any macro cells that are not modeled, run the `report_rail_macro_model` command with the `-unmodeled_macro_list` option. The command reports the unmodeled or corrupted macro cells in the log message.

Here is an example of the report:

```
pr_shell> report_rail_macro_model -unmodeled_macro_list

=====
Unmodeled Hard Macro List
=====
*****
ram64x8
*****
```

See Also

- [Creating Rail Macro Models](#)

Displaying Current Source Information

You can check the generated current sources in GUI by selecting Object Window > Rail Analysis > Macro Current Source. To display the current source information inside a rail macro model, run the `get_macro_current_sources` command and specify the criteria. The command supports the top-level CONN views only.

The following example queries the macro current source locations within the specified area:

```
prompt> get_macro_current_sources -within {{102 15} {103 16}}
```

The following example queries the macro current source locations that have the specified layer number:

```
prompt> get_macro_current_sources -filter {layer_number==15}
```

The following example finds the macro current source locations whose `current_weight` is less than 0.004:

```
prompt> get_macro_current_sources -filter {current_weight < 0.004}
```

Use this command either at the command prompt or as an argument of another command, such as the `query_objects` command. In addition, you can assign the result to a variable. To control the number of the objects to be reported, set the `collection_result_display_limit` variable before executing the

`get_macro_current_sources` command. By default, the maximum of the reported objects is 100.

For information about collections and querying objects, see the `collections` man page.

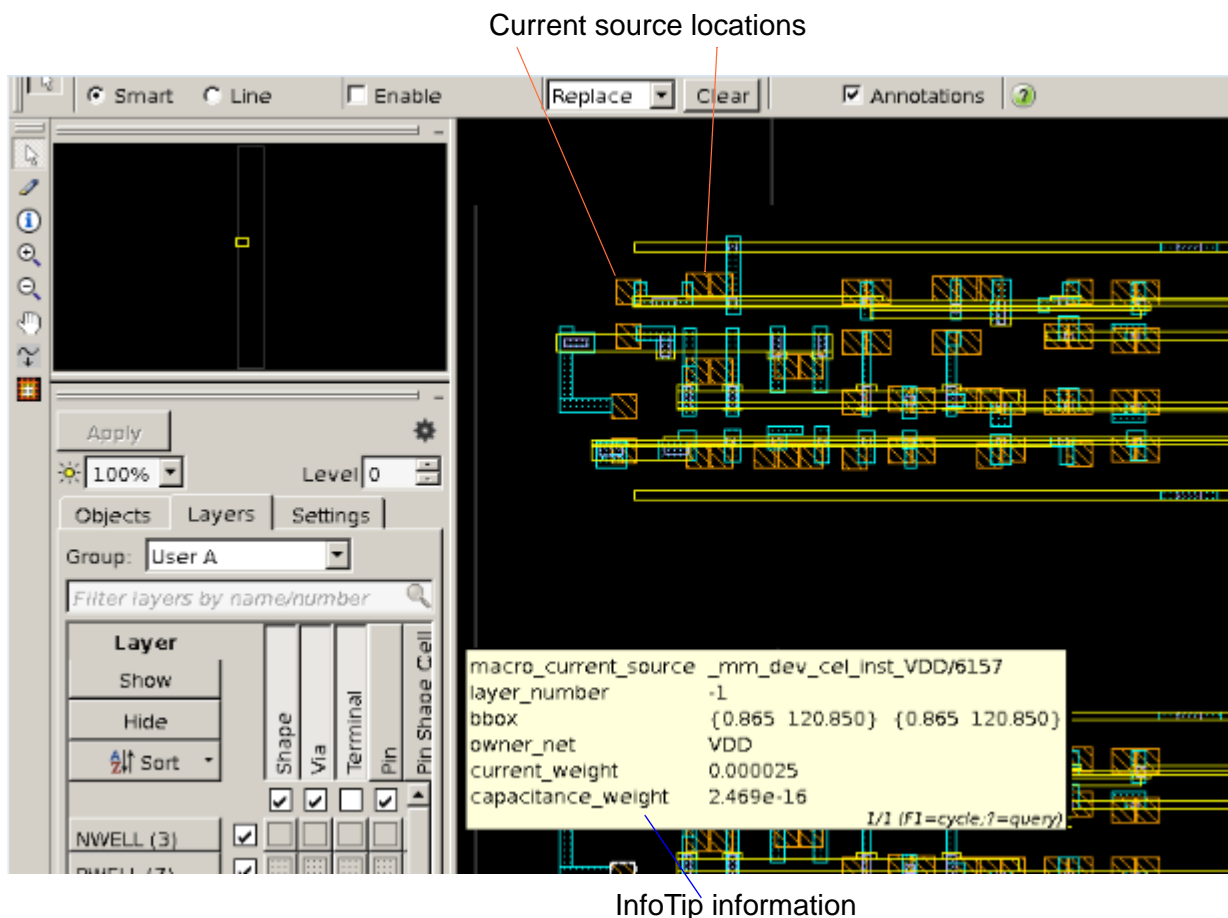
Note:

The command displays the requested macro current source information in the log window. To save information to an output list, use the `report_rail_macro_model -pin_detail` command.

Viewing Macro Models in the GUI

After you generate a macro model with the `create_rail_macro_model` command, you can open and examine it in the GUI with the `open_mw_cel -library designName.CONN` command. You can then query physical shapes or current source objects of the macro model in the GUI, as shown in [Figure 11-11](#).

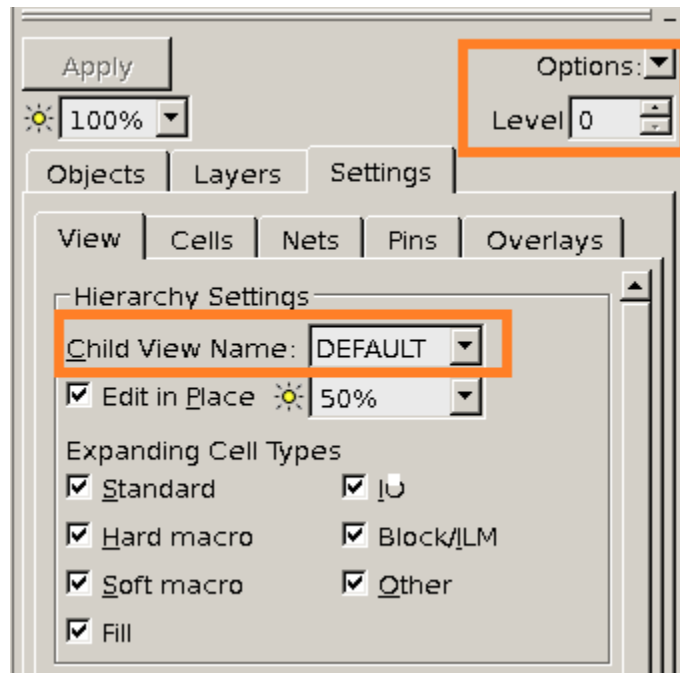
Figure 11-11 Querying Physical Shapes in the Macro Model



Displaying Macro Models as Child Cells

By default, PrimeRail does not display a macro model in the GUI if the CONN view is loaded as a child cell. To display it in the GUI, you have to change the hierarchy level and the name of the child view, as shown in [Figure 11-12](#).

Figure 11-12 Modifying the Hierarchical Level and Name of a Child View



Changing Displayed Views

By default, PrimeRail displays the FRAM view of the design in the GUI. To switch between FRAM view and CEL view, use the `change_macro_view` command.

For example, to change what is displayed in the GUI layout window from FRAM view to CEL view,

```
pr_shell> change_macro_view -reference sram1056x12 -view CEL
```

See Also

- [Creating Rail Macro Models](#)

12

Power Management Cells

A power management (PM) cell acts like a switch in the design with an internal resistive network. It takes one power or ground net as an input and provides controlled connection to an output power or ground net in a different name. The input net obtains power and ground net from a pad cell while the output net provides this power and ground net to the core region or another specific portion of the full-chip design.

You can define the resistance of the connection between the input net and the output net based on the specific power management cell being used. This resistance is virtual and there is no actual geometry involved with creating this resistance.

This chapter consists of the following sections:

- [Analyzing Power Management Cells](#)
- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)
- [Rail Analysis With Hard Macro Power Management Cells](#)
- [What-If Analysis With Switch Cells](#)
- [Inrush Current Analysis](#)

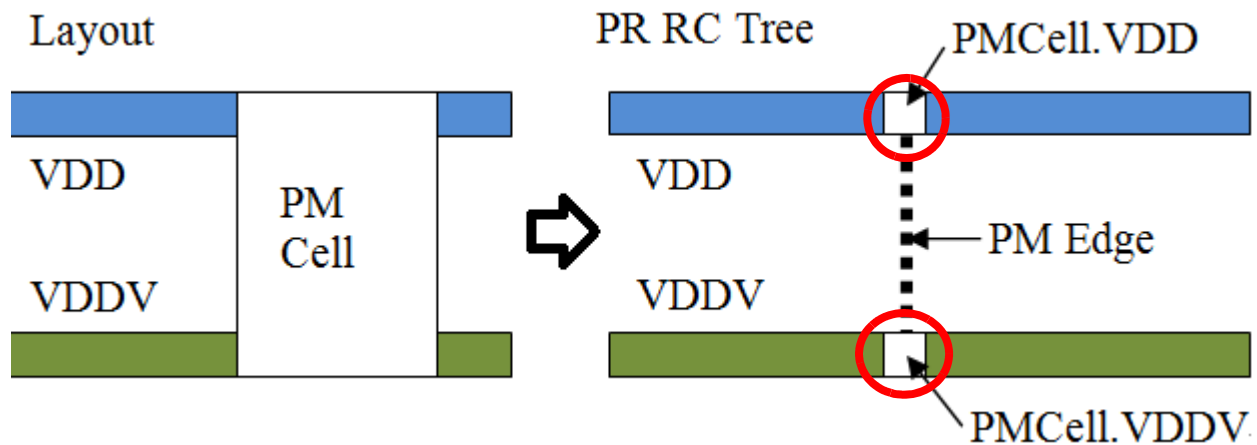
Analyzing Power Management Cells

PrimeRail supports two types of power management cells: multithreshold-CMOS switch cells (also called power gating cells) that are of cell type standard and hard macro power management cells that are of cell type macro.

- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)

When analyzing a multithreshold-CMOS switch cell, PrimeRail models a standard cell by placing a resistor edge in the power management cell and the internal circuit representation by connecting the port instance on the main power net (VDD) to the virtual power net (VDDV). PrimeRail assigns the power management resistor edge with an on-resistance value and leakage current and ignores the internal connections, as shown in [Figure 12-1](#).

Figure 12-1 Analyzing A Multithreshold-CMOS Switch Cell

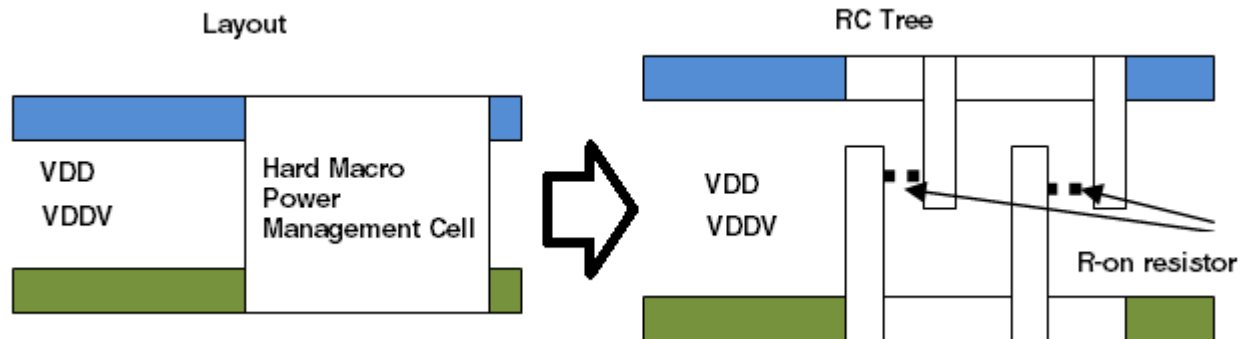


- [Rail Analysis With Hard Macro Power Management Cells](#)

However, when analyzing a hard macro power management cell, PrimeRail assumes it is a hard macro that connects to the main and virtual power and ground nets through multiple multithreshold-CMOS sources and drains. Based on the CONN views and the power management location (PML) file, PrimeRail inserts a resistor to connect between the source and drain ports of the transistors. PrimeRail models each power management transistor as a resistor with an assigned on-resistance value in the PML file, as shown in [Figure 12-2](#).

Use the hard macro power management cell flow if you want to consider the interconnect shapes during rail analysis.

Figure 12-2 Analyzing a Switch Cell



Rail Analysis With Multithreshold-CMOS Switch Cells

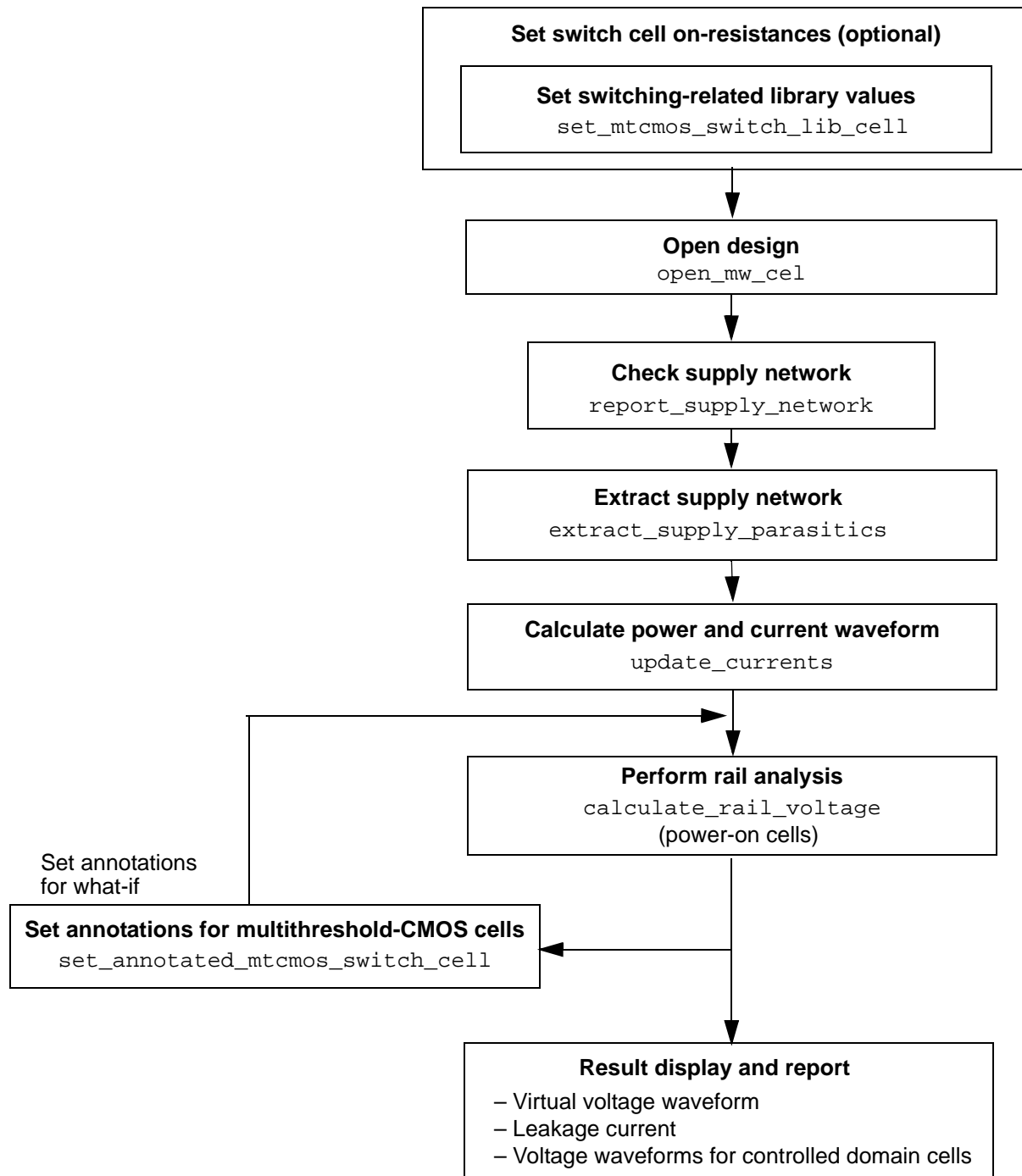
Before running static analysis with multithreshold-CMOS switch cells, PrimeRail needs the linear resistance data for the switch cells. You can define on-resistance data for switch cells without knowing any implementation details by using `set_mtcmos_switch_lib_cell` command or PrimeRail reads the library information from the .db file (see [Setting On-State Resistances](#)).

You can also set annotations for the specified switch-cell instances to experiment with different power management strategies without changing the design (see [Annotating Switch-Cell Instances](#)).

When rail analysis is complete, you can optimize voltage drop and electromigration effects by manually changing the on-resistance values that are associated with each switch transistor (see [What-If Analysis With Switch Cells](#)).

[Figure 12-3](#) describes the design flow of the power-on analysis in the static rail analysis flow.

Figure 12-3 Multithreshold-CMOS Switch Cell Design Flow



Analyzing voltage drops with switch cells consists of the following steps:

1. Set on-state resistance data for switching cells by reading on-resistances from the library or running the `set_mtcmos_switch_lib_cell` command.

For more information, see [Setting On-State Resistances](#).

2. Set annotations on the switching cells with the `set_annotated_mtcmos_switch_cell` command.

For more information, see [Annotating Switch-Cell Instances](#).

3. Run the `report_supply_network` command to check the connectivity between supply nets and their corresponding multithreshold-CMOS switch cells.

For more information, see [Checking Supply Networks](#).

4. When switch cells are modeled correctly, run static power analysis and generate static current waveforms using the `update_currents` command. For static power analysis, the activity of the switch cell control signals can be specified in the format of VCD, SAIF, and so on.

For details about calculating power current waveforms, see [Analyzing Power and Generating Current Waveforms](#).

5. Run the `extract_supply_parasitics` command to extract the resistance and capacitance data for the supply nets and virtual power and ground nets of the switch cell.

For more information, see [Extracting Supply Net Parasitics](#).

6. Run the `calculate_rail_voltage` command to analyze the voltage drop and current violations at the full-chip level.

For more information, see [Static Rail Analysis](#).

