

PrimeTime®

User Guide

Version J-2014.12, December 2014

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2014 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

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

Synopsis and certain Synopsis product names are trademarks of Synopsis, 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. Synopsis does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

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

Copyright Notice for the Command-Line Editing Feature

© 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the Line-Editing Library

© 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

Copyright Notice for the jemalloc Memory Allocator

© 2002-2013 Jason Evans <jasone@canonware.com>. All rights reserved.

© 2007-2012 Mozilla Foundation. All rights reserved.

© 2009-2013 Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND

ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the CDPL Common Module

© 2006-2014, Salvatore Sanfilippo. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Redis nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

About This Guide	xxxvi
Customer Support.....	xxxix
1. Introduction to PrimeTime	
PrimeTime Features	1-2
Types of Design Checking	1-2
Analysis Features	1-2
Supported Timing Models	1-3
PrimeTime Add-On Tools.....	1-4
Using PrimeTime in the Implementation Flow	1-4
Compatibility With Design Compiler and IC Compiler	1-6
Overview of Static Timing Analysis.....	1-7
Timing Paths	1-7
Delay Calculation.....	1-9
Cell Delay	1-10
Net Delay	1-10
Constraint Checking.....	1-10
Setup and Hold Checking for Flip-Flops.....	1-11
Setup and Hold Checking for Latches	1-13
Timing Exceptions	1-15
2. Getting Started	
Before You Begin	2-2
Setup Files	2-2

Starting a PrimeTime Session	2-2
Working With Licenses	2-4
Listing the Licenses in Use	2-4
Checking Out Licenses	2-4
Enabling License Queuing	2-5
Releasing Licenses	2-5
Entering pt_shell Commands	2-6
Tcl Packages and Autoload	2-6
TclPro Toolkit	2-7
Install TclPro Tools	2-7
Check the Syntax in Scripts With the TclPro Checker	2-7
Create Bytecode-Compiled Scripts With the TclPro Compiler	2-10
Debug Scripts With the TclPro Debugger	2-11
Support of the incr Tcl Extension	2-11
Getting Help on the Command Line	2-11
Using Tcl/Tk in PrimeTime	2-12
The PrimeTime Static Timing Analysis Flow	2-12
Ending a PrimeTime Session	2-16
Saving a PrimeTime Session	2-16
Exiting a PrimeTime Session	2-17
Accessing the Session History in the Command Log File	2-17
3. Managing Performance and Capacity	
High Capacity Mode	3-2
Fast Analysis Mode	3-2
Threaded Multicore Analysis	3-3
Configuring Threaded Multicore Analysis	3-3
Executing Commands in Parallel	3-4
The parallel_execute Command	3-5
The redirect -bg Command	3-5
The parallel_foreach_in_collection Command	3-5
Running Threaded Multicore Path-Based Analysis	3-7
Distributed Multi-Scenario Analysis	3-8
Definition of Terms	3-9

Overview of the DMSA Flow	3-11
Preparing to Run DMSA.....	3-11
DMSA Usage Flow	3-12
Distributed Processing Setup	3-17
Starting PrimeTime for Distributed Processing.....	3-17
Managing Compute Resources	3-18
Creating Scenarios.....	3-21
Specifying the Current Session and Command Focus.....	3-23
Checking the Setup	3-23
Executing Commands Remotely	3-23
Script Example	3-25
Common Image Generation	3-28
Modeling Support	3-29
Achieving Optimal Performance.....	3-29
Manipulating Variables.....	3-29
Master Context Variables	3-30
Slave Context Variables and Expressions	3-30
Setting Distributed Variables.....	3-30
Getting Distributed Variable Values	3-31
Merging Distributed Variable Values.....	3-32
Merged Reporting	3-35
Using Merged Reporting.....	3-35
Generating Merged Reports.....	3-35
Saving and Restoring Your Session.....	3-46
License Resource Management	3-46
Incremental License Handling	3-47
License Pooling	3-48
License Autoreduction	3-48
License Queuing.....	3-49
Database Support	3-49
Controlling Fault Handling	3-49
Merged Reporting Commands	3-50
Netlist Editing Commands	3-50
Using the remote_execute Command	3-51
Other Commands	3-51
Messages and Log Files	3-52
Interactive Messages	3-52
Progress Messages	3-52
User Control of Task Execution Status Messages	3-52
Error Messages	3-53