Setting On-State Resistances

Running static rail analysis in the power-on mode needs the on-state resistance data.

Specify on-state resistances in one of the following ways:

- Reading on-resistances from the library

If the CCS power or NLPM libraries already have IV curves for switch cells, PrimeRail automatically determines the on-state resistance values from the IV curve tables.

The following example describes an IV curve table where the on-resistance values can be derived.

```
dc_current ( ivt1 ) { \
    related_switch_pin : SLEEP; /* control pin */ \
    related_pg_pin : VDD; /* source */ \
```

```

related_internal_pg_pin : VVDD; /* drain */
values( "0.010, 0.020, 0.030, 0.030, 0.030", \
        "0.011, 0.021, 0.031, 0.041, 0.051", \
        "0.012, 0.022, 0.032, 0.042, 0.052", \
        "0.013, 0.023, 0.033, 0.043, 0.053", \
        "0.014, 0.024, 0.034, 0.044, 0.054"); \
}

```

- Setting on-resistances on Switching cell references

For NLPM libraries and CCS power libraries without IV curve data for the switch references, you can specify on-state resistances explicitly using the `set_mtcmos_switch_lib_cell` command in the `.synopsys_pr_setup` file. For example,

```

pr_shell> 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

```

Use the `set_mtcmos_switch_lib_cell` command to set switching-related library values for switch cell references, including on-state resistance, off-state leakage, or pin state name for a switch cell reference which might exist in the library. PrimeRail uses the on-state resistance, off-state leakage, and pin state information during rail analysis.

To report switching-related library information of switch-cell instances, run the `report_mtcmos_switch_lib_cell` command.

Note:

Note that if you execute the `set_mtcmos_switch_lib_cell` command before the `open_mw_cel` command, PrimeRail creates a new switch cell in the library. By default, PrimeRail saves all the switch cell references created from the command in a specific library, called `rail_sidefile_lib_0`. If the specified switch cells or cell masters already exist in the Liberty library, executing the command overwrites the current on-state resistance value or the off-state leakage current value and creates a new pin state with the values you specify.

If you execute the `set_mtcmos_switch_lib_cell` command after the `open_mw_cel` command, you can only assign the pin state name for an existing pin state. You cannot create a new switch library cell or cell master, or set a new pin state after the `open_mw_cel` command is run.

Example 1

Assume the switch cell reference or master, named `HEADER`, does not exist in the library. The following example creates a new switch cell reference, called `HEADER`, assigns an on-state resistance value 100.0 to a pin state, relates this value to the PG pins, and finally names the pin state as `low_res`. Then run the `open_mw_cel` and `get_attribute` commands to get the on-state resistance attribute.

```
pr_shell> set_mtcmos_switch_lib_cell HEADER -pin_state {sw 0} \
-on_state_resistance 100.0 \
-input_pg_pin VDD -output_pg_pin VVDD \
-pin_state_name low_res
pr_shell> report_mtcmos_switch_lib_cell
```

Example 2

Assume a switch cell, called FOOTER, has been already defined in the library and an on-state resistance is defined as 0.52 for pin state {sw1 1 sw2 0}. However, the pin state has not yet been named. The following example creates an on-state resistance for another pin state defined with {sw1 0 sw2 1} and names the pin state as high_res:

```
pr_shell> set_mtcmos_switch_lib_cell HEADER \
-input_pg_pin VDD -output_pg_pin VVDD \
-pin_state {SLEEPIN 0} -on_state_resistance 1.0 \
-pin_state_name ON
pr_shell> set_mtcmos_switch_lib_cell HEADER \
-input_pg_pin VDD -output_pg_pin VVDD \
-pin_state {SLEEPIN 1} -off_state_leakage 0.1 \
-pin_state_name OFF
pr_shell> report_mtcmos_switch_lib_cell HEADER
```

```
*****
Report : report_mtcmos_switch_lib_cell
Design : sample_design
Version: D-2010.06
Date   : Fri Feb  5 17:17:46 2010
*****
```

Library cell : 'HEADER'

Pin State	Pin State Name	Input PG Pin	Output PG Pin	On State Resistance	Off state Leakage
SLEEPIN 0	ON	VDD	VVDD	1.0	-
SLEEPIN 1	OFF	VDD	VVDD	-	0.1

Example 3

Continuing from Example 2, open the cell with the `open_mw_cel` command and rename the pin state name as `low_res` for an existing pin state {sw1 1 sw2 0}.

Note that if you execute the `set_mtcmos_switch_lib_cell` command before the `open_mw_cel` command, PrimeRail creates a new switch cell in the library. By default, PrimeRail saves all the switch cell references created from the command in a specific library, called `rail_sidefile_lib_0`. If the specified switch cells or cell masters already exist in the Liberty library, executing the command overwrites the current on-state resistance value or the off-state leakage current value and creates a new pin state with the values you specify.

If you execute the `set_mtcmos_switch_lib_cell` command after the `open_mw_cel` command, you can only assign the pin state name for an existing pin state. You cannot create a new switch library cell or cell master, or set a new pin state after the `open_mw_cel` command is run.

See Also

- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)

Annotating Switch-Cell Instances

Use the `set_annotated_mtcmos_switch_cell` command to set an annotation for the specified switch-cell instances. You can set three types of annotations: on-state resistance, off-state leakage, and state. PrimeRail uses these settings during rail analysis.

By default, all the states of the switch-cell instances are considered to be on for static rail analysis. When you specify the `-state` option, the command checks whether the state is consistent with the analysis result. The command issues an error message if a switch cell is set to off during timing or power analysis, but on during rail analysis.

Setting annotations for switch-cell instances does not change the design. You can run the `set_annotated_mtcmos_switch_cell` command multiple times for what-if analysis to experiment with different power management strategies if necessary.

The annotation has the following limitations:

- If an on-state resistance value is not present in the library, the command assumes that the switch cell is unable to connect two power and ground networks. On the other hand, if an off-state leakage is not present in the library, the default is set to 0.0.
- The on-state resistance is represented as a `cell` or a `lib_cell` attribute, called `on_state_resistance`. The off-state resistance is represented as a `cell` or a `lib_cell` attribute, called `off_state_leakage`.

To get a listing of object attributes, use the `list_attributes -application -class cell` command or the `list_attributes -application -class lib_cell` command.

- When no library information is available, run the `set_mtcmos_switch_lib_cell` command instead. The command can only add an annotation to a cell, not a library cell.
- The `pin_state` attribute specifies a state where the switch cell is fully turned on (in cases when a switch is inside the switch cell), partially turned on (in cases when multiple switches are inside the switch cell), or fully turned off (when all internal switches inside the switch are set to off by the pin state).

The tool also checks to see whether the annotated value has any conflict with the pin state. A conflict occurs if an on-state resistance or the `state_on` attribute is annotated to

a pin state when the cell is actually off, as defined in the library definition, or vice versa. If there is no conflict, the command sets the annotation to the specified pin state.

Use annotations in a what-if flow in which the resistance values, the leakage values, or the on- and off-states can be adjusted to make voltage drop tradeoffs. This allows you to simulate switch cell sizing operations without changing the cell references and the design, and then quickly iterating with static rail analysis to assess the tradeoff results.

Note that

- When neither the `-pin_state` nor the `-pin_state_name` option is specified, the on or off state is applied to all pin states where the cell can be turned on or off, respectively.
- When only the option setting of `-state on` is used, the pin state which has the smallest on-state resistance becomes the current pin state.
- When only the `-state off` option is used, the pin state which has the smallest off state leakage becomes the current pin state.
- When either the `-pin_state` or `-pin_state_name` option is specified, the on or off state is applied to the specified pin state, and the specified pin state becomes the current pin state. With the pin state being specified, you cannot change the state of the cell to on or off.

Example 1

A switch cell has been defined in the library, where the cell is on if the control pin `sw` that has an input of a logic 0. The following example sets the user-annotated on-state resistance to the cell such that the user-annotated resistance value is then used during rail analysis.

```
pr_shell> set_annotated_mtcmos_switch_cell \
          -on_state_resistance 100.0 -pin_state {sw 0} \
          [get_cell -filter is_mtcmos_switch_cell]
```

Example 2

The following command turns on the cell without specifying a pin state or pin state name:

```
pr_shell> set_annotated_mtcmos_switch_cell -state on \
          [get_cell -filter is_mtcmos_switch_cell]
```

Example 3

The following command sets the current pin state to be “sw1 1 sw2 0,” which means only one switch transistor inside the cell is turned on:

```
pr_shell> set_annotated_mtcmos_switch_cell -state on \
          -pin_state {sw1 1 sw2 0} \
          [get_cell -filter is_mtcmos_switch_cell]
```

Deleting Annotations

Use the `remove_annotated_mtcmos_switch_cell` command to remove the user annotation from the switch cell and to revert the `on_state_resistance` and `off_state_leakage` parameters to the default. The user annotation was previously applied with the `set_annotated_mtcmos_switch_cell` command.

Specify the switch cells whose annotations are to be removed. Use the `-on_state_resistance` and `-off_state_leakage` options to indicate which on-state resistance or off-state leakage is to be removed. Enable the `-pin_state` *pin_state* option and specify the pin states whose annotation settings are to be removed from the switch cell. If the switch cell has more pins than those specified in the pin state list, state X (unknown) will be assigned to the pins whose states are not set in the list. The command removes annotations from all on-states of the switch cell in the library if you do not specify the `-pin_state` option.

Use the `-pin_state_name` *ps_name* option to specify a pin state name for which the annotated value is to be removed.

When the applied `on_state_resistance` is removed, the `on_state_resistance` reverts back to the default on-state resistance for the switch cell object. If the characterization data of the library is not available, the `on_state_resistance` value is set to infinite, which means the switch cell is turned off.

When the applied `off_state_leakage` is removed, the `off_state_leakage` reverts back to the default off-state leakage for the switch cell object. If the characterization data of the library is not available, the `off_state_leakage` value is set to 0.

Example

The following example shows how to retrieve the resistance values that are specified using the `set_annotated_mtcmos_switch_cell` command:

```
pr_shell> get_attribute [get_cell -filter is_mtcmos_switch_cell] \
               on_state_resistance
{on low "i 1" vdd vdd 0.52 10.0 }
```

See Also

- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)

Checking Supply Networks

Before calculating power and current waveforms, run the `report_supply_network` command to check the connectivity between supply nets and their corresponding multithreshold-CMOS switch cells, as well as the relationship between supply nets through multithreshold-CMOS switch cells.

In PrimeRail, supply nets are related if one or more multithreshold-CMOS switch cells directly connect to both nets. For instance, in a multi power domain design, a supply net VDD might be related to several supply nets by providing current paths.

By default, the `report_supply_network` command reports which supply nets are directly related to the specified supply net via the multithreshold-CMOS switch cell objects. To report which multithreshold-CMOS switch cells connect to the nets and display their current on/off status, use the `-verbose` option. For more information about how to specify the on/off status, see the `set_annotated_mtcmos_switch_cell` command man page.

If no `supply_nets` parameter is specified, the command reports on all the supply nets.

Example 12-1 Reporting supply network relationships for the supply net VDD

```
pr_shell> report_supply_network VDD

*****
Report : report_supply_network
Design : sample_design
Version: <Version>
Date   : <Date>
*****
Supply Net : VDD
-----
Connected Supply Net : VDD1
Connected Supply Net : VDD2
```

Example 12-2 Creating a verbose supply network report

```
pr_shell> report_supply_network -verbose VDD

*****
Report : report_supply_network
Design : sample_design
Version: <Version>
Date   : <Date>
*****
Supply Net : VDD
-----
Connected Supply Net : VDD1
  MTCMOS Switch Cell : switch_class_15 (state:on)
    ("sw1 & !sw2" state: on) ("!sw1 & sw2" state: off)
  MTCMOS Switch Cell : switch_class_20 (state:on) ("sw" state: on)
  MTCMOS Switch Cell : switch_class_21 (state:off) ("sw" state: off)
Connected Supply Net : VDD2
  MTCMOS Switch Cell : switch_class_15 (state:on) ("sw" state: on)
  MTCMOS Switch Cell : switch_class_20 (state:on) ("sw" state: on)
  MTCMOS Switch Cell : switch_class_21 (state:on)
    ("sw1 & !sw2" state: off) ("!sw1 & sw2" state: on)
```

See Also

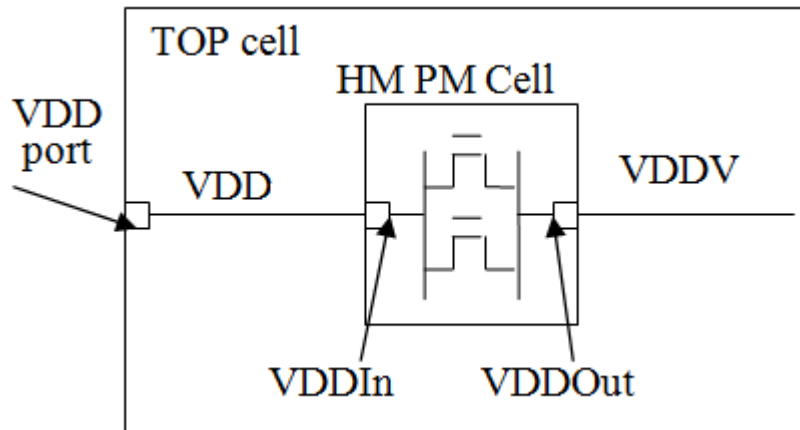
- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)

Rail Analysis With Hard Macro Power Management Cells

In PrimeRail, hard macro power management cells are the switch cells that have CONN views and power management location (PML) files of cell type macro. Usually, hard macro power management cells have multiple multithreshold-CMOS devices and complex pin shapes. You should not ignore these interconnect wires during rail analysis.

Figure 12-4 shows an example of a hard macro power management cell. In the figure, VDD is the main PG net. VDDV is the virtual net controlled by the hard macro power management cell; VDDV might not have a terminal. The hard macro power management cell has VDDIn and VDDOut nets. Both the HMPM.VDDIn pin and the HMPM.VDDOut pin connect to the TOP cell's VDD and VDDV nets, respectively. The net shapes are described in the CONN views. In this example, there are two multithreshold-CMOS devices in the hard macro cell. These two devices are modeled as two resistors and are described in the PML file.

Figure 12-4 An Example of A Hard Macro Power Management Cell



Analysis Flow When Analyzing With Hard Macro PM Cells

A hard macro power management cell must have CONN views and PML files if you want to consider the rail effects inside this hard macro power management cell. PrimeRail allows you to create CONN views and PML files with the `create_rail_macro_model` command. The CONN views provide the interconnection shapes and the PML files describe the on-resistance source and drain locations of power management cells.

PrimeRail requires the following data for analyzing the hard macro power management cell:

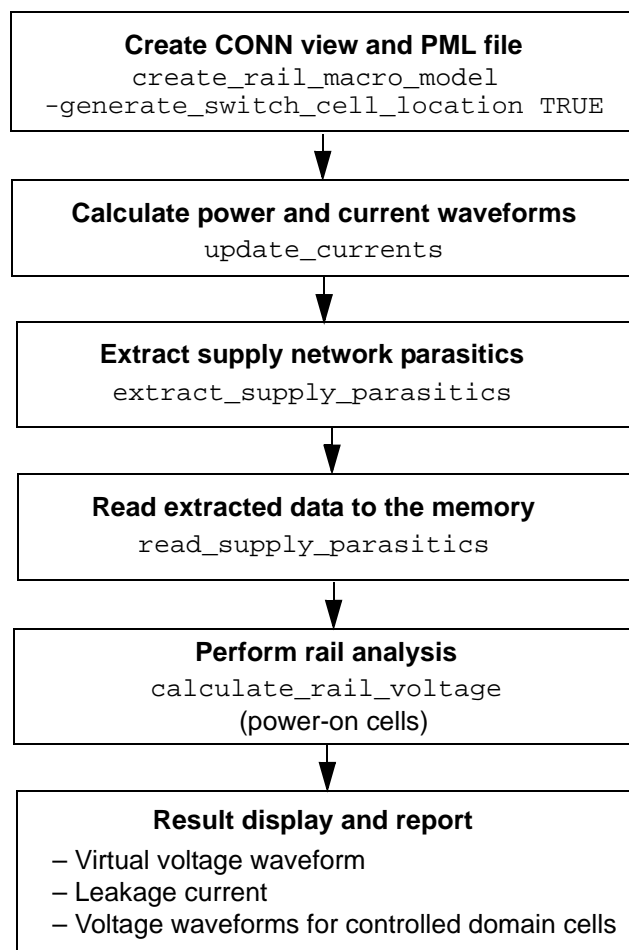
- The CEL view which is streamed in from the GDS data
- Main and virtual power and ground net names

- The number of the Oxide Diffusion (OD) layer that defines the diffusion of active area (that is, the transistor's source and drain). The tool uses this layer to find transistors that connect two different nets and replace them with a virtual resistor. This is required only when you want to run an automatic on-resistance insertion without using a PML file as input.

PrimeRail uses the diffusion layer for the transistor between the terminals for inserting resistance values.

Figure 12-5 illustrates the steps for analyzing a hard macro power management cell in the full-chip rail analysis flow.

Figure 12-5 Hard Macro Power Management Cell Design Flow

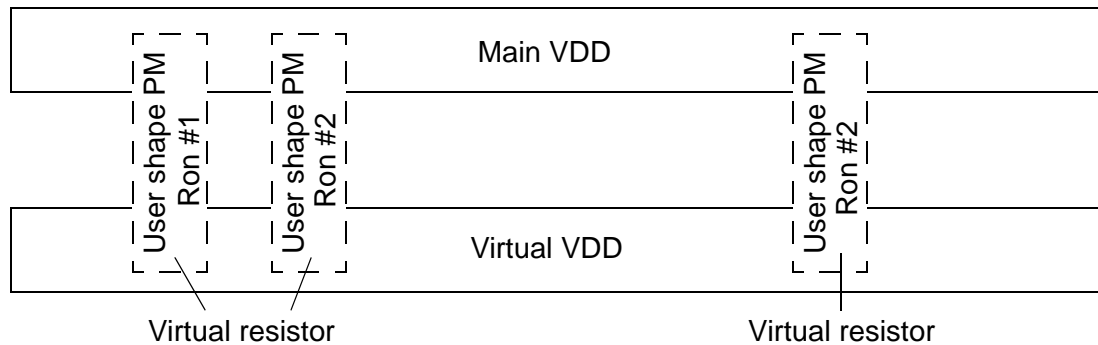


Power Management Location Files

A power management location file is an ASCII file that records the locations of on-resistances inside the cell. PrimeRail creates these power management locations as virtual resistors in the CONN view. These virtual resistors represent the on-resistance resistor of the power management cell, as shown in [Figure 12-6](#).