Warning Messages	3-53
Log Files	3-54
Command Output Redirection	3-55
Limitations	3-56
Single and Multiple Scenario Constraint Reports	3-56
Constraint Merging Operations.	3-56
Printing Styles.	3-57
Reporting Modes	3-59
DMSA Commands, Options, and Variables.	3-61
DMSA Command List.	3-61
DMSA report_timing Options	3-64
DMSA Variables	3-65
Disallowed Slave Commands	3-66
Memory and CPU Resource Usage Reports	3-66
Performance Profiling of Tcl Scripts	3-69
4. Design Data	
Search Path and Link Path	4-2
Reading Design and Library Data.	4-3
Reading Design Data in .ddc Format.	4-3
Reading Design Data in .db Format	4-4
Reading Verilog and VHDL Design Data	4-5
Using the PrimeTime Verilog Reader	4-5
Optional Preprocessor	4-6
Assign Statements and Synonyms.	4-6
Physical Verilog Power Rail Connections	4-7
PrimeTime Verilog Reader Limitations	4-7
Using the HDL Compiler Verilog Reader	4-8
Setup Files for the HDL Compiler Verilog Reader	4-8
Reading VHDL Design Files	4-9
Setup Files for the VHDL Compiler.	4-9
Using a Milkyway Database	4-10
Reading a Milkyway Database.	4-10
Writing a Milkyway Database.	4-11
Limitations When Reading Milkyway Format.	4-11
Removing Designs and Libraries	4-12

Setting the Current Design and Current Instance	4-12
Examples	4-14
Listing Design and Library Information	4-15
Linking the Design	4-16
Per-Instance Link Paths	4-17
Handling Incomplete Data	4-18
Link Errors	4-19
Checking the Consistency of Units With the set_units Command	4-20
Design Objects	4-20
State Objects	4-21
Netlist Objects	4-22

5. Design Constraints

Input Delays	5-3
Using Input Ports Simultaneously for Clock and Data	5-4
Output Delays	5-5
Drive Characteristics at Input Ports	5-5
Setting the Port Driving Cell	5-6
Setting the Port Drive Resistance	5-7
Setting a Fixed Port Transition Time	5-7
Displaying Drive Information	5-7
Removing Drive Information From Ports	5-8
Port Capacitance	5-8
Wire Load Models	5-9
Setting Wire Load Models Manually	5-9
Automatic Wire Load Model Selection	5-10
Setting the Wire Load Mode	5-11
Reporting Wire Load Models	5-12
Slew Propagation	5-12
Design Rule Constraints	5-13
Maximum Transition Time	5-13
Handling Slew in Multiple Threshold and Derating Environment	5-14
Converting Single-Float Slews Between Thresholds and Deratings	5-14

Maximum Transition Constraint Storage	5-15
Evaluating Maximum Transition Constraint	5-16
Example 1 - Setting a Maximum Transition Limit	5-17
Example 2 - Scaling the Maximum Transition by Slew Threshold	5-17
Example 3 - Scaling the Maximum Transition by Derating	5-17
Minimum and Maximum Net Capacitance	5-18
Constraining Rise and Fall Maximum Capacitance	5-18
Frequency-Based Maximum Capacitance Checks	5-19
Maximum Fanout Load	5-20
Fanout Load Values for Output Ports	5-21
Ideal Networks	5-21
Propagating Ideal Network Properties	5-21
Using Ideal Networks	5-22
Using Ideal Latency	5-24
Using Ideal Transition	5-24
6. Clocks	
Clock Overview	6-2
Specifying Clocks	6-2
Creating Clocks	6-3
Creating a Virtual Clock	6-4
Selecting Clock Objects	6-4
Applying Commands to All Clocks	6-4
Removing Clock Objects	6-4
Specifying Clock Characteristics	6-5
Setting Clock Latency	6-5
Setting Propagated Latency	6-5
Specifying Clock Source Latency	6-6
Setting Clock Uncertainty	6-8
Dynamic Effects of Clock Latency	6-9
Estimating Clock Pin Transition Time	6-10
Checking the Minimum Pulse Width	6-11
Checking the Minimum Period	6-12
Using Multiple Clocks	6-12
Synchronous Clocks	6-13
Asynchronous Clocks	6-15

Exclusive Clocks	6-15
Removing Clocks From Analysis	6-20
Clock Sense	6-21
Using Pulse Clocks	6-25
Constraining Pulse Widths in the Fanout of Pulse Generator Cells	6-27
Constraining Transition Times for Pulse Generator Cells	6-29
Timing PLL-Based Designs	6-30
Usage for PLL Timing	6-31
Sequential Cells on a Feedback Path	6-31
PLL Drift and Jitter	6-32
CRPR Calculations for PLL Paths	6-33
Reporting and Timing Checks	6-34
Requirements for PLL Library Cells	6-34
Specifying Clock-Gating Setup and Hold Checks	6-35
Disabling or Restoring Clock-Gating Checks	6-38
Specifying Internally Generated Clocks	6-38
Specifying a Divide-by-2 Generated Clock	6-39
Creating a Generated Clock Based on Edges	6-40
Creating Clock Senses for Pulse Generators	6-40
Creating a Divide-by Clock Based on Falling Edges	6-42
Shifting the Edges of a Generated Clock	6-44
Multiple Clocks at the Source Pin	6-45
Selecting Generated Clock Objects	6-45
Reporting Clock Information	6-46
Removing Generated Clock Objects	6-46
Generated Clock Edge Specific Source Latency Propagation	6-46
Clock Mesh Analysis	6-47
Overview of Clock Mesh Analysis	6-47
Usage Flow	6-49
Clock Network Simulation Commands	6-52
sim_setup_simulator	6-52
sim_setup_library	6-53
sim_setup_spice_deck	6-53
sim_validate_setup	6-54
sim_analyze_clock_network	6-55

7. Timing Paths and Exceptions

Path Groups	7-2
Path Timing Reports	7-2
Path Timing Calculation	7-5
Path Specification Methods	7-6
Multiple Through Arguments	7-7
Rise/Fall From/To Clock	7-7
Reporting of Invalid Startpoints or Endpoints	7-12
Timing Exceptions	7-12
Timing Exception Overview	7-13
Single-Cycle (Default) Path Delay Constraints	7-13
Path Delay for Flip-Flops Using a Single Clock	7-14
Path Delay for Flip-Flops Using Different Clocks	7-14
Setting False Paths	7-19
Setting Maximum and Minimum Path Delays	7-20
Setting Multicycle Paths	7-21
Specifying Exceptions Efficiently	7-26
Exception Order of Precedence	7-28
Exception Type Priority	7-28
Path Specification Priority	7-29
Reporting Exceptions	7-30
Reporting Timing Exceptions	7-31
Reporting Exceptions Source File and Line Number Information	7-34
Checking Ignored Exceptions	7-36
Removing Exceptions	7-37
Transforming Exceptions	7-38
Exception Removal	7-40
Exception Flattening	7-42

8. Operating Conditions

Operating Conditions	8-2
Interconnect Model Types	8-2
Setting Operating Conditions	8-4
Creating Operating Conditions	8-5
Operating Condition Information	8-5