PrimeRail uses a power management location file when analyzing a power management cell during rail analysis.

Figure 12-6 Creating Power Management Locations in CONN views



See Also

- [Power Management Cells](#)
- [Creating CONN Views and PML Files](#)

Creating CONN Views and PML Files

PrimeRail requires CONN views and [Power Management Location Files](#) to analyze the voltage drop values and current density violations of a hard macro power management cell. Run the `create_rail_macro_model` command with appropriate options to automatically identify multithreshold-CMOS device locations (that is, the locations and values of on-resistances) for hard macros.

The following table lists the options of the `create_rail_macro_model` command for creating PML files.

Table 12-1 Options for Creating PML Files

Option	Description
<code>-generate_switch_cell_location TRUE</code>	Generates a power management location file and attach it to the CONN view being created.

Table 12-1 Options for Creating PML Files (Continued)

Option	Description
<code>-switch_cell_output_supply_nets supply_net_names</code>	Specifies the virtual supply net name for the power management cells. This is required when extracting a power management cell.
<code>-switch_cell_on_resistance ron_value</code>	Specifies the on-resistance value. During static rail analysis, the tool uses the specified on-resistance value to model a power management cell.
<code>-od_layer layer_number</code>	Specifies the OD layer number to be used to identify devices when extracting a power management cell. Use the option when you want the tool to automatically insert on-resistances without using the PML file.

Example

The following example shows how to generate a CONN view and a PML file for the MyDesign cell in the MyLibrary library, using the VDD power net, VDDV, as the virtual power net. The OD layer is 17 and the on-resistance is 3.4 ohm.

```
pr_shell> create_rail_macro_model -library MyLibrary -cell MyDesign \
        -power_nets VDD -generate_switch_cell_location TRUE \
        -switch_cell_output_supply_nets VDDV \
        -od_layer 17 -switch_cell_on_resistance 3.4
```

Manually Inserting On-Resistances

Use the `export_power_management_mtcmos_locations` command to write out the PML file. When the PML file is generated, you can edit the output PML file by manually changing the settings, such as on-resistance values. When finished, read in the edited PML file into the CONN view with the `import_power_management_mtcmos_locations` command.

If you want to remove the imported PML data from the CONN view, run the `remove_power_management_mtcmos_locations` command.

Performing Rail Analysis

When both CONN views and PML files are available, invoke PrimeRail and run the `open_mw_cel` command to open the design. PrimeRail first searches for the CONN view and PML file in the reference library. If PrimeRail finds a cell with PML data but its output power or ground pin is in the FRAM view, the tool identifies it as a hard macro power management cell.

By default, PrimeRail automatically generates the on-mode pin state for a hard macro power management cell, which means all signal points are in the power-on mode. PrimeRail collects signal pin names from the FRAM view. If you want to change the power mode to off, use the `set_annotated_mtcmos_switch_cell` command (see [Setting On-State Resistances](#)).

To perform top-level rail analysis on the hard macro power management cell,

1. Invoke PrimeRail and open the top-level design. Make sure all the reference libraries are open and correctly point to the macro model generated by the `create_rail_macro_model` command.
2. Run the `report_rail_macro_model` command to report all the hard macros used by this design.
3. Run the `extract_supply_parasitics` command to extract the parasitics of the supply nets inside the macro.

See [Extracting Supply Net Parasitics](#) for more information.

4. Read in the required setup and input information, such as SDC, SPEF, voltage constraints, switching activity information, and so on.

Run the `update_currents -static` command to calculate power and current waveforms of the design.

See [Extracting Supply Net Parasitics](#) for more information.

5. Run the `calculate_rail_voltage` command to perform voltage drop and electromigration analyses on the design, including the virtual power and ground nets that are controlled by the hard macro power management cell.

See [Static Rail Analysis](#) for more information.

See Also

- [Power Management Cells](#)

What-If Analysis With Switch Cells

You can optimize voltage drop and electromigration effects by manually changing the on-resistance values that are associated with each switch transistor. This allows you to easily diagnose design weaknesses of the power and ground network without repeating the CONN view and PML file creation process.

Note:

PrimeRail not only loads the PML file to memory for all the hard macros but also changes them into hard macro power management cells. However, if you run the `load_power_management_mtcmos_locations` command after the `open_mw_cel` command is run, the command changes the on-resistance values only.

To manually insert on-resistance values for what-if purposes,

1. Invoke PrimeRail and open the top-level design with the `open_mw_cel` command. Make sure all the reference libraries are open and correctly point to the macro models generated by the `create_rail_macro_model` command.
2. Edit the PML file exported by the `export_power_management_mtcmos_locations` command.
3. Modify the on-resistance values associated with each switch transistor.
4. Run the `load_power_management_mtcmos_locations` command. The command replaces the original hard macro power management cell data with those saved in the loaded PML file.

The following is an example PML file in which switch transistors are automatically identified.

Each entry represents one resistor being recognized

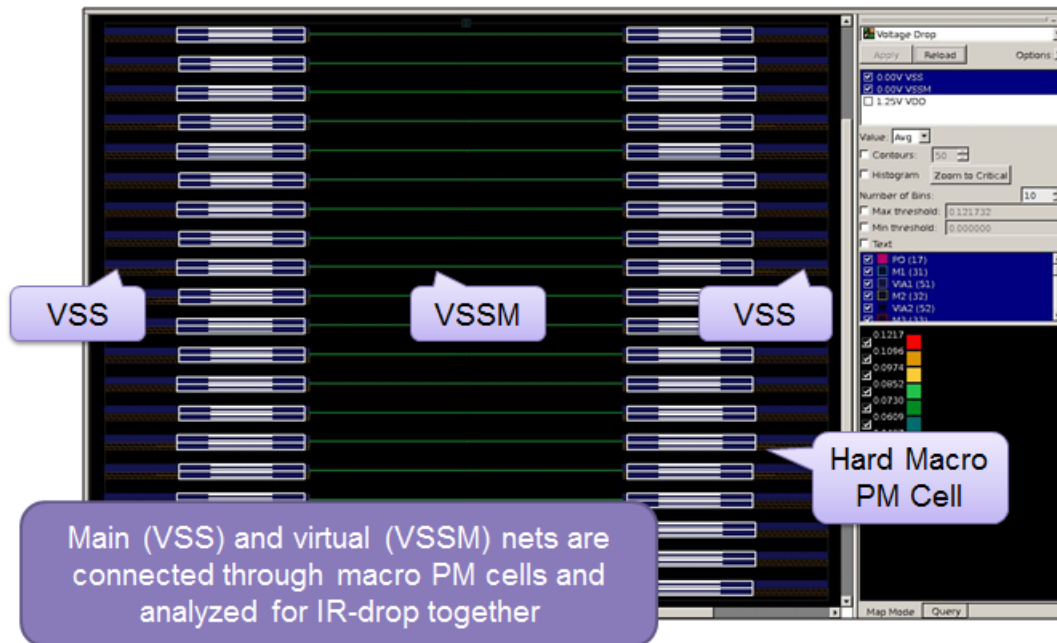
```
# Syntax
# from          to
# net layer    x      y net layer    x      y  res
VSS  31 43000      0 VSSM 31      43000 660 49.72
VSSM 31 43000 660 VSS  31      43000 1250 49.72
VSS  31 43000 1250 VSSM 31      43000 1900 49.72
VSSM 31 43000 1900 VSS  31      43000 2500 49.72
VSS  31 43000 2500 VSSM 31      43000 3100 49.72
VSSM 31 43000 3100 VSS  31      43000 3750 49.72
VSS  31 43000 3750 VSSM 31      43000 4400 49.72
VSSM 31 43000 4400 VSS  31      43000 5000 49.72
```

On-resistances are assigned uniformly based on the extracted power-gating MOS setting.

5. Complete the steps for performing rail analysis on the top-level design, including timing and power analysis, supply net extraction, and rail analysis.

- When rail analysis is complete, examine the voltage drop report and compare the results to see if the voltage drop reported for virtual nets reflects the changes made in the edited PML file.

Figure 12-7 Voltage Drop Map When Top-Level Rail Analysis is Complete



See Also

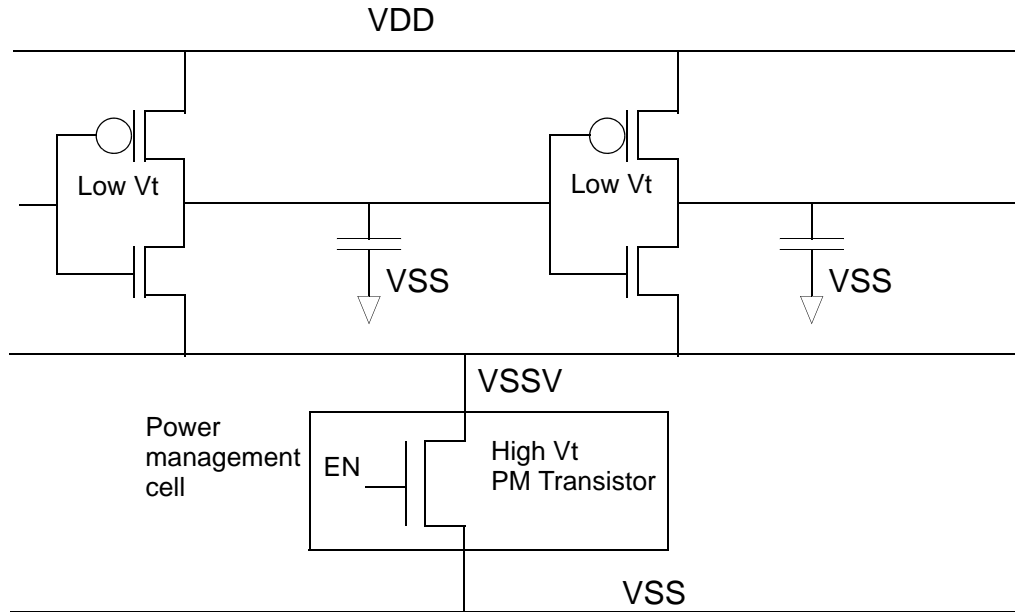
- [Rail Analysis With Multithreshold-CMOS Switch Cells](#)

Inrush Current Analysis

In multithreshold-CMOS circuits, power management cells with high threshold voltage (V_t) transistors are used to reduce the leakage power of power-gated (or switched) functional blocks when they are inactive. However, the number of the power management cells in a real design can be large and turning them on (or off) all at the same time can draw a lot of current from the power grid. You must carefully control the power-up sequence of those power management transistors to minimize large inrush current draw when the circuits are turned on for normal operation.

Use the inrush current analysis feature to control the simultaneous switching noise by turning on these circuits sequentially and setting current state of the virtual supply nets or the switching cells. Accordingly, the number of power management cells, their drive strength, and timing sequence of the control signals are the design parameters that must be optimized and verified through accurate circuit simulation.

Figure 12-8 A Multithreshold-CMOS Circuit



Shown in [Figure 12-8](#), when a power management transistor (High Vt) is turned off while in the inactive mode, the virtual ground net (VSSV) as well as internal logic nodes are set to logic state 1. When the power management transistor is turned on for a normal operation, the virtual ground net (VSSV) eventually reaches the VSS voltage level. The time for the VSSV net's voltage level to reach the VSS net's voltage level is called the “wake-up” time.

The logic nodes will be at 0 or 1, which is dependent on the primary inputs. For the internal logic nodes at state 0, discharging current would flow through the circuit's NMOS transistors, the power management transistor, then eventually to VSS. For the internal logic nodes charging to 1, the charge current flows from VDD to those nodes. A significant amount of short-circuit current also flows from VDD to VSS through the power management transistor. The sum of the discharging current and the short-circuit current is called “rush current.”

Inrush current analysis estimates rush current and wake-up time of the virtual power domains in either of the following ways:

- [Calculating Rush Current By Cell States](#)
- [Calculating Ramp-Up Current By Net States](#)

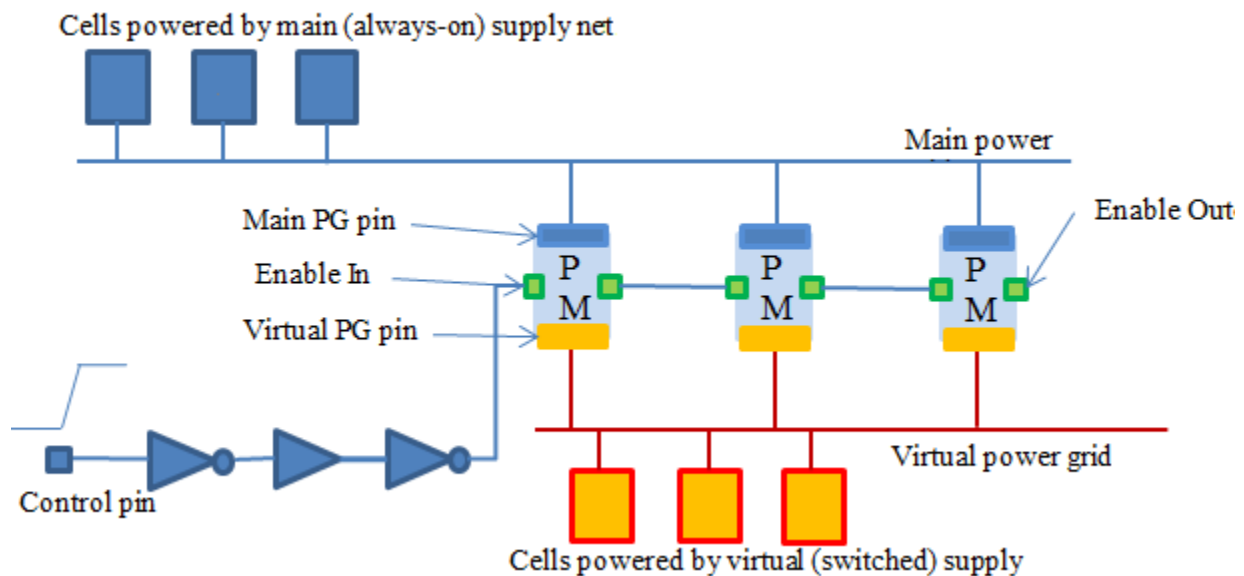
Note:

During inrush current analysis, PrimeRail does not change the OFF state back to ON after the state is transited from ON to OFF. Therefore, running inrush current analysis on designs with long VCD streams and multiple ON/OFF cycles might produce inaccurate analysis results. It is recommended that you run inrush analysis with an event file, such as a VCD file, for moderate runtime.

Power Gating Cells

A power gating logic block (that is, a virtual power domain) has three modes of operation: sleep mode, wake-up mode, and active mode. In sleep mode, power is cut off from the block to reduce the leakage power. In active mode, logic gates in the block need to function normally and the consumed power is supplied through the power management cells. The logic block is in wake-up mode when it makes transition from sleep mode to active mode. During wake-up, a large amount of current, also called inrush current, needs to flow through the power management cells to charge up the virtual power grid to the main supply voltage. This inrush current might cause $L \, di/dt$ noise (that is, supply bounce) to the main power grid and electromigration failures in the combined power grid. A combined power grid consists of a main power grid, virtual power grids, and power management cells which connect the main grid to the virtual grids. Figure 12-9 shows a simplified power-gated design in which a supply net is a logic net and the power grid is a physical implementation of a supply net.

Figure 12-9 An Example of a Power-Gating Design



Calculating Rush Current By Cell States

To calculate the rush current that occurs during the wake-up period, first specify the power-up sequence for the power management cells with the `create_mtcmos_switch_cell_sequence` command. The power-up sequence information can be from the VCD file or the power management cell arrival events that are either imported from a file or inferred from the static timing analysis.

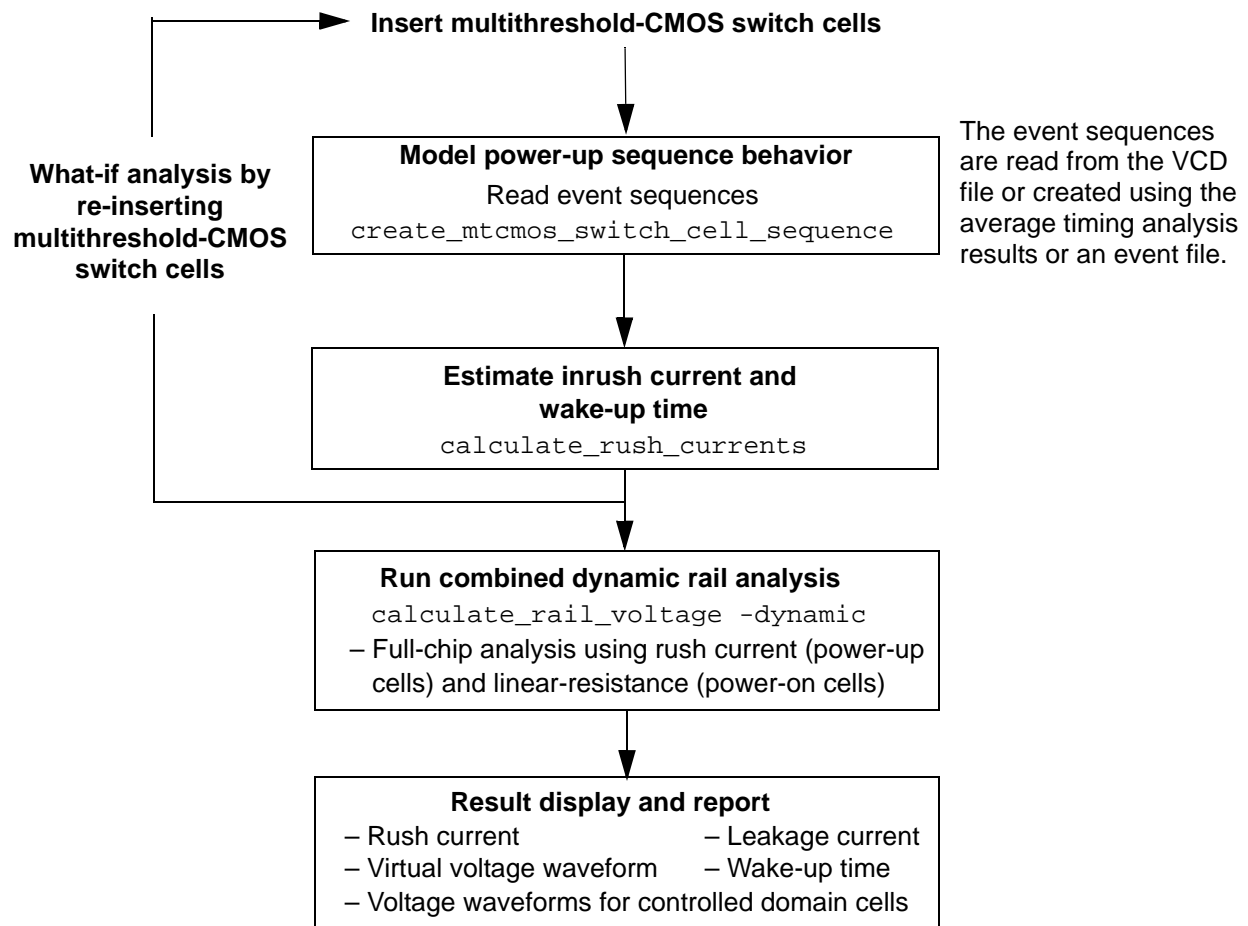
Then, run the `calculate_rush_currents` command to calculate the rush current and wake-up voltage waveforms at each power management cell. IV characteristics of the power management cells and voltage-dependent intrinsic capacitance of standard cell PG pins in

the CCS power library and Liberty library are also used to produce accurate results. You can run the inrush current analysis for what-if purposes while adjusting power-up sequence elements (or switch cell pin connectivities), such as peak current, turn-on time, and so on.

When inrush current analysis is complete, run the `calculate_rail_voltage` command to perform dynamic rail analysis on the power grid with the calculated rush current; the power grid has both main and virtual power domains. The main power domain connects to one or more virtual power domains; each virtual domain might be turned on or off any time during the simulation period. Running the dynamic rail analysis with inrush currents detects excessive voltage drops and electromigration violations on the combined power grid, while considering both rush currents and switching currents of the active devices. Package models at the PG taps are also taken into account during dynamic rail analysis.

Figure 12-10 illustrates the steps to perform inrush current analysis by cell states.

Figure 12-10 The Design Flow for Running Inrush Current Analysis by Cell States



To calculate rush currents at the wake-up period,

1. Use the `create_mtcmos_switch_cell_sequence` command to model the power-up sequence behavior of the switch cell by loading the event sequence information.

To report or remove the user-specified power-up sequence information, use the `report_mtcmos_switch_cell_sequence` or `remove_mtcmos_switch_cell_sequence` command. You cannot remove the power-up sequence information derived from a VCD file.

For more information about creating power sequences, see [Power-Up Sequence Information](#).

2. Run the `calculate_rush_currents` command to calculate the inrush currents flowing through the virtual supply nets. Specify the virtual nets and the `start_voltage` value (when the nets are in sleep mode) and the `stop_voltage` values (when the nets are fully charged). If `stop_voltage` is not defined, use the `-threshold` option to specify how the value of the virtual supply net is calculated.

For more information about calculating the rush current and the wake-up time, see [Running the calculate_rush_currents Command](#).

3. Run the `calculate_rail_voltage -dynamic` command to perform dynamic rail analysis on main and virtual power grids with rush current that is calculated during inrush current analysis. For a detailed description about how to analyze voltage drop violations with the calculated rush currents, see [Dynamic Rail Analysis With Inrush Currents](#).

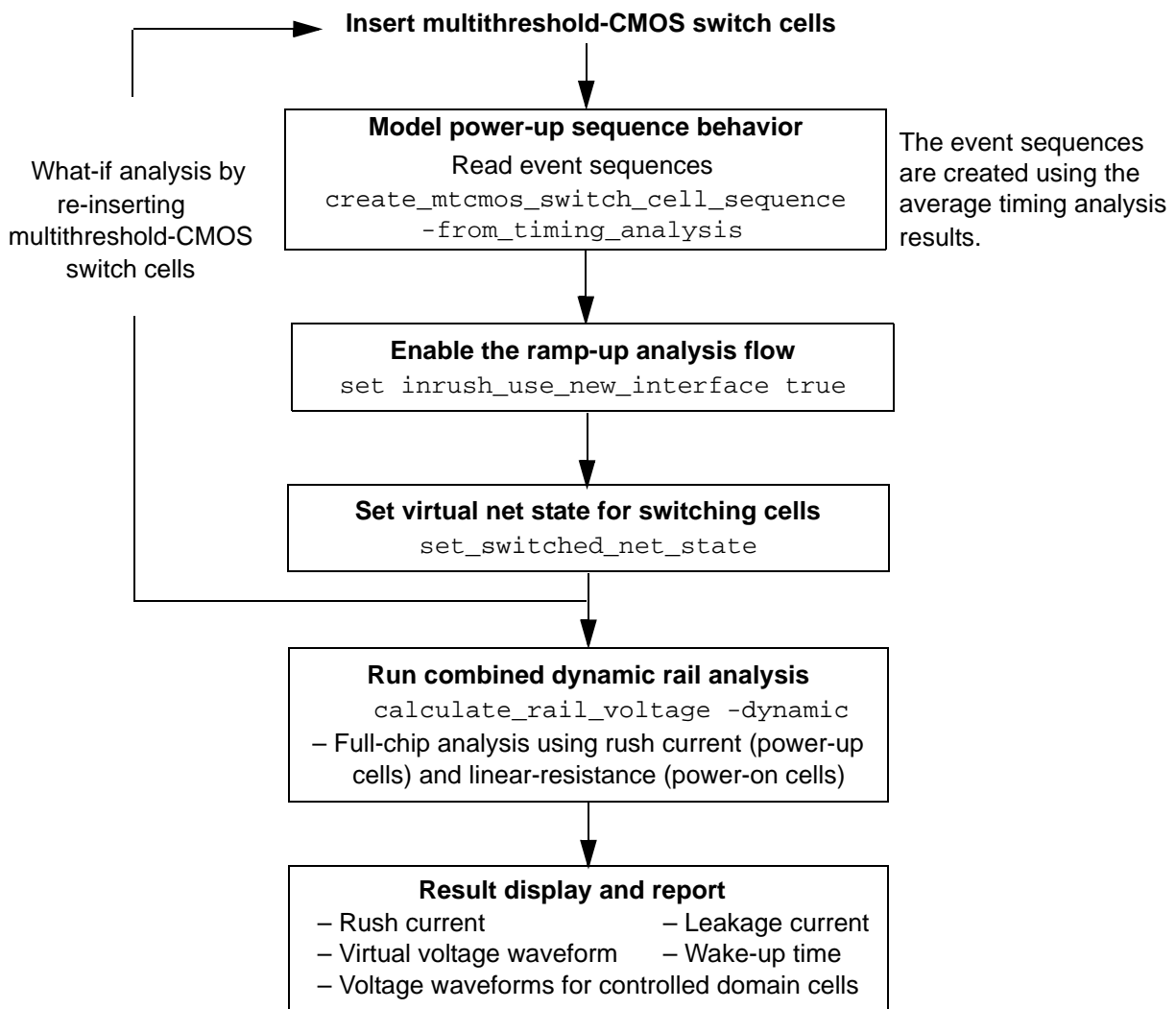
See Also

- [Inrush Current Analysis](#)
- [Calculating Ramp-Up Current By Net States](#)
- [Running the calculate_rush_currents Command](#)
- [Dynamic Rail Analysis With Inrush Currents](#)

Calculating Ramp-Up Current By Net States

By specifying the current state of the virtual supply nets to `ramp_up` with the `set_switched_net_state` command, the ramp-up current of various virtual nets, together with the main PG grid switching activity and the connected package model, are calculated when analyzing rush current for a power-gated design.

Figure 12-11 Design Flow of Inrush Current Analysis by Net States



To consider the ramp-up behavior when calculating rush currents for a power-gated design,

1. Open the design with the `open_mw_cel` command.
2. Run the `extract_supply_parasitics` command to generate the extraction data.
For a detailed description about parasitic extraction, see [Extracting Supply Net Parasitics](#).
3. Run the `update_currents -dynamic_vectorfree` command to generate dynamic power and current waveform data for obtaining capacitance data of the cells.
For a detailed description about timing and power analysis, see [Analyzing Power and Generating Current Waveforms](#).

4. Run the `create_mtcmos_switch_cell_sequence -from_timing_analysis` command to create a sequence file from the static timing analysis results.

For more information about creating power-up sequence data, see [Power-Up Sequence Information](#).

5. Set the `inrush_use_new_interface` variable to `true` to enable the ramp-up analysis flow.
6. Run the `set_switched_net_state` command to specify the state of the virtual nets for switching cells.

For example,

```
pr_shell> set_switched_net_state -state off VDD2
pr_shell> set_switched_net_state -state ramp_up {VDD3 VDD4}
pr_shell> set_switched_net_state -state ramp_up \
    -start_voltage 0.1 -stop_voltage 1.0 VDDV
```

To check the current state of the virtual supply nets, use the `report_switched_net_state` command.

For more information about setting or reporting net states for virtual nets, see the command man pages.

7. Run the `calculate_rail_voltage` command to retrieve the state status of the virtual nets of the VDD nets, and calculate voltage waveforms and rush current waveforms of a power grid which has both main and virtual power domains.

For example,

```
pr_shell> calculate_rail_voltage -dynamic {VDD VSS} \
    -fsdb_file waveform.fsdb
```

The command saves voltage waveforms, rush current waveforms for each PM cell, total rush current waveforms, and virtual net ramp-up voltage waveforms in an FSDB file for each virtual net. The FSDB files are saved in the `RAIL_DATABASE/fsdb_dir` directory.

See Also

- [Inrush Current Analysis](#)
- [Calculating Rush Current By Cell States](#)
- [Running the calculate_rush_currents Command](#)
- [Dynamic Rail Analysis With Inrush Currents](#)

Power-Up Sequence Information

PrimeRail needs the event sequence information to accurately calculate the inrush currents. The event sequence information is either extracted from a VCD file, an event file, or the static timing analysis result. When all three files are present, the event file takes a higher priority.

Use the `create_mtcmos_switch_cell_sequence` command to model the power-up sequence behavior of the switch cell by loading the event sequence information using one of the following methods:

- Reading from the VCD file

A VCD file records the logic changes for the design. When a VCD file is loaded using the `read_vcd` command, running the `calculate_rush_currents` command automatically uses the information from the VCD file for calculating inrush currents.

- Reading from the average timing analysis results

If a VCD file is not available, create power-up event sequences by retrieving the event sequence information from the static timing analysis result.

For example:

```
pr_shell> create_mtcmos_switch_cell_sequence \
          -from_timing_analysis
```

The command creates event sequences for all switch cells in the design.

Note:

Use the SDC commands to set the arrival time and input transition for the primary inputs in the switch cell's group path. If no SDC constraint is found for the primary inputs, by default, the arrival time is set to 0.0 ns while the input transition time is set to 50 ps.

To report or remove the user-specified power-up sequence information, use the `report_mtcmos_switch_cell_sequence` or `remove_mtcmos_switch_cell_sequence` command, respectively. You cannot remove the power-up sequence information derived from a VCD file.

Modifying Wake-Up Time

When the power-up sequence is already created, you can add an offset to all wave-up time points in the time path, using the `-offset` option of the `create_mtcmos_switch_cell_sequence` command. Doing this changes the event interval for an existing event sequence. When finished, check the generated event sequences with the `report_mtcmos_switch_cell_sequence` command.

For example, to add a universal time offset of 2.0 nanoseconds to all starting time points in the existing event sequences for all switch cell groups, use the following commands:

```
pr_shell> create_mtcmos_switch_cell_sequence -offset 2.0

# report switch cell sequence
pr_shell> report_mtcmos_switch_cell_sequence

[VCD-based pm cell switch events]
(cell name: {event_time signal_pin_states on_channel_list})
Cell name: header0.  Input signal pins: EN.  Output signal pins: EN_OUT.
PG port name: VDDT
PM cell header0 sequence:  {time=2.0ns  "EN_OUT"=1  "EN"=f
Switching_Pin="EN" Transition=1.43042e-11 [0]}

Cell name: header1.  Input signal pins: EN.  Output signal pins: EN_OUT.
PG port name: VDDT
PM cell header1 sequence:  {time=3.0ns  "EN_OUT"=1  "EN"=f
Switching_Pin="EN" Transition=1.43042e-11 [0]}
```

See Also

- [Calculating Rush Current By Cell States](#)
- [Calculating Ramp-Up Current By Net States](#)

Running the calculate_rush_currents Command

When the power-up sequence or net state information is available, use the `calculate_rush_currents` command to calculate the inrush currents that flow through the virtual supply nets in a power-gated design.

Each virtual supply net is independently analyzed based on the power-up sequence that is either automatically derived from the VCD file or manually specified by using the `create_mtcmos_switch_cell_sequence` command. When a VCD-based power-up sequence is used, the transition time for each switching pin is determined by the signal pin transition time derived from timing analysis. When a user-specified power-up sequence is used, the transition time for each switching pin is defined by using the `create_mtcmos_switch_cell_sequence` command. If a power-up sequence is specified but no transition time is defined for switching pins, a zero transition time is applied.

During inrush current analysis, the tool also calculates the total off-state leakage current for each virtual supply net, based on IV curves of the PM cells. The average leakage power consumption of each cell instance connected to the virtual supply net is also available.

For example,

```
pr_shell> calculate_rush_currents -start_voltage 0.0 \
                                     -stop_voltage 1.0 VDD
```


[Table 12-1](#) lists the commonly used options when running the `calculate_rush_currents` command to calculate rush currents and wake-up time.

Table 12-2 Commonly Used Options for the `calculate_rush_currents` Command

Option	Description
<code>-noexpand</code>	Excludes the virtual supply nets that are connected by switch cells. By default, the command analyzes all the virtual nets that are connected to the specified nets through the power-gated cells.
<code>-include_main_net_ir_drop</code>	Includes IR drops at each main PG pin of the power management cells during the inrush current analysis. When specified, the IR drop value at each main PG pin is estimated by its minimum path resistance value from the tap points.
<code>-fsdb_dir</code>	Specifies the name of the directory for saving the output FSDB files. The default is <code>\$rail_database/fsdb</code> .
<code>-reuse</code>	Reuses the power-up sequence information that was generated in a previous session. Use the option if you close the session but want to rerun the <code>calculate_rush_currents</code> command with different settings for what-if purposes.

When the calculation is finished, PrimeRail reports and saves the peak rush current values and the wake-up time for each virtual power domain in the working directory. These rush current waveforms are used in the subsequent [Dynamic Rail Analysis With Inrush Currents](#) step to calculate dynamic rail voltages with inrush currents.

Note:

For accurate inrush current analysis results, you must use properly characterized Synopsys Liberty libraries which contain voltage- and state-dependent intrinsic RC values at each standard cell PG pin and correct IV curves for each PM cell master.

See Also

- [Inrush Current Analysis](#)
- [Calculating Rush Current By Cell States](#)
- [Calculating Ramp-Up Current By Net States](#)

Dynamic Rail Analysis With Inrush Currents

To perform dynamic rail analysis on main and virtual power grids with the rush currents that are calculated during inrush current analysis, run the `calculate_rail_voltage -dynamic` command. This allows you to analyze the dynamic IR drop and electromigration violations of the power grid while simultaneously considering the following:

- Package models of the PG taps on the main power grid
- Active devices in the main power domain
- Active devices in the virtual power domain when the domain is in active mode
- Rush currents during the wake-up period of the virtual power domain
- Leakage currents of the power management cells
- ON-state resistances of the power management cells

During the simulation period, the operating mode of each virtual power domain changes from sleep to wake-up and then active, or the reverse. The following list explains how PrimeRail models a virtual power domain and the related power management cells for each operating mode during dynamic rail analysis with inrush current.

- When the combined power grid model is in sleep mode,
 - The RC network of the main power grid is used.
 - The resistive network of the virtual power grid is used, which means wire and intrinsic capacitances are ignored.
 - The power management cells are replaced by their ON-state resistances.
 - The power management cell leakage current (constant) is attached to its main PG pin.
 - Logic device currents, including leakage currents, are set to zero.
- When the combined power grid model is in wake-up mode,
 - The RC network of the main power grid is used.
 - The resistive network of the virtual power grid is used.
 - The power management cells are replaced by their ON-state resistances.
 - The rush current waveforms saved during the inrush analysis are attached to the main PG pins of the power management cells.
 - Logic device currents, including leakage currents, are set to zero.

- When the combined power grid model is in active mode,
 - The RC network of the virtual power grid is used. The wire and intrinsic capacitances are added at the beginning of the active mode.
 - The power management cells are replaced by their ON-state resistances.
 - Logic device currents from the ON-state resistances of the combined power domain are used.

See Also

- [Inrush Current Analysis](#)
- [Calculating Rush Current By Cell States](#)
- [Calculating Ramp-Up Current By Net States](#)

13

Other Analysis Topics

Aside from voltage drop and electromigration analyses, PrimeRail provides other analysis capabilities, such as reduced core models, top-level-only analysis, pin-to-pad resistance calculation, and minimum path resistance analysis.