Operating Condition Analysis Modes	8-6
Minimum and Maximum Delay Calculations	8-7
Minimum-Maximum Cell and Net Delay Values	8-9
Setup and Hold Checks	8-10
Path Delay Tracing for Setup and Hold Checks	8-10
Setup Timing Check for Worst-Case Conditions	8-11
Hold Timing Check for Best-Case Conditions	8-12
Path Tracing in the Presence of Delay Variation	8-13
Specifying the Analysis Mode	8-14
Single Operating Condition Analysis	8-14
On-Chip Variation Analysis	8-15
Using Two Libraries for Analysis	8-17
Setting Derating Factors	8-18
Clock Reconvergence Pessimism Removal	8-22
On-Chip Variation Example	8-22
Reconvergent Logic Example	8-23
Minimum Pulse Width Checking Example	8-24
Using CRPR Commands	8-25
CRPR and Crosstalk Analysis	8-28
CRPR With Dynamic Clock Arrivals	8-29
Transparent Latch Edge Considerations	8-30
Reporting CRPR Calculations	8-30
Clock On-Chip Variation Pessimism Reduction	8-31
9. Delay Calculation	
Overview of Delay Calculation	9-2
Nonlinear Delay Models	9-3
Composite Current Source Timing Models	9-4
Pin Capacitance Reporting	9-6
Guidelines for Characterizing Design Rule Constraints	9-7
Resolving the CCS Extrapolation Warning Message (RC-011)	9-8
CCS Cross-Library Scaling and Exact Matching	9-8
Defining Library Groups for Scaling	9-9
Defining Library Groups for Exact Matching	9-9

Guidelines for Scaling and Exact Matching	9-10
Scaling Interpolation for Timing Constraints	9-13
Scaling for Design Rule Constraints	9-14
Scaling for Multirail Cells	9-14
Fast Multidrive Delay Analysis	9-16
Parallel Driver Reduction	9-17
Invoking Parallel Driver Reduction	9-19
Working With Reduced Drivers	9-19
Path-Based Timing Analysis	9-20
Setting Recalculation Limits	9-23
CCS Receiver Model for Path-Based Analysis	9-24
Multi-Input Switching Analysis	9-24
Enabling Multi-Input Switching Analysis	9-25
Specifying the Coefficients for Multi-Input Switching Analysis	9-25
Configuring the Window Alignment and Overlap Checking	9-27
Multi-Input Switching Analysis Flow Example	9-27
10. Case and Mode Analysis	
Case Analysis	10-2
Performing Case Analysis	10-2
Constant Propagation Based on Cell Logic	10-5
Case Analysis of Integrated Clock-Gating Cells	10-7
Setting Case Analysis Values	10-8
Evaluating Conditional Arcs Using Case Analysis	10-9
Reporting Case Analysis Values	10-10
Removing Case Analysis Values	10-13
Constant Propagation Log File	10-13
Case Analysis and Maximum Capacitance Checking	10-16
Mode Analysis	10-16
Mode Analysis Overview	10-16
How Cell Modes Are Defined	10-19
Mode Groups	10-19
Setting Modes Using Case Analysis	10-19
Setting Modes Directly on Cells	10-20
Defining and Setting Design Modes	10-21

Defining Design Modes	10-22
Mapping Design Modes	10-23
Setting Design Modes	10-25
Reporting Modes	10-26
Mode Merging for Scenario Reduction	10-27
Running Mode Merging	10-28
Debugging Mode Merging Conflicts	10-29

11. Back-Annotation

Overview of SDF Back-Annotation	11-2
Reading SDF Files	11-2
Annotating Timing From a Subdesign Timing File	11-4
Annotating Load Delay	11-4
Annotating Conditional Delays From SDF	11-4
Annotating Timing Checks	11-6
Setting Annotations From the Command Line	11-7
Annotating Delays	11-7
Annotating Timing Checks	11-8
Annotating Transition Times	11-8
Generating Timing Constraints for Place and Route	11-8
Providing Constraint Coverage for the Entire Design	11-10
Reporting Delay Back-Annotation Status	11-11
Reporting Annotated or Nonannotated Delays	11-11
Reporting Annotated or Nonannotated Timing Checks	11-12
Faster Timing Updates in SDF Flows	11-12
Writing an SDF File	11-13
SDF Constructs	11-13
SDF Delay Triplets	11-14
SDF Conditions and Edge Identifiers	11-14
Reducing SDF for Clock Mesh/Spine Networks	11-15
PORT Construct	11-15
Normalizing Multidriven Arcs for Simulation	11-17
Writing VITAL Compliant SDF Files	11-19
Writing a Mapped SDF File	11-20
Specifying Timing Labels in the Library	11-20