The chapter consists of the following sections:

- [Top-Level-Only Analysis](#)
- [Calculating Effective Resistances](#)
- [Reporting Effective Voltage Drops](#)
- [Reporting Switching-Window-Based Effective Voltage Drops](#)
- [Minimum Path Resistance Analysis](#)
- [Rail Analysis With Decoupling Capacitors](#)
- [Voltage Drop Analysis With Regulators](#)
- [Advanced Electromigration Analysis](#)
- [What-If Analysis](#)

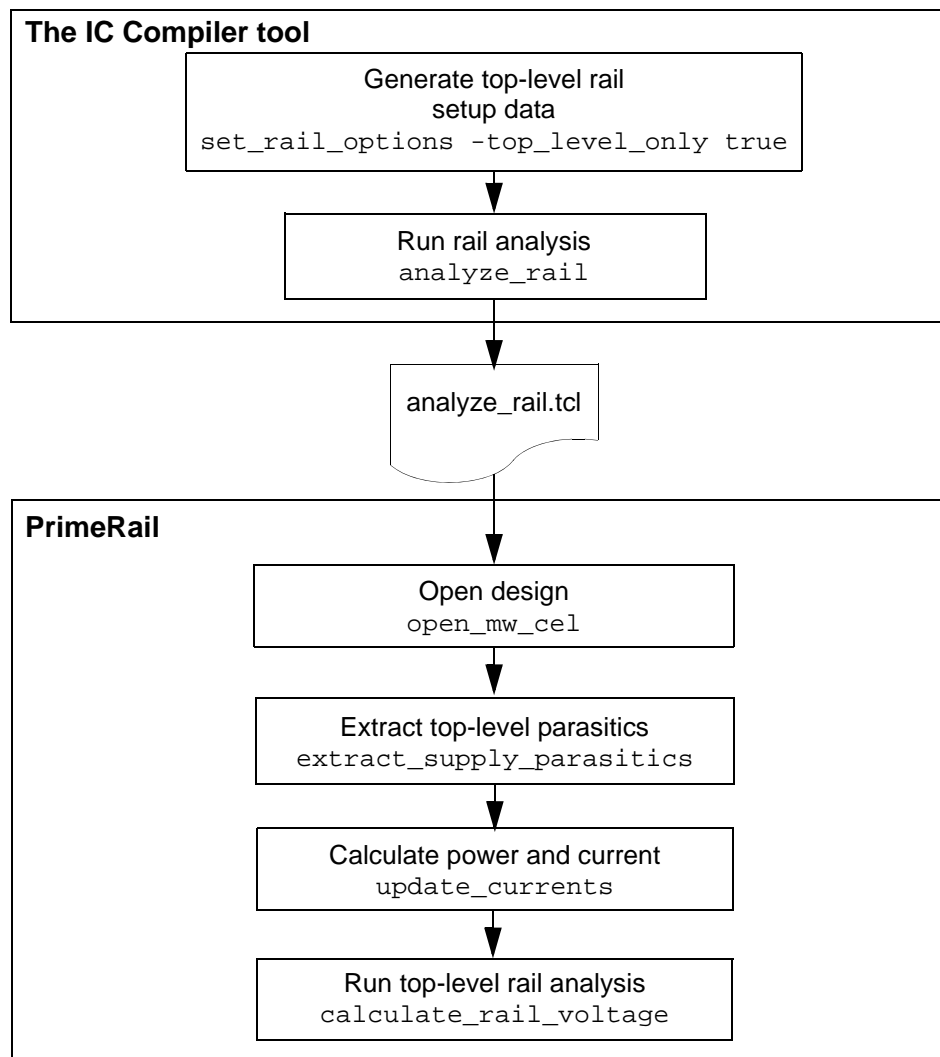
Top-Level-Only Analysis

Early in the hierarchical design process, floorplans are partitioned into subblocks, and top-level and block-level implementations are performed in parallel by different design groups. When no sub-blocks are available, you need to do top-level rail analysis in order to check the robustness of the overall power network.

PrimeRail allows you to analyze only the top-level routing of the design in combination with the results generated from the In-Design rail analysis in the IC Compiler environment.

[Figure 13-1](#) illustrates the steps to perform a top-level-only analysis with the rail setup data generated from the In-Design rail analysis flow in the IC Compiler environment.

Figure 13-1 Performing Top-Level Only Analysis



See Also

- [Analyzing Top-Level Designs](#)

Analyzing Top-Level Designs

To analyze a top-level design,

1. If you prepare your design data in the IC Compiler environment, you should
 - Run the `set_rail_options -top_level_only true` command. When the `-top_level_only` option is enabled, the IC Compiler tool generates the script and the SPEF and SDC files for the top-level design only. Neither SPEF nor SBPF files are created for the `update_currents` command.
 - Run the `analyze_rail` command. The tool generates a replay file for use in PrimeRail to perform the top-level-only rail analysis.
2. In PrimeRail, run the `open_mw_cel` command to read in a Milkyway design. All the executed commands honor this setting afterwards.
3. Proceed to the remaining of the steps to complete the rail analysis flow,
 - a. Run the `extract_supply_parasitics` command to generate top-level-only extraction data.
For more information about supply net extraction, see [Extracting Supply Net Parasitics](#).
 - b. Read in the necessary input data and design constraints, such as the SDC file, the SPEF file, the voltage setting with the `set_voltage` command, taps and so on.
For more information about loading input data and design constraints, see [Preparing and Checking Design Data](#).
 - c. Run the `update_currents` command to generate top-level-only power and current waveform data.
For more information about creating current waveforms, see [Extracting Supply Net Parasitics](#).
 - d. Run the `check_supply_net_integrity` command to perform top-level-only integrity checking.
For more information about integrity checking, see [Integrity Checking](#).
 - e. Run the `calculate_rail_voltage` command to perform top-level-only rail analysis, since both extraction and power data is all top-level-only. Note that all the data for subblocks will be removed, if

- A subblock is complete, meaning a FRAM view of this subblock is included in the database. The tool ignores any power and extraction data from this subblock. However, if you assign a power value to this subblock, the tool honors this power value during rail analysis.
- A subblock is incomplete, meaning no FRAM view of this subblock is included in the database. The tool ignores this subblock completely, along with the assigned power value.

For more information, see [Static Rail Analysis](#).

Note:

You cannot reuse data from a previous full-chip analysis in the top-level analysis flow, and vice versa.

See Also

- [Top-Level-Only Analysis](#)

Calculating Effective Resistances

Use the `calculate_effective_resistance` command to calculate the effective resistance between a given power or ground pin and the taps. When calculation is complete, the tool reports the effective resistances and the number of floating pins in the log window. To write the report to an output file, use the `-output_file_name` option.

Important:

You must create taps using the `create_taps` command before you execute the `calculate_effective_resistance` command.

In PrimeRail, you

- Use the `-all` option to calculate effective resistances for all the power and ground pins of the current design.

Here is an example:

```
pr_shell> calculate_effective_resistance -all [get_supply_nets]
```

- Use the `-pg_pins` option to calculate effective resistances for target power or ground pins, or the `-pin_list_file` option to describe the target pins in a text file.

The following example calculates effective resistances for the `instA/VDD` and `instB/VSS` pins that are specified in the file named `pin_list`:

```
pr_shell> exec cat pin_list
instA/VDD
instB/VSS
pr_shell> calculate_effective_resistance \
```



```
-pin_list_file pin_list [get_supply_nets]
```

- Calculate effective resistances from power or ground pins to pads by describing the target pin and tap names in a text file and then importing it with the `-import` option.

The following example reads the file `p2p.in` as the input file and saves the effective resistances in an output file named `p2p.out`.

```
pr_shell> calculate_effective_resistance -import p2p.in \
        -output_file_name p2p.out
```

Preparing an Input File for PG Pin-to-Pad Resistance Calculation

In a text file, describe the names of power or ground pins and taps in the following format:

```
pin_name tap_name
```

The `pin_name` can be either the name of a pin or the location of a power or ground pin. If the name of a pin contains white space, specify the name with quotation marks. If the name of a pin ends with the asterisk character (*), the command considers all pins with names that match the specified pin name pattern. Note that the asterisk character (*) has to be at the end of a pin name, not an instance name.

- To specify a pin with its location, describe the net name, the layer number, the x-coordinates, and the y-coordinates within parentheses. For example, to specify the pin in net VDD in layer 31 located at (30.0, 50.0), enter the following:

```
(VDD 31 30.0 50.0)
```

- Specify a tap with its layer number, x-coordinate, and y-coordinate within parentheses. If no taps are specified, the command considers all taps in the design. For example, to specify a tap at (0.2, 0.5) in layer 31, enter the following:

```
(31 0.2 0.5)
```

- To insert a comment, use a semicolon character (;). Any characters after a semicolon are treated as comments.

Example

```
;Calculate one instance pin to one boundary tap point.
"blk1.VDD 1" (28 327.880 317.100)
```

```
;Calculate one instance pin to 3 boundary tap points.
"blk1.VDD 1" (28 327.880 317.100) (18 10.500 0.280) (28 13.400
317.090)
```

```
;Calculate resistance for instance pins whose names matched with
;wildcards to one boundary tap point.
blk1.VDD* (28 327.880 317.100)
```

```
;Calculate instance pins whose names matched with wildcards to all
;boundary tap points.
```

```

blk1.VDD*

;Specify one instance pin by its location and calculate the
resistances
;to all boundary tap points.
(VDD 28 87.0 76.24)

```

Reporting Effective Voltage Drops

During combined static and dynamic rail analysis, PrimeRail calculates effective voltage drops and rises based on the pin types defined in the .db file. For each cell instance, PrimeRail traces the difference in the voltage drop and rise points for every time step and then takes the maximum value over the simulation time as the effective voltage drop value.

By default, PrimeRail calculates effective voltage drops for the whole rail analysis session. To calculate the effective voltages on the timing window in which the cell is in transition, set the `pr_timing_window_effective_voltage` variable to `true` before running the `calculate_rail_voltage` command.

To examine the effective voltage drops and rises of each domain, run the `report_rail_voltage -effective_voltage_drop_or_rise` command after rail analysis is complete. PrimeRail creates effective voltage drop data for the power domain VDD and the ground domain VSS. By default, the tool reports the top five voltage drops or rises for each supply net. To control the number of effective drop values reported, use the `-limit` option.

You can display the effective voltage drop map to check the total reduction of cell bias caused by the voltage drop in the VDD net and the ground bounce in the VSS net.

You can also examine the waveform for an effective instance voltage drop point in the waveform viewer.

Example

The following example runs combined rail analysis on the supply nets VDD and VSS, and reports the top 5 effective voltage drop values for supply nets VDD and VSS.

```

pr_shell> calculate_rail_voltage -static|dynamic [list VDD VSS]
pr_shell> report_rail_voltage {VDD VSS} \
    -effective_voltage_drop_or_rise -limit 5

```

```

#Effective Voltage Drops Report
Power domain VDD top 5 effective instance voltage drop report
1: 24.838 (mV) at core/VDD time 39e-9 (voltage drop 11.486 mV
   at VDD, voltage rise 13.352 mV at VSS)
2: 24.74 (mV) at core/VDD time 39e-9 (voltage drop 10.85 mV
   at VDD, voltage rise 13.89 mV at VSS)
3: 24.735 (mV) at core/VDD time 39e-9 (voltage drop 11.744 mV

```

```

    at VDD, voltage rise 12.991 mV at VSS)
4: 24.721 (mV) at core/VDD time 39e-9 (voltage drop 11.148 mV
    at VDD, voltage rise 13.573 mV at VSS)
5: 24.644 (mV) at core/VDD time 39e-9 (voltage drop 11.794 mV
    at VDD, voltage rise 12.85 mV at VSS)

Power domain VSS top 5 effective instance voltage rise report
1: 24.838 (mV) at core/VSS time 39e-9 (voltage rise 13.352 mV
    at VSS, voltage drop 11.486 mV at VDD)
2: 24.74 (mV) at core/VSS time 39e-9 (voltage rise 13.89 mV
    at VSS, voltage drop 10.85 mV at VDD)
3: 24.735 (mV) at core/VSS time 39e-9 (voltage rise 12.991 mV
    at VSS, voltage drop 11.744 mV at VDD)
4: 24.721 (mV) at core/VSS time 39e-9 (voltage rise 13.573 mV
    at VSS, voltage drop 11.148 mV at VDD)
5: 24.644 (mV) at core/VSS time 39e-9 (voltage rise 12.85 mV
    at VSS, voltage drop 11.794 mV at VDD)

```

Reporting Switching-Window-Based Effective Voltage Drops

During dynamic or dynamic vectorfree rail analysis, by default PrimeRail calculates voltage drops of the power and ground network and reports cell instances that have the maximum, minimum and average voltage drops across the entire simulation period. Enable the switching-window-based effective voltage capability if you want to calculate effective voltage drops that occur within timing windows of each cell.

Note:

Not all cells have switching events generated during VCD or dynamic vectorfree analysis. The effective voltage drop values for those non-switching cells is zero.

To calculate effective voltage drops within timing windows,

1. Run the `set_effective_voltage_options` command with the `-type switching_window` option to enable the switching-window-based effective voltage capability.
2. Run the `update_currents -dynamic|dynamic_vectorfree` command to calculate dynamic current waveforms. Specify other options as necessary.

For more information about generating time-varying waveforms, see [Analyzing Power and Updating the Currents](#).

3. To calculate effective voltage drops, run the `calculate_rail_voltage -dynamic` command. Specify other options as necessary.

For more information about dynamic rail analysis flow, see [Static Rail Analysis](#).

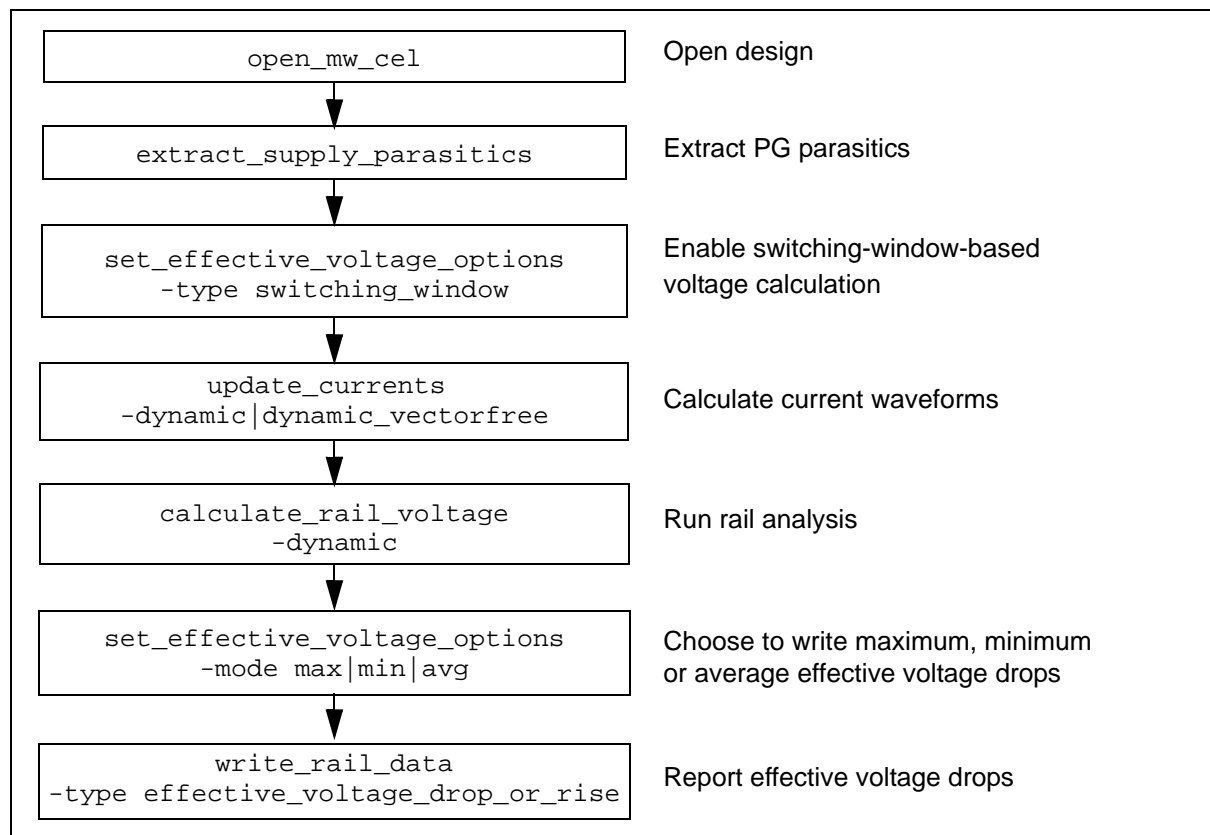
4. When rail analysis is complete, the rail results contain all maximum, minimum and average effective voltage drop values.

Run the `set_effective_voltage_options` command with the `-mode max|min|avg` option to specify whether maximum, minimum or average effective voltage drops are to be reported in step 5. The default is `max`.

5. Run the `write_rail_data` command with the `-type effective_voltage_drop_or_rise` option to write the effective voltage drop values to a text file.

Figure 13-2 illustrates the steps to calculate effective voltage drops that occur within timing windows of each cell.

Figure 13-2 Calculating Switching-Window-Based Effective Voltage Drops



Example

```
# Enable switching-window-based effective voltage calculation
set_effective_voltage_options -type switching_window

update_currents -dynamic
calculate_rail_voltage -dynamic {VDD VSS}

# Report the maximum effective voltage within switching windows.
set_effective_voltage_options -mode max
write_rail_data -type effective_voltage_drop_or_rise max.rpt

# Report the minimum effective voltage within switching windows.
set_effective_voltage_options -mode min
write_rail_data -type effective_voltage_drop_or_rise min.rpt

# Report the average effective voltage within switching windows.
set_effective_voltage_options -mode avg
write_rail_data -type effective_voltage_drop_or_rise avg.rpt
```

Minimum Path Resistance Analysis

Use the minimum path resistance analysis feature to quickly detect power and ground network weaknesses in the design.

In PrimeRail, the minimum path resistance value of a node is the resistance value on the smallest resistive path from the node to one of the design's boundary nodes, that is, ideal voltage sources (taps). The ideal voltage sources can be power or ground pins, user-defined taps, or packagings. Because this value represents only the resistance on a single path, it does not need to be exactly the same as the effective resistance value from the node to the global ground. It represents an upper bound of the effective resistance of a node to the ground.

To calculate the path resistances from each point in the power and ground network to the tap point on any of the target nets that are reachable along the least resistive path, use either of the following methods:

- [Using the calculate_minimum_path_resistances Command](#)
- [Calculating Minimum Path Resistance During Voltage Drop Analysis](#)

When the minimum path resistance analysis is complete, you can choose to save the calculated results to a rail result for later retrieval. The results file lists the shortest resistance path from the selected instance to the closest pad, as well as information about the resistors in this path. Use the minimum path resistance map if you want to debug rail analysis issues (see [Debugging With Minimum Path Resistance Maps](#)).

Using the `calculate_minimum_path_resistances` Command

Important:

You must load the supply net parasitics into the session before executing the `calculate_minimum_path_resistances` command.

The `calculate_minimum_path_resistances` command automatically expands the nets specified with the `supply_nets` argument to include all of the supply nets that are connected to the listed supply nets through power switch cells. Doing this allows the command to analyze all of the connected power domains at the same time. Use the `-noexpand` option to suppress the expansion.

Example

The following example shows how to calculate the minimum path resistances for the nets VDD1 and VDD2.

```
pr_shell> calculate_minimum_path_resistances {VDD1 VDD2}
```

The following example calculates the minimum path resistances for the net VDD and saves the result to MyVDDRResult.

```
pr_shell> calculate_minimum_path_resistances -result MyVDDRResult \  
[get_supply_nets VDD]
```

See Also

- [Debugging With Minimum Path Resistance Maps](#)

Calculating Minimum Path Resistance During Voltage Drop Analysis

Alternatively, you can calculate the minimum path resistances of the design during voltage drop analysis by using the `-resistivity` option of the `calculate_rail_voltage` command. When the analysis is completed, the tool writes the minimum path resistance values into the same rail result with voltage drop and bounce results.

The following example performs voltage drop analysis and minimum path resistance calculation at the same time and saves the result to `ir_resistivity`:

```
pr_shell> calculate_rail_voltage VDD -resistivity -result ir_resistivity
```

See Also

- [Debugging With Minimum Path Resistance Maps](#)

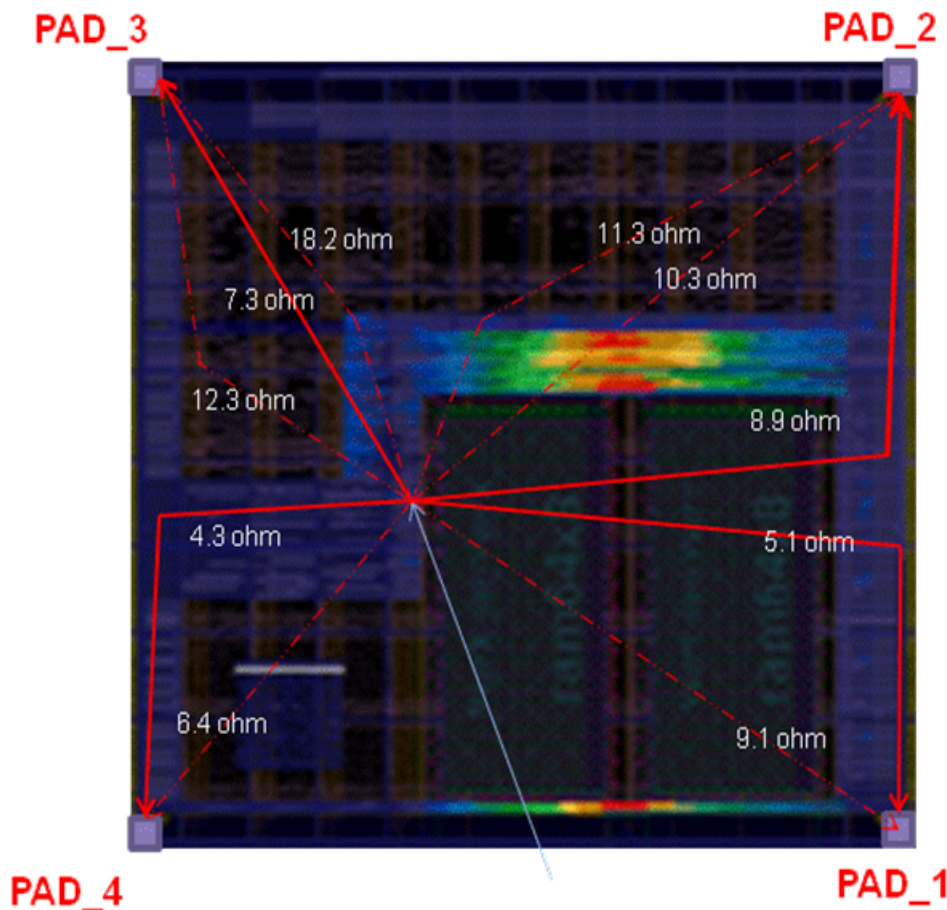
Debugging With Minimum Path Resistance Maps

When running the `calculate_minimum_path_resistances` or `calculate_rail_voltage -resistivity` command for resistance calculation, the tool saves the analysis results to a result for displaying maps or error views. Use the `-result` option if you want to assign a name to the result. By saving results from multiple command runs to different result sets, you can easily compare and debug the power and ground grid weaknesses for what-if purposes. You can always specify a result to be used in the session with the `current_rail_result` command.

To display a minimum path resistance map,

1. In the PrimeRail Layout window, choose Rail > Min Path Resistance from the GUI.
2. Click Apply. The Minimum Path Resistance map is shown in the layout window. The minimum paths are highlighted in different colors in the map (see [Figure 13-3](#)).

Figure 13-3 Displaying Minimum Path Resistance in the GUI



See Also

- [Minimum Path Resistance Analysis](#)

Rail Analysis With Decoupling Capacitors

Adding decoupling capacitance (decap) cells in the affected areas is one of the widely used techniques to solve power supply noises. In PrimeRail, use the `insert_virtual_decap_cells` command to virtually insert decoupling capacitors in a region where huge voltage drops might occur. The region is specified as a bounding box.

During insertion, PrimeRail uses all the empty spaces in the standard cell rows plus the existing filler and decap cells as resources for inserting virtual capacitors. For a quick estimate, you can specify how much capacitance is used for all the decap cells inserted in the region. The tool evenly distributes the total capacitance value to all capacitive nodes within the region without considering any layout information (for example, empty spaces in the standard cell rows).

When the insertion is complete, rerun dynamic rail analysis to validate the efficiency of the inserted decoupling capacitors. The rail analysis result describes the decoupling capacitance value between the coupled power and ground nets.

If you are not satisfied with the insertion result, run the `remove_virtual_decap_cells` command to remove all the inserted decap cells or only those in the specified region.

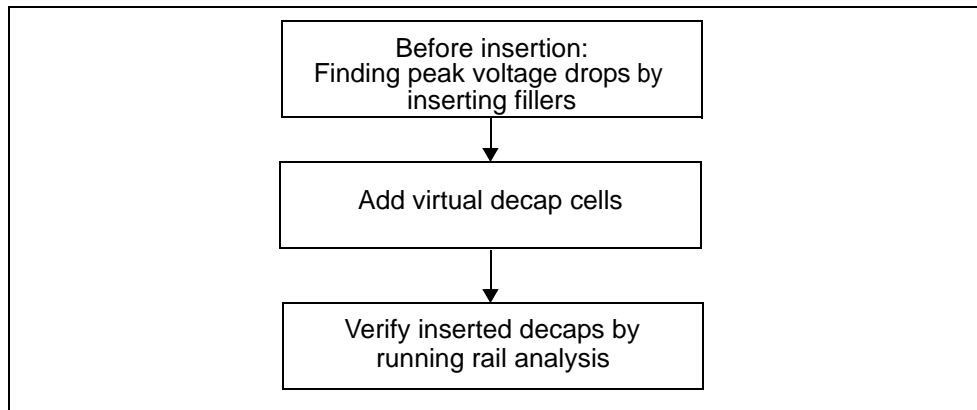
Note that in PrimeRail, inserting a decap cell is not effective in the following cases:

- When the power and ground network is not properly routed, some instances are “isolated” from the rest of the network. The effective resistance from the ideal voltage source to these instances is so high that the dynamic voltage drop is dominated by large resistance. In this case, solving the voltage drop problem requires unrealistically large capacitance to reduce the voltage drop.
- When the power and ground network is close to ideal, there is a very small effective parasitic resistance. It usually happens on a very conservative power and ground net design or flip-chip packaging. The dynamic voltage drop is dominated by a large peak inductance. To solve this problem, reduce the peak current instead of adding more capacitors.
- When the inductive noise is a dominant factor in the dynamic voltage drop flow, the number of affordable decoupling capacitors inserted is much less than what is needed to reduce the noise effectively.

Virtual Decap Insertion Flow

[Figure 13-4](#) illustrates the flow for inserting decoupling capacitors to reduce voltage drops within a user-specified region.

Figure 13-4 Virtual Decoupling Capacitor Insertion Flow



Inserting virtual decap cells consists of the following steps:

1. Before insertion, you can add fillers to the design by using any place and route tool, such as the IC Compiler tool. Then run CCS power library characterization to characterize the intrinsic capacitance and leakage current for these filler and decap cells in PrimeRail. When the filler cells are added to the design, regenerate the rail database and extract the parasitic information.

For more information about characterizing intrinsic capacitance and leakage current, see [Running Library Characterization](#).

If you want to overwrite the intrinsic capacitance and leakage current derived from library characterization for what-if purposes, run the `set_rail_current` and `set_intrinsic_rc` commands.

For more information about setting assigned intrinsic capacitance and leakage current on master cells, see [Setting Assigned Leakage Current or Capacitance](#).

2. Run the `insert_virtual_decap_cells` command to insert virtual decap cells in the area that might have large voltage drops, that is, voltage drop hotspots. During insertion, PrimeRail uses all the empty spaces in the standard cell rows plus the existing filler and decap cells as decap resources. A voltage drop threshold is required to calculate excessive voltage drops for cell instances within the user-specified region.
3. Run the `calculate_rail_voltage` command to verify the efficiency of the inserted virtual decap cells. If the maximum voltage drop is not at the desired level, increase the input capacitance value and then rerun dynamic rail analysis.

Run the `remove_virtual_decap_cells` command to remove the unnecessary decap cells. To report the information of the added virtual decap cells in an output file, run the `report_virtual_decap_cells` command.

4. Run the `save_mw_cel` command to save the result.

Setting Assigned Leakage Current or Capacitance

To override the capacitance or leakage current values derived from library characterization, use the `set_rail_current` and `set_intrinsic_rc` commands. The assigned capacitance or leakage current values are honored only by the `calculate_rail_voltage` command.

Note:

You must assign capacitance or leakage current values on master cells before the design is linked using either the `open_mw_cel` or `open_block` command.

Running the `set_rail_current` and `set_intrinsic_rc` commands before or after the design is linked results in the following behavior and unit changes.

Before the <code>open_mw_cel</code> or <code>open_block</code> command	<p>If the library cell already exists in <code>.db</code>, the tool overwrites the capacitance or leakage current values with the assigned ones.</p> <p>If the library cell does not exist in <code>.db</code>, the tool creates a new library cell and replaces the existing capacitance and leakage current values with the assigned ones.</p> <ul style="list-style-type: none"> - Current unit in <code>set_rail_current -waveform</code> is ampere. - Time unit in <code>set_rail_current -waveform</code> is sec. - Resistance unit in <code>set_intrinsic_rc</code> is ohm. - Capacitance unit in <code>set_intrinsic_rc</code> is farad.
After the <code>open_mw_cel</code> or <code>open_block</code> command	<p>If library cell already exists in <code>.db</code>, the tool overwrites the capacitance or leakage current values with the assigned ones.</p> <p>If library cell does not exists in <code>.db</code>, the tool stops the process with a warning message.</p> <p>The units for current, time, resistance and capacitance are the same as the main library.</p>

[Table 13-1](#) lists the options of the `set_rail_current` command used to specify user-defined leakage currents on a decap cell:

Table 13-1 Options of the `set_rail_current` Command to Specify User-Defined Leakage Current

Option	Description
<code>-lib_cell</code>	Specifies a master cell name to which rail current is applied. Only cell name string list is supported. This option is unavailable when the design is linked.
<code>-pg_pin</code>	Specifies the PG pin name to which the rail current is applied.
<code>-leakage</code>	Sets the mode as leakage to apply user-defined leakage currents on PG pin only cells.

[Table 13-2](#) lists the options of the `set_intrinsic_rc` command used to specify user-defined capacitance on a decap cell:

Table 13-2 Options of the `set_intrinsic_rc` Command to Specify User-Defined Capacitance

Option	Description
<code>-lib_cell</code>	Specifies a master cell name to which intrinsic resistance and capacitance values are applied. Only cell name string list is supported. This option is unavailable when the design is linked.
<code>-pg_pin</code>	Specifies the PG pin name to which intrinsic resistance and capacitance values are applied.
<code>-rc_list {r, c, r, c...}</code>	Specifies the resistance and capacitance values to be applied.

Script Example

```
# set decap cell cap 1fF
set_intrinsic_rc -rc_list {0.0 1.0e-15} -lib_cell DCAP64 -pg_pin VDD

# set decap leakage current 1nA
set_rail_current -leakage 1.0e-9 -lib_cell DCAP64 -pg_pin VDD

# report intrinsic RC just set
report_intrinsic_rc -lib_cell DCAP64

# report leakage current just set
report_rail_current -lib_cell DCAP64

# open top_design from MW lib top
open_mw_cel -library top top_design
```

```
# set intrinsic RC for an instance
set_intrinsic_rc -rc_list {0.0 2.0e-15} [inst1_dcap64/VDD]

# run rail analysis
calculate_rail_voltage VDD -dynamic
```

See Also

- [Virtual Decap Insertion Flow](#)
- [Inserting Virtual Decap Cells](#)

Inserting Virtual Decap Cells

To insert virtual decap cells within the specified area before the replacement is actually done, run the `insert_virtual_decap_cells` command. By assigning the total capacitance for the area, the capacitance values are evenly distributed to all the capacitive nodes within the area without considering any layout information in the design. The capacitance value for each inserted virtual decap cell is calculated as

$$(\text{total capacitance}) / (\text{Number of cell instances})$$

The virtual decap cell is added between a pair of power and ground pins of the cell instance connected to the specified power and ground nets during dynamic rail analysis. This provides a quick estimate on how the decap cells are placed without the decap cell list.

When a cell instance belongs to one or more user-specified regions, its virtual decap value is increased accordingly. For example, if a cell instance gets 5 fF from one region and 2 fF from another region, its decap value becomes 7 fF.

Each virtual decap cell assigned to a cell instance has the following attributes:

`decap_value`, `power_net`, `ground_net`

The virtual decoupling capacitance of a cell instance is ignored during dynamic rail analysis if

- Either or both of the `power_net` and `ground_net` for the virtual decap cell do not match the supply nets being analyzed during combined analysis.
- Neither the `power_net` nor `ground_net` for the virtual decap cell matches the supply net analyzed during net-based analysis.

[Table 13-3](#) lists the options that are commonly used for inserting virtual decap cells.

Table 13-3 *Commonly Used Options for the `insert_virtual_decap_cells` Command*

Option	Description
<code>-total_capacitance cap_value</code>	Specifies the total capacitance of virtual decap cells to be inserted to the region. The <code>cap_value</code> value is in the library unit.
<code>-bounding_box rectangle</code>	Specifies the rectangle region (in microns) where decap cells are to be inserted. The rectangle is specified as a list of four coordinates, for example {left bottom right top}, for the rectangle box.
<code>-connect_to_power net_name</code>	Specifies a power supply net which connects to the decap cells.
<code>-connect_to_ground net_name</code>	Specifies a ground supply net which connects to the decap cells.

The following example inserts virtual decap cells.

```
pr_shell> calculate_rail_voltage [list VDD VSS] -dynamic
pr_shell> start_gui
pr_shell> insert_virtual_decap_cells \
    -bounding_box { 30.290 -0.695 45.560 21.535 } \
    -connect_to_power VDD \
    -connect_to_ground VSS -total_capacitance 13
pr_shell> report_virtual_decap_cells -connect_to_power VDD \
    -connect_to_ground VSS -verbose
pr_shell> calculate_rail_voltage [list VDD VSS] -dynamic
```

See Also

- [Rail Analysis With Decoupling Capacitors](#)
- [Virtual Decap Insertion Flow](#)

Voltage Drop Analysis With Regulators

During dynamic rail analysis, PrimeRail considers on-chip voltage regulators and the resulting transient effects when these voltage regulators are active. PrimeRail also generates a reduced circuit netlist, which describes information for the voltage regulator cell instances, the supply net parasitics, and a package model.

PrimeRail then invokes the HSPICE simulation tool with this reduced circuit netlist for circuit simulation. When the circuit simulation is complete, the tool generates voltage and current waveforms at the input, output, and ground pins of each voltage regulator cell instance. You can check for voltage violations by displaying a voltage drop map. To view the generated voltage and current waveforms of each voltage regulator and the PG pins, open the generated waveform files in a waveform viewer that supports the FSDB format.

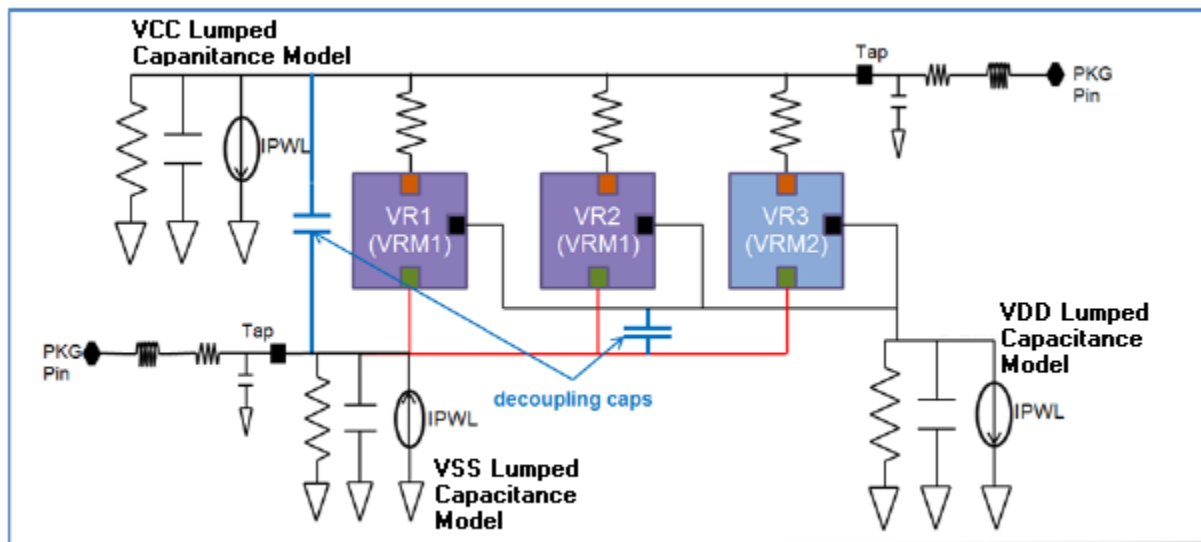
See Also

- [Using Voltage Regulators in Dynamic Rail Analysis](#)
- [Voltage Regulator Analysis Flow](#)

Using Voltage Regulators in Dynamic Rail Analysis

Voltage regulators maintain a consistent supply voltage value for a regulated power domain. When analyzing dynamic voltage drops for a regulated power domain, PrimeRail captures load regulation and line regulation characteristics for the regulator cells. When analyzing a voltage regulator model, the capacitance in the lumped capacitance model does not include the decoupling capacitances between VCC and VSS, and those between VDD and VSS (see [Figure 13-5](#)). Resistance between VCC and voltage regulators is a minimum path resistance. For voltage regulators, the output pin voltage and the input pin current waveforms are both imported from the HSPICE tool.