Specifying the min_pulse_width Constraint	11-21
Using SDF Mapping	11-22
Supported SDF Mapping Functions	11-25
SDF Mapping Assumptions	11-27
Bus Naming Conventions	11-28
Labeling Bus Arcs	11-28
SDF Mapping Limitations	11-30
Mapped SDF File Examples	11-30
Managing SDF Large Files	11-37
Setting Lumped Parasitic Resistances and Capacitances	11-38
Setting Net Capacitance	11-38
Setting Net Resistance	11-39
Reduced and Detailed Parasitics	11-39
Reduced Parasitics	11-39
Detailed Parasitics	11-40
Supported File Formats for Parasitic Annotation	11-41
Characterization Trip Points	11-42
Reading Parasitic Files	11-44
Applying Location Transformations to Parasitic Data	11-45
Multivalued SPEF Files	11-45
Requirements for Multivalued SPEF Files	11-47
Reading Multivalued SPEF Files	11-47
Limitations of Multivalued SPEF	11-48
Reading a Single Corner From Multicorner Parasitic Data	11-48
Converting a SPEF Parasitics File to SBPF	11-48
Reading Multiple Parasitic Files	11-49
Checking the Annotated Nets	11-51
Scaling Parasitic Values	11-51
Net-Specific Parasitic Scaling	11-51
Ground-Capacitance and Resistance Scaling	11-52
Coupling-Capacitance Scaling	11-52
Resetting Scale Parasitics	11-52
Reporting Scale Parasitics	11-53
Net-Specific Parasitic Scaling Examples	11-54
Reading Parasitics for Incremental Timing Analysis	11-59
Incomplete Annotated Parasitics	11-60
Selecting a Wire Load Model for Incomplete Nets	11-60

Completing Missing Segments on the Net	11-61
Reporting Annotated Parasitics	11-62
Removing Annotated Delays, Checks, Transitions, and Parasitics	11-63
Back-Annotation Order of Precedence	11-63
12. Variation	
Advanced On-Chip Variation.....	12-2
AOCV Flow	12-2
Graph-Based AOCV Analysis.....	12-3
Path-Based AOCV Analysis	12-3
Specifying the Scope of the AOCV Analysis	12-4
Importing AOCV Information	12-4
Specifying AOCV Derating Tables	12-4
File Format for AOCV	12-5
AOCV Table Groups	12-9
Specifying Depth Coefficients.....	12-10
Guard-Banding in AOCV	12-10
Incremental Timing Derating.....	12-11
Using OCV Deratings in AOCV Analysis	12-12
Querying AOCV Deratings on Timing Paths	12-13
Parametric On-Chip Variation.....	12-14
Summary of Variables and Commands for Parametric On-Chip Variation	12-15
Preparing Input Data for Parametric On-Chip Variation.....	12-16
POCV Single Coefficient Specified in a Side File	12-17
POCV Slew-Load Table in Liberty Variation Format.....	12-18
Importing a SPEF File With Physical Locations.....	12-19
Enabling Parametric On-Chip Variation Analysis.....	12-20
Loading the Parametric On-Chip Variation Input Data.....	12-20
Specifying Guard Banding	12-21
Scaling the Parametric On-Chip Variation Coefficient	12-21
Enabling Constraint and Slew Variation.....	12-22
Reporting Parametric On-Chip Variation Results	12-22
Report the POCV Coefficient and Deratings	12-22
Report the POCV Analysis Results	12-24
Querying POCV Slack and Arrival Attributes.....	12-27