Figure 13-5 Analyzing a Voltage Regulator Model



See Also

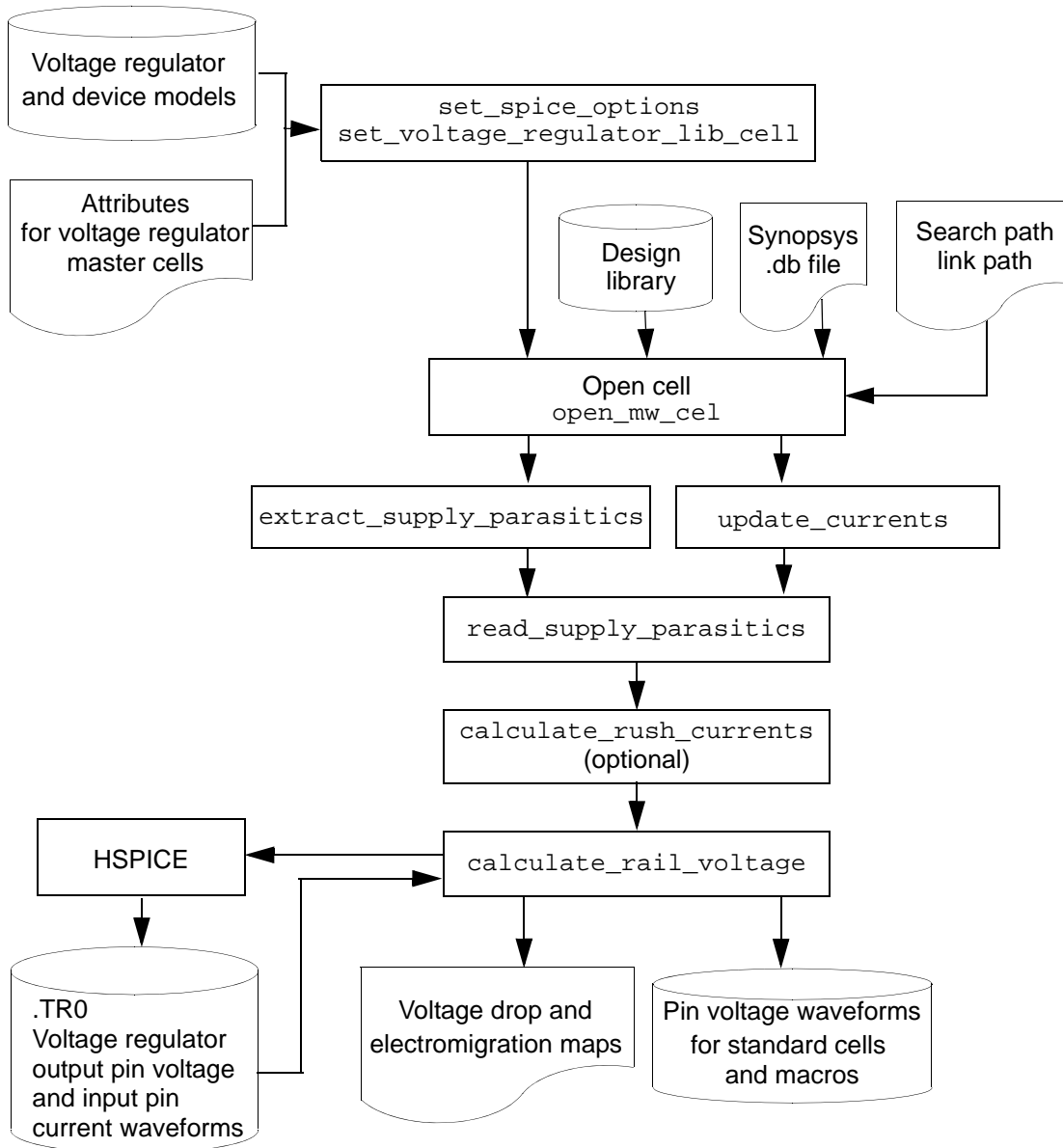
- [Voltage Drop Analysis With Regulators](#)

- [Voltage Regulator Analysis Flow](#)

Voltage Regulator Analysis Flow

Figure 13-6 illustrates the analysis flow when considering voltage regulators during dynamic rail analysis.

Figure 13-6 The Voltage Regulator Rail Analysis Flow



To perform a voltage regulator voltage drop analysis,

1. Run the `set_voltage_regulator_lib_cell` command to specify the FRAM view pin names and voltages for each voltage regulator master cell.

To examine the voltage regulator settings, run the `report_voltage_regulator_lib_cell` command.

For more information about the commands, see the man pages.

2. Run the `set_spice_options` command to specify HSPICE simulation settings, including

- Sub-circuit definitions and device models for the voltage regulators
- HSPICE executable path
- Simulation setup files
- Pin name mappings if pin names in the FRAM view do not match with those in the SPICE sub-circuit file

To examine the simulation-related settings, run the `report_spice_options` command. For more information about the commands, see the man pages.

3. Run the `open_mw_cel` command. A new library cell is automatically created for each voltage regulator master cell that is defined with the `set_voltage_regulator_lib_cell` command. Each voltage regulator cell instance is recognized and linked to its corresponding new library cell.
4. Continue with the steps in the dynamic rail analysis flow, including power and ground net extraction, timing and power analysis, and rail analysis.

Output Files

When rail analysis is complete, PrimeRail saves the following output files under the run directory:

- `LumpCoreCap.sp`
The file that records the total wire and intrinsic capacitance of the PG net.
- `LumpCoreCurrent.sp`
The file that records the total instance switching current on the PG net.
- `LumpCoreLeakR.sp`
The file that records the leakage current from the PG net to the ground.
- `Core_VoltageRegulator_cosim.sp`
The SPICE deck for cosimulation.

- Core_VoltageRegulator_cosim.*
The SPICE cosimulation output files.
- design.vr.fsdb
The file that records transient waveforms of dynamic rail analysis.

Script Examples

Here is an example of the script for running a voltage regulator rail analysis.

```
# setup voltage regulator lib cell before open_mw_cel
set_voltage_regulator_lib_cell test \
-input_power_pin {VCC 3.0} \
-output_power_pin {VDD 1.35} \
-reference_pin {VREF 1.35} \
-ground_pin {VSS 0.0} \
-other_io_pins {PIN1 3.0 PIN2 0.0 PIN3 0.0}

report_voltage_regulator_lib_cell > vr_rpt.txt

# enable voltage feedback flow for voltage drop regulator aware analysis
set_rail_enable_voltage_feedback true

...
open_mw_cel -library design.mw top
# setup SPICE options before calculate_rail_voltage
set_spice_options \
  -spice_exec "/global/apps/hspice_2013.03/hspice/bin/hspice" \
  -top_spice_ckt_file "V_test.net" \
  -setup_file "vr_setup_file.sp" \
  -include_path "./validation/VRmodels" \
  -hdl_path "./validation/VRmodels"

#CAVEAT: pin name mismatch between Milkyway FRAM and SPICE .subckt
# .subckt port names have 'VR_' in front of the pin names
set_spice_options \
  -pin_name_mapping {VCC VR_VCC VDD VR_VDD VSS VR_VSS \
                    VREF VR_VREF}
report_spice_options

...
# save transient waveform with the -fsdb_file option
calculate_rail_voltage -dynamic {VDD VCC VSS} -fsdb_file design.vr.fsdb
```

See Also

- [Voltage Drop Analysis With Regulators](#)
- [Using Voltage Regulators in Dynamic Rail Analysis](#)
- [Static Rail Analysis Flow](#)

Advanced Electromigration Analysis

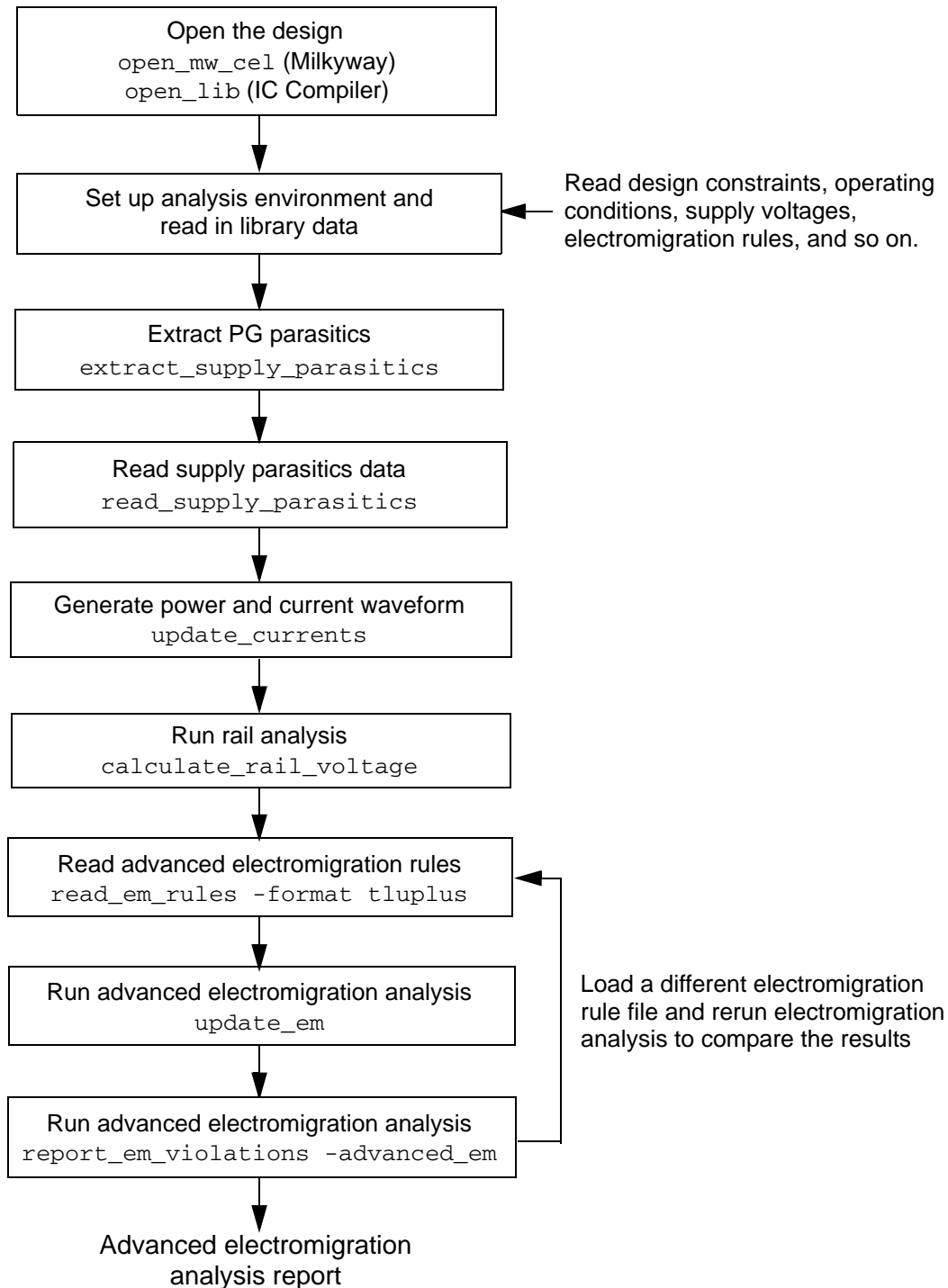
During electromigration analysis, PrimeRail finds the current that violates the current density and temperature limits defined in the electromigration file. For advanced electromigration analysis, PrimeRail reports a current violation by considering length dependence, current direction and power rail recognition based on the limits defined in the advanced electromigration file.

Run the `read_em_rules -format file_name` command (or choose Rail > Read EM Rules in the GUI) to load an advanced electromigration rule file. PrimeRail supports advanced electromigration rules in the TLUPlus format.

For more information about loading electromigration rule files, see [Reading Electromigration Rules](#).

[Figure 13-7](#) illustrates the steps in the advanced electromigration analysis flow.

Figure 13-7 Required Steps for the Advanced Electromigration Analysis Flow



To perform an advanced electromigration analysis,

1. Run the `open_mw_cel` command to open the design.
2. Set up the environment and load required library and input data, such as ideal voltage drop sources, timing constraints, signal parasitics, switching activity, and so on.

For more information, see [Preparing Input Data](#).

3. Run the `extract_supply_parasitics` command to perform extraction on the specified power or ground nets.

For more information, see [Extracting Supply Net Parasitics](#).

4. Run the `update_currents` command to perform timing and power analysis and to generate instance-specific current waveforms.

For more information, see [Extracting Supply Net Parasitics](#).

5. Run the `calculate_rail_voltage` command to perform rail analysis.

6. Run the `read_em_rules -format t1plus test.t1plus` command to read in the advanced electromigration rules.

For more information, see [Reading Electromigration Rules](#).

7. Run the `update_em` command to perform electromigration analysis using the loaded advanced electromigration rules.

8. Run the `report_em_violations` command with the `advanced_em` option to check the current violations that are found in the design. Repeat steps 6 to 8 if you want to load another electromigration rule file and compare the results.

When advanced electromigration analysis is complete, check the analysis results via maps or error cells. The analysis report lists all electromigration rules used when performing the analysis in the header section, along with wire and via segments violating current limits and fixing recommendations.

Example

The following is an example of the advanced electromigration report.

```
Information: temperature 110C from the nominal operating condition in the
first library in the link path is used for EM analysis
*****
Report : EM violations
Design : test
Version: <Version>
Date : <Date>
*****
Section: EM Rules
-----
Source: Build in
```

```

Units:
WIDTH : 1e-06 Meter
AREA : 1e-12 Meter**2
CURRENT : 0.001 Amp
Temperature: Degree Celsius
Variables:
L : length
W : width
LB : the length of a via's bottom metal
WB : the width of a via's bottom metal
LU : the length of a via's upper metal
17
WU : the width of a via's upper metal
...
Layer: metall
Mode : average
Temperature Rating Factor
...
EM Rule
IF(Power_Grid) { 3.0 * 1.0 * ( Width - 0.005) }
IF(Length > 4.0 AND Width < 1.0) { 1.0 * 1.0 * ( Width - 0.005 ) }
...
Layer: via8
Mode : average
Temperature Rating Factor
...
EM Rule
IF((Area == 0.09 ) AND ( Lb <= 4 ) AND ( DOWNSTREAM )) { 4.0 * 2.0 }
...
-----
Section: EM Violations
-----
Scale Factor : 1.000
Temperature : 110 C
Units:
WIDTH : 1e-06 Meter
AREA : 1e-12 Meter**2
CURRENT : 0.001 Amp
Keys:
L_NAME : layer name
L_NUM : layer number
THRESHOLD : EM threshold for current
W|A : width or area (rule dependent) 18
REC W|A : recommended width or area (rule dependent) needed to resolve
violation

* NET: VDD
L_NAME L_NUM MODE CURRENT (>THRESHOLD) W|A REC W|A (LEFT,
BOTTOM) (RIGHT, TOP)
-----
metall 11 average 293.170 (>280.000) 0.160 0.168
(1508.350, 1310.800) (1508.860, 1310.960)
metall 11 average 286.008 (>280.000) 0.160 0.163

```

```
(1508.350, 1410.800) (1508.860, 1410.960)
...
via8 28 average 345.135 (>280.000) 0.160 0.197
(1508.370, 2384.240) (1513.820, 2384.400)
via8 28 average 313.528 (>280.000) 0.160 0.179
(1508.370, 2484.240) (1513.820, 2484.400)
* Summary for NET: VDD
* # of total violations: 12
* metall violations: 8
* via8 violations: 4
```

The following is an example of script file for running an advanced electromigration analysis.

```
# initial design setup
set ::search_path {.}
lappend ::search_path ../lib/db
lappend ::search_path ../tech
set ::link_path {}
lappend ::link_path *
lappend ::link_path f_ccs.db f_ccs.db f_ccs.db test.db

# open_mw_cel
open_mw_cel -library ./MW_CTOP test_top
19

# setup IR-drop and EM analysis environment
create_taps -import ./scripts/pg_file

# set_voltage commands
set_voltage 0.99 -object_list { VDD }
set_voltage 0 -object_list { VSS }

# read timing constraints
read_sdc ../design/test_top.sdc

# set operating conditions
set_operating_conditions -max ff -min ff

# power analysis
update_currents -static

# P/G net extraction
set_extract_supply_parasitics_options \
-max_tluplus ../tech/typical.tluplus \
-tech2itf_map ../tech/star.map \
-temperature 25.000
extract_supply_parasitics [list VDD VSS]

# IR-drop analysis
read_supply_parasitics [ list VDD VSS ]
calculate_rail_voltage [list VDD VSS] -static

# advanced EM analysis
```

```
read_em_rules -format tluplus test.tluplus
update_em
report_em_violations ./advanced_EM_rules.txt
#quit
```

See Also

- [Reading Electromigration Rules](#)

What-If Analysis

Calculating voltage drop and current violations in PrimeRail consists of multiple analysis stages, such as parasitic extraction, power and current waveform calculation, and rail analysis. You can use the what-if analysis capability to experiment with different elements to see the effects on IR drop and electromigration without rerunning the entire analysis flow. You can even switch between the results before and after what-if constraints are applied.

For example, you can assign a power or current value with the `set_rail_power` or `set_rail_current` command and rerun the `update_currents` command to compare the results. You can also reuse the analysis data from another analysis run, such as power and extraction data, and rail analysis results. Inserting virtual decoupling cells is also supported.

In PrimeRail, you can do what-if analysis by

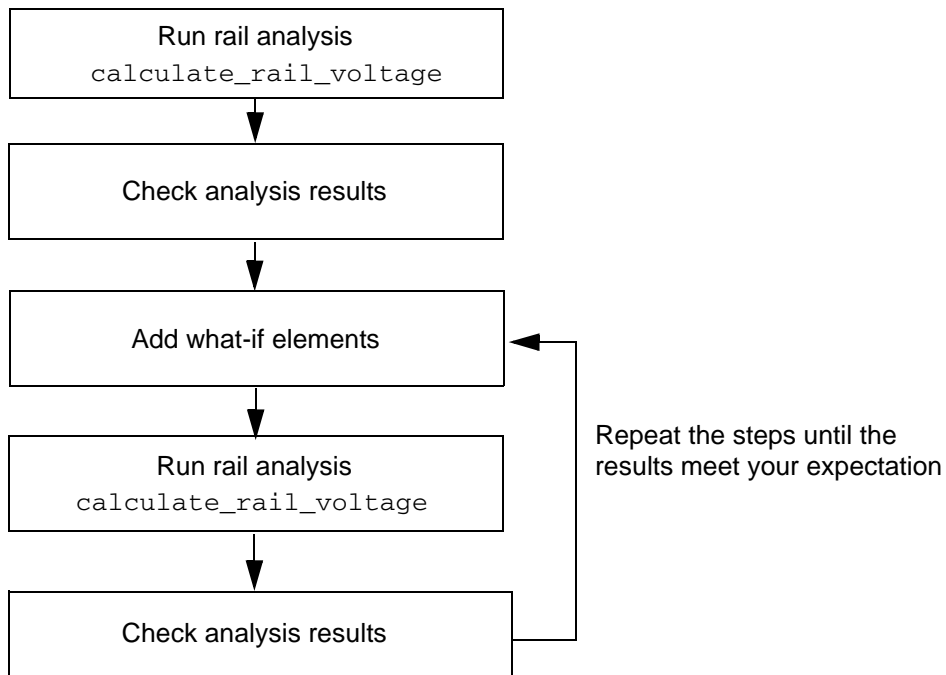
- Assigning a user-specified power value with the `set_rail_power` command. Use the command when you want to use a power value other than the one generated by the `update_currents` command, or if no correct total power is available.
For more information, see [Power Analysis With Assigned Power](#).
- Assigning a user-specified current value with the `set_rail_current` command. Use the command when you want to assign a current value to the cell directly.
For more information, see [Waveform Calculation With Assigned Current](#).
- Scaling power with the `scale_rail_power` command.
When rail analysis is complete, you can rerun rail analysis with a scaling factor or value to estimate how much total power can be reduced without changing the actual power consumption result. This temporarily affects the power that is used to calculate the voltage drop at power and ground pins when you run the `calculate_rail_voltage` command; it does not change the actual saved power.
For more information, see [Power Scaling](#).
- Running the `insert_virtual_decap_cells` command to insert virtual decoupling capacitors in a region where huge voltage drops might occur.
For more information, see [Rail Analysis With Decoupling Capacitors](#).

- Inserting or modifying physical objects in the layout, including inserting external resistors or non-ideal voltage sources.

For more information, see [Adding External Resistors or Non-Ideal Voltage Sources](#).

[Figure 13-8](#) illustrates the steps for adding what-if elements in the analysis flow.

Figure 13-8 Adding What-If Elements in the Analysis Flow



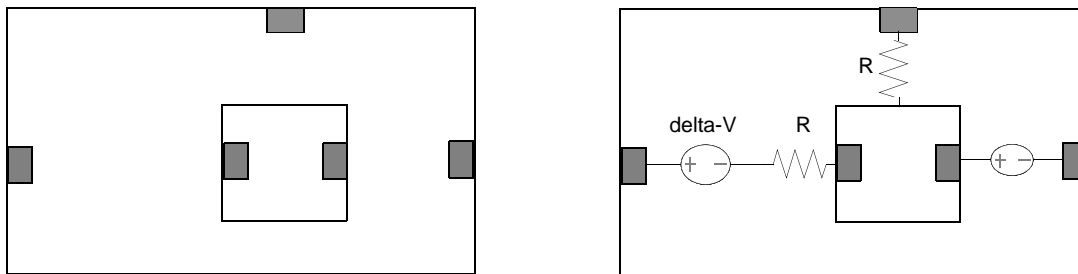
Adding External Resistors or Non-Ideal Voltage Sources

If the power and ground nets of your design are not connected to pads or pins or you are conducting a block-level analysis, you can assign non-ideal voltages and external resistors to pins or pads in your design, predicting what the actual voltages will be for calculating the current consumption of the design. The non-ideal voltage sources can be controlled by a resistance value or a voltage drop or rise, or by both.

If you have not defined pins or finalized the locations of the pad cells in your design, you can insert non-ideal voltage sources and external resistors through a user-defined tap file, where the voltage is specified by coordinate and layer number.

[Figure 13-9](#) is an example of a block at the top level before and after inserting non-ideal voltages and external resistors for the what-if scenario.

Figure 13-9 An Example of Inserting Non-Ideal Voltages and External Resistors to Pins or Pads



In PrimeRail, you can insert non-ideal voltages using the `create_taps` command. You can either manually specifying the tap locations with the command or in an ASCII file that is imported with the `create_taps -import file_name` command.

To insert a virtual resistor, define tap xy locations, layer numbers and supply nets in an ASCII file and then import it to the tool using the `create_taps -import` command. Alternatively, use the `set_tap_package_model` command to assign a fixed set of electrical characteristics to the taps, so these effects are considered during rail analysis.

For a detailed description about adding non-ideal voltages or external resistors, see [Manually Specifying Tap Locations](#), [Importing a Tap File](#), and [Creating Taps With Packages](#).

A

SiliconSmart Script Examples

The SiliconSmart tool is the Synopsys characterization and modeling solution focused on creating cell libraries for today's advanced nanometer processes. It supports multiple modeling formats, such as Nonlinear Delay Model (NLDM) format for timing and Nonlinear Power Model (NLPM) for power analysis, Composite Current Source Models (CCS) for timing, power and noise analysis. These modeling extensions enable you to generate consistent models across multiple tools and multiple vendors.

This appendix includes SiliconSmart script examples for characterizing libraries in the CCS power format, as described in the following topics:

- [SiliconSmart configure.tcl Script Example](#)
- [SiliconSmart Run Script Example](#)

SiliconSmart configure.tcl Script Example

A configure.tcl script file contains parameters and definitions needed for setting the operation condition of library characterization.

The following is an example of the configure.tcl file used for characterizing libraries and creating CCS power models for PrimeRail dynamic rail analysis. For a complete list of parameters and definitions, see the related chapter in *SiliconSmart Online Help*.

```
#####
# OPERATING CONDITIONS DEFINITION
#####
#
# Create one or more operation conditions here.
#
create_operating_conditions TP_08_n40
set_opc_process TP_08_n40 {
{.inc "/test/PR_LIB/spice/model"}
}
add_opc_supplies TP_08_n40 VDD 0.78
add_opc_grounds TP_08_n40 VSS 0.0
set_opc_temperature TP_08_n40 -40
#

#####
# GLOBAL CONFIGURATION PARAMETERS
#####
define_parameters default {
set nmos_model_names {N n105 }
set pmos_model_names {P p105 }

# List of operating conditions as defined by create_operating_conditions
set active_pvts { TP_08_n40 }

# If using IBIS, one operating condition must be specified in
ibis_typ_pvt
# set ibis_typ_pvt op_cond

# FINESIM
#set simulator finesim
#set simulator_cmd {finesim_spice -w input_deck \
-o listing_file >&/dev/null}
# FINESIM EMBEDDED
# set simulator finesim_embedded

# HSPICE
set simulator hspice
set simulator_cmd {hspice input_deck -o listing_file}

# HSPICE (client/server mode)
```

```

# set simulator hspice_cs
# set simulator_cmd {hspice -CC input_deck \
    -port port_num -o listing_file}

# SPECTRE
# set simulator spectre6
# set simulator_cmd {spectremdl -tab -batch mdl_file \
    -design input_deck listing_file >&/dev/null}

# ELDO
# set simulator eldo# set simulator_cmd {eldo -compact -i input_deck \
    listing_file >&/dev/null}

# MSIM
# set simulator msim
# (csh)
# set simulator_cmd {msim -hsp -i input_deck -o listing_file >&/dev/null}
# (sh)
# set simulator_cmd {msim -hsp -i input_deck -o listing_file 2>/dev/null}

# Default simulator options for Finesim, Hspice, Spectre, Msim, and Eldo
set simulator_options {
    "common,finesim: probe=1 finesim_output=tr0
    finesim_mode=spicel finesim_method=gearv"
    "common,hspice: probe=1 runlvl=5 numdgt=7 measdgt=7
    acct=1 nopage method=gear"
    "common,spectre6: compression=yes step=10ps
    maxstep=1ns relref=allglobal"
    "common,spectre6: method=trap lteratio=4 gmin=1e-18
    autostop=0 save=none"
    "common,msim: probe=1 accurate=1"
    "common,eldo: gmindc=1n gmin=1p itl1=500 ingold=1
    numdgt=4 measout=0 cptime=18000 relvar=0.01"
    "op,eldo: dv=0.5 method=gear"
    "tran,eldo: brief=0 relvar=0.001"
    "optimize,eldo: lvltim=3 relvar=0.001"
    "power,eldo: method=gear"
}

# Simulation resolution
set time_res_high 1e-13

# Controls which supplies are measured for power consumption
set power_meas_supplies { VDD }
# List of ground supplies used (required for Functional Recognition)
set power_meas_grounds { VSS }
# Specify which multi rail format to be used in Liberty model; none, # v1,
or v2.
set liberty_multi_rail_format v2
# LOAD SHARE PARAMETERS
# job_scheduler: 'lsf' (Platform), 'grid' (SunGrid), or 'standalone'#
(local machine)

```

```

# set job_scheduler lsf
set job_scheduler grid

# set job_scheduler standalone
set run_list_maxsize 150
set normal_queue {-P bnormal -V -l arch=glinux,mem_free=32G,qsc=f|g|h}
set scheduler_poll_time 60
set char_engine_max_lifespan 2880
set simulation_tmpdir /var/tmp

# Archive conditions
set archive_condition_on_success compress
set archive_condition_on_failure yes
set enable_cache on
}

#####
# DEFAULT PINTYPE PARAMETERS
#####
pintype default {
    set logic_high_name VDD
    set logic_high_threshold 0.7
    set logic_low_name VSS
    set logic_low_threshold 0.3

    set prop_delay_level 0.5
    set slew_based_margin 0.0

# Number of points in current waveforms;
# To use 1 % error instead of default 0.5% with a value of 0.005
# set ccs_power_max_current_error 0.01
# Number of slew and load indexes
# (when importing with -use_default_slews -use_default_loads)
    set numsteps_slew 5
    set numsteps_load 5
    set constraint_numsteps_slew 3

# Operating load ranges
    set smallest_load 10e-15
    set largest_load 90e-15

# Operating slew ranges
#set smallest_slew 10e-12
#set largest_slew 1.2e-9
    set smallest_slew 1e-12
    set largest_slew 10.0e-9

#set max_tout 1.0e-9

# Automatically determine largest_load based on max_tout; off or on
set autorange_load off

# Noise of points in for noise height

```

```

set numsteps_height 8

# Input noise width
set numsteps_width 5

# driver model: pwl, emulated, active, active-waveform, custom
set driver_mode emulated

# driver cell name (relevant only when driver_mode is "active")
set driver pwl
set smc_constraint_style relative-degradation
set smc_degrade 0.1
set smc_degrade_absolute 5e-12
set glitch_high_threshold 0.9
set glitch_low_threshold 0.1
set constraint_resolution 2e-12
}

#####
# LIBERTY MODEL GENERATION PARAMETERS
#####

# Add Liberty header attributes here for use with
# "model -create_new_model"
define_parameters liberty_model {
    set technology "cmos"
    set delay_model "table_lookup"

    set time_unit "lns"
    set pulling_resistance_unit "lkohm"
    set voltage_unit "1V"
    set current_unit "1uA"
    set leakage_power_unit "1pW"

    set default_inout_pin_cap 0.0
    set default_input_pin_cap 0.0
    set default_output_pin_cap 0.0
    set default_cell_leakage_power 0.0
    set default_max_capacitance 9999.0
    set default_fanout_load 1.0
    set default_max_fanout 9999.0
    set default_leakage_power_density 0.0
    set default_max_transition 2.0

    set input_threshold_pct_fall 50
    set input_threshold_pct_rise 50
    set output_threshold_pct_fall 50
    set output_threshold_pct_rise 50

    set slew_derate_from_library 1.0
    set slew_lower_threshold_pct_fall 20
    set slew_lower_threshold_pct_rise 20

```

```
set slew_upper_threshold_pct_fall 80
set slew_upper_threshold_pct_rise 80
}

##### VALIDATION PARAMETERS
#####

define_parameters validation {
# Add validation parameters here
set capacitance_absolute_tolerance 0.0001
set capacitance_relative_tolerance 0
set capacitance_product_tolerance 0

set delay_absolute_tolerance 0.002
set delay_relative_tolerance 0.02
set delay_product_tolerance 0
set slew_absolute_tolerance 0.002
set slew_relative_tolerance 0.02
set slew_product_tolerance 0
set zen_absolute_tolerance 0.002
set zen_relative_tolerance 0.02
set zen_product_tolerance 0
set zdis_absolute_tolerance 0.002
set zdis_relative_tolerance 0.02
set zdis_product_tolerance 0

set setup_absolute_tolerance 2.0
set setup_relative_tolerance 0.05
set setup_product_tolerance 0 set hold_absolute_tolerance 2.0
set hold_relative_tolerance 0.05
set hold_product_tolerance 0
set recovery_absolute_tolerance 2.0
set recovery_relative_tolerance 0.05
set recovery_product_tolerance 0
set removal_absolute_tolerance 2.0
set removal_relative_tolerance 0.05
set removal_product_tolerance 0
set mpw_absolute_tolerance 2.0
set mpw_relative_tolerance 0.05
set mpw_product_tolerance 0

set energy_absolute_tolerance 0.001
set energy_relative_tolerance 0.1
set energy_product_tolerance 0
set leakage_absolute_tolerance 0.005
set leakage_relative_tolerance 0.05 set leakage_product_tolerance 0
}
```

SiliconSmart Run Script Example

The SiliconSmart tool can be invoked in batch mode by providing the name of a Tcl script as an argument on the command-line. When done, SiliconSmart will read the Tcl file and execute each of the commands. This can be used to automate the process of configuring, characterizing, and modeling a whole library of cells.

The following script example automates these processes for characterizing CCS power libraries in SiliconSmart.

```
# Run script run.tcl to characterize CCS models using SiS
set charpt chp

# Include a list of cells to be characterized
source ./celllist
create $charpt
set_log_file $charpt/siliconsmart.log
exec cp configure.tcl $charpt/config/configure.tcl
set_location $charpt
set_config_opt enable_parasitic_sweep 1
set_config_opt numsteps_voltage 5
import -liberty reference_seed.lib -netlist_dir subckt -ext .spc $cells
configure -power -ccs -ccs_power $cells
characterize $cells
model -nocompact -ccs_power -lib_name my_SiS_lib -output my_SiScsp

# Use -create_new_model to generate new CCSP models with the compact
# CCS liberty as inputs
model -create_new_model -nocompact -timing -power -ccs -ccs_power \
    -lib_name my_SiS_lib -output out_test_ccs
```


Glossary

CONN View

Generated by the `create_rail_macro_model` command. CONN views represent the physical information of power and ground network inside the GDSII or hard macros. For memory cells and complex I/O cells, CONN views need to be created if the FRAM view does not contain detailed PG shapes. If the power consumed by I/O cells is negligible, it is recommended not to create CONN views for I/O cells. Also, skipping bitcells is recommended if the cell being analyzed is a memory cell.

Current Source File (CSF)

Generated by the `create_rail_macro_model` command. A current source file records how current is drawn inside the cell in ASCII format. By default, the power consumption of a macro cell is assumed to be evenly provided by all available power pins. To improve rail analysis accuracy, you might want to distribute the current demands accurately by creating a current source file and loading it during rail analysis.

Electromigration

The movement of metal molecules due to high current density, which leads to metal opens and shorts in the chip. High electromigration greatly reduces reliability and chip life span.

Hard Macro

A block that is generated in a methodology other than place and route and is brought into the Milkyway database as a GDSII file by use of the `auStreamIn` command. In PrimeRail, a hard macro has to be from a GDSII source.

Macro Model

Created to improve performance or conceal detailed intellectual property (IP) vendor data.

Map

The graphical view of analysis results, including current violation, IR drop violation, power distribution, and parasitic data. Different colors can be assigned to help visualize

problem areas, like indicating various step levels of voltage rise and drop or current density limits, for example. Visibility control over layout layers and design hierarchy levels is also supported.

Minimum Path Resistances

The resistance value on the smallest resistive path from the node to one of the design's boundary nodes, that is, ideal voltage sources (taps). The ideal voltage sources can be power or ground pins, user-defined taps, or packagings. Because this value represents only the resistance on a single path, it does not need to be exactly the same as the effective resistance value from the node to the global ground net. It represents an upper bound of the effective resistance of a node to the ground net.

Multithreshold-CMOS (MTCMOS) Circuit

The circuit that contains high and low threshold (V_t) transistors that can be used to reduce leakage power in low-power and high-performance applications. High threshold voltage CMOS transistors are used to disconnect the power grid to the core at the stand-by state, resulting in a very low leakage current as set by the high threshold voltage of the series transistor.

Pin

The input and output of cells within a design (such as gates and flip-flops). The ports of a subdesign are pins within the parent design.

Port

The inputs and outputs of a design. Port direction is designated as input, output, or bidirectional.

A logical connection from one block (the child) to the parent block in which this child is placed. For example, there might be one power port named VDD and one named VCC.

Power Management Cell

Power management cells act as switch cells. When turned on, they allow currents to flow through; when turned off, the cells act as open circuits. The power management cells are used to reduce leakage currents in a design. A multithreshold-CMOS cell is an example of a power management cell.

Power Management Location (PML) File

The file that describes the on-resistance source and drain locations of power management cells.

Rail Macro Model

A rail macro model is an abstract representation of the internal physical model (including PG structure and currents) and the transistor-level electrical model of a hard macro cell. By making a soft macro into a hard macro, many data objects inside the soft macro are no longer needed for analysis and thus freed from memory. This greatly reduces memory usage and improves performance of the rail analysis.

Synopsys Design Constraints (SDC) File

A design constraints file that is used during timing and power analysis.

Signal Net Parasitic

The SPEF or DSPF signal net parasitic used for timing and power analysis. Make sure this is consistent with database, which means the signal parasitic is generated from the same design that can be analyzed in PrimeRail.

Subblock

A block that is under the top cell.

SPICE Netlist

Contains .SUBCKT information for each of the leaf cells. The file is used for library characterization purpose. The .SUBCKT description should also have power and ground ports defined (sometimes, they are defined globally).

SPICE Model

Contains the model information for devices or transistors that are used in the SPICE netlist for each of the leaf cells.

TLUPlus Model

A Table Lookup Plus (TLUPlus) based model for resistance and capacitance calculations.

Value Change Dump (VCD) File

The file that is used for vector-based analysis. Make sure the delay information described in the file is not zero.

Verilog Netlist

A post-layout Verilog netlist used for the timing and power analysis. Make sure this netlist is consistent with the database (or you can choose to write it out from the database).

Voltage Drop

Voltage drop or IR drop is the reduction of voltage due to the currents passing through the resistance network. The impact occurs at points farther from the pad. Lower voltage supply to cells can lead to longer ramp times and changes to the switching threshold. Timing can be incorrect, causing poor performance.

Waveform

The shape and form of a signal, such as a wave moving across the surface of water or the vibration of a plucked string.