13. Multivoltage Design Flow

Multivoltage Analysis	13-3
UPF Commands	13-4
UPF Supply Sets	13-8
UPF Supply Set Handles	13-8
Identifying the Power and Ground Supply Nets With Attributes	13-10
Virtual Power Network	13-11
Setting Voltage and Temperature	13-12
Library Support for Multivoltage Analysis	13-14
Support for Fine-Grained Switch Cells in Power Analysis	13-15
Multivoltage Reporting and Checking	13-16
Collection (get_*) Commands	13-16
Reporting Power Supply Network Information	13-19
Querying Power and Ground Pin Attributes	13-23
Using the check_timing Command	13-23
Voltage Set on Each Supply Net Segment	13-24
Supply Net Connected to Each PG Pin of Every Cell	13-24
Compatible Driver-to-Load Signal Levels	13-24
Effect of Correlated Supplies on Signal Level Checking	13-28
Library PG Tcl File	13-29
Default Power and Ground Pin Names	13-29
Golden UPF Flow	13-30
Power Domain Mode of Version Z-2007.06	13-31
Multivoltage Method Before Version Z-2007.06	13-34
Setting Operating Conditions on Cells	13-35
Setting Rail Voltages Directly on Cells	13-35
Multiple Supply Voltage Analysis	13-35
Multivoltage Analysis Overview	13-36
Simultaneous Multivoltage Analysis	13-37
Simultaneous Multivoltage Analysis Usage Flow	13-40
Analysis Using Cross-Domain Derating	13-41
IR Drop Annotation	13-42
Signal Level Checking With IR Drop	13-42

14. Signal Integrity Analysis

Overview of Signal Integrity and Crosstalk	14-2
Crosstalk Delay Effects	14-3
Delta Delay and Fanout Stage Effect	14-3
Crosstalk Noise Effects	14-4
Aggressor and Victim Nets	14-5
Timing Windows and Crosstalk Delay Analysis	14-6
Cross-Coupling Models	14-6
Crosstalk Delay Analysis	14-7
Performing Crosstalk Delay Analysis	14-8
How PrimeTime SI Operates	14-9
PrimeTime SI Variables	14-11
Logical Correlation	14-12
Electrical Filtering	14-13
Usage Guidelines	14-15
Preparing to Run Crosstalk Analysis	14-15
Capacitive Coupling Data	14-15
Operating Conditions	14-15
Using check_timing	14-16
Including or Excluding Specific Nets From Crosstalk Analysis	14-16
Initial Crosstalk Analysis Run	14-22
Timing Window Overlap Analysis	14-23
Clock Groups	14-26
Crosstalk Analysis with Composite Aggressors	14-31
Path-Based Crosstalk Analysis	14-34
PrimeTime SI Crosstalk Delay Calculation Using CCS Models	14-34
Waveform Propagation	14-37
Annotated Delta Delays	14-38
Iteration Count and Exit	14-38
Timing Reports	14-39
Viewing the Crosstalk Analysis Report	14-39
Bottleneck Reports	14-41
Crosstalk Net Delay Calculation	14-43
Reporting Crosstalk Settings	14-43
Double-Switching Detection	14-45
Invoking Double-Switching Error Detection	14-47
How Double-Switching Is Detected	14-47
Reporting Double-Switching Violations	14-48
Fixing Double-Switching Violations	14-48

Static Noise Analysis	14-49
Static Noise Analysis Overview	14-49
Noise Bump Characteristics	14-51
Noise Bump Calculation	14-53
Noise-Related Logic Failures	14-54
PrimeTime SI Noise Analysis Flow	14-56
Noise Analysis Commands	14-58
Performing Noise Analysis	14-60
Reporting Noise Analysis Results	14-67
Setting Noise Bumps	14-73
Performing Noise Analysis with Incomplete Library Data	14-74
Noise Modeling With CCS Noise Data	14-77
Noise Immunity	14-78
CCS Noise Analysis for Unbuffered-Output Latches	14-78
Noise Modeling With Nonlinear Delay Models	14-80
Steady-State I-V Characteristics	14-81
Noise Immunity	14-86
Propagated Noise Characteristics	14-95

15. Advanced Analysis Techniques

Parallel Arc Path Tracing	15-2
Support for Retain Arcs	15-2
Asynchronous Logic Analysis	15-3
Combinational Feedback Loop Breaking	15-4
Unrelated Clocks	15-5
Three-State Bus Analysis	15-6
Limitations of the Checks	15-7
Disabling the Checks	15-7
Bus Contention	15-7
Floating Buses	15-8
Three-State Buffers	15-8
Performing Transient Bus Contention Checks	15-8
Performing Floating Bus Checks	15-10
Data-to-Data Checking	15-11
Data Check Examples	15-11
Data Checks and Clock Domains	15-14

Library-Based Data Checks	15-15
Interdependent Setup and Hold Pessimism Reduction	15-15
Use Model for SHPR	15-16
Setup-Preferred Slack Improvement	15-16
Hold-Preferred Slack Improvement	15-16
Total Negative Slack Improvements	15-17
SHPR Optimization Constraints	15-17
SHPR Optimization Mechanism	15-17
SHPR User Interface	15-18
SHPR Examples	15-19
Setup-Preferred Slack Improvement Example	15-19
Total Slack Improvement Example	15-20
Liberty Format Extension	15-21
Time Borrowing in Latch-Based Designs	15-22
Borrowing Time From Logic Stages	15-22
Latch Timing Reports	15-23
Maximum Borrow Time Adjustments	15-26
Time Borrowed and Time Given	15-31
Limiting Time Borrowing	15-33
PrimeTime ADV Advanced Latch Analysis	15-34
Enabling Advanced Latch Analysis	15-36
Breaking Loops	15-36
Specifying Loop-Breaker Latches	15-36
Finding Loop-Breaker Latches	15-36
Latch Loop Groups	15-37
Listing Collections of Latch Loop Groups	15-37
Reporting Latch Loop Groups	15-38
Specifying the Maximum Number of Latches Analyzed per Path	15-38
Timing Exceptions Applied to Latch Paths	15-39
False Path Exceptions	15-39
Multicycle Path Exceptions	15-40
Maximum and Minimum Delay Exceptions	15-40
Clock Groups	15-41
Specification of Exceptions on Throughpaths	15-42
Reporting Paths Through Latches	15-43
Filtering Paths That Arrive Early at Intermediate Latches	15-45
Reporting Through Loop-Breaker Latches	15-45
Tracing Forward Through Loop-Breaker Latches	15-45

Calculating the Worst and Total Negative Slack	15-46
Normalized Slack Analysis	15-47
Finding Recovered Paths.....	15-47
16. Checking the Design and Analysis Setup	
Basic Constraint Checking With the <code>check_timing</code> Command	16-2
Detailed Constraint Checking and Debugging With PrimeTime GCA.....	16-4
17. Reporting and Debugging Analysis Results	
Global Timing Summary Report	17-3
Path Timing Report.....	17-4
Using the <code>report_timing</code> Command	17-5
Reporting Normalized Slack	17-9
Running Normalized Slack Analysis.....	17-9
Setting Limits for Normalized Slack Analysis	17-10
Using Normalized Slack to Adjust the Clock Period.....	17-10
Using the <code>report_timing -exclude</code> Option	17-11
Cover Design Report.....	17-12
Quality of Results Report	17-13
Constraint Reports	17-13
Timing Constraints.....	17-13
Design Rule Constraints	17-14
Generating a Default Report	17-15
Reporting Violations	17-16
Maximum Skew Checks.....	17-18
No-Change Timing Checks	17-20
Minimum Pulse Width Report	17-21
Minimum Period Report.....	17-21
Bottleneck Report	17-23
Global Slack Report	17-26
Analysis Coverage Report	17-27
Design Reports.....	17-29
Disabled Timing Arcs Report	17-31

Clock Network Timing Report	17-31
Latency and Transition Time Reporting	17-32
Skew Reporting	17-32
Interclock Skew Reporting	17-34
Clock Timing Reporting Options	17-35
Summary Report	17-36
List Report	17-37
Verbose Path-Based Report	17-39
Limitations of Clock Network Reporting	17-41
Using the <code>get_clock_network_objects</code> Command	17-41
Clock-Gating and Recovery/Removal Checks	17-41
Timing Update Efficiency	17-42
Status Messages During a Timing Update	17-43

18. Graphical User Interface

Opening and Closing the GUI	18-3
Opening the GUI	18-3
Closing the GUI	18-4
GUI Windows	18-4
View Windows	18-7
Configuring View Windows	18-9
View Settings Panel	18-10
Toolbars and Panels	18-11
Displaying or Hiding Toolbars and Panels	18-12
Configuring Toolbars and Panels	18-12
Hierarchy Browser	18-13
Console	18-14
Object Selection	18-15
Selecting Timing Paths	18-16
Selecting Fanin or Fanout Logic	18-16
Object Chooser	18-17
Viewing Reports	18-17
Viewing and Customizing Histograms	18-18
Configuring Data Table Columns	18-20
Filtering Tables and Lists	18-21
Setting GUI Preferences	18-22

Timing Analysis Flow With the GUI	18-23
Viewing Timing Analysis Histograms	18-24
Examining Endpoint Slack	18-25
Examining Path Slack	18-27
Examining Net Capacitance	18-30
Examining Design Rule Slack	18-31
Examining Timing Bottlenecks	18-32
Analyzing Collections of Timing Paths	18-34
Loading Path Collections	18-37
Categorizing the Timing Paths	18-39
Removing Categories	18-40
Saving Path Categories in a File	18-41
Loading Path Categories From a File	18-41
Marking Blocks and Applying Block Category Rules	18-42
Creating Custom Category Rules	18-43
Selecting Category Attributes for a Custom Category Rule	18-45
Defining a Filter Expression for a Custom Category Rule	18-46
Examining Clock Paths in an Abstract Clock Graph	18-48
Opening Abstract Clock Graph Views	18-51
Displaying and Hiding Clock Path Elements	18-51
Reversing and Reapplying Changes	18-54
Collapsing Selected Side Branches	18-54
Collapsing Duplicate Clock Gates	18-56
Viewing Clock Latency Over Time	18-56
Displaying the Clock Trees Hierarchically	18-58
Measuring Distances in an Abstract Clock Graph View	18-58
Analyzing Clock Domains and Clock-to-Clock Relationships	18-60
Querying Launch-Capture Clock Pairs	18-63
Selecting Clocks or Clock Domains	18-63
Sorting the Clock Tree View	18-64
Finding Clocks by Name	18-65
Filtering Clock Domains	18-66
Locating Launch and Capture Clocks in the Clock Matrix	18-67
Saving Clock Attribute or Clock Matrix Constraint Data	18-68
Clock Matrix Symbols and Colors	18-69

Examining Timing Paths and Design Logic in a Schematic	18-70
Examining Hierarchical Cells	18-72
Grouping Cells by Hierarchy	18-74
Moving Down or Up the Design Hierarchy	18-75
Displaying or Hiding Buffers and Inverters	18-75
Displaying or Hiding Unconnected Macro Pins	18-77
Expanding or Collapsing Buses	18-78
Examining Clock Paths in a Schematic	18-80
Highlighting Clock Propagation Paths	18-81
Displaying or Hiding Buffers and Inverters	18-82
Querying and Selecting Metaobjects	18-84
Collapsing Clock Gating Cells	18-85
Viewing and Modifying Schematics	18-86
Schematic Window	18-88
Selecting or Highlighting Objects By Name	18-90
Highlighting Schematics By Using Filtering Rules	18-91
Annotating Schematic Pins and Ports	18-93
Adding or Removing Selected Logic	18-94
Adding Timing Paths	18-94
Adding Fanin and Fanout Logic	18-95
Reversing and Reapplying Schematic Changes	18-97
Printing Images of Schematics, Abstract Clock Graphs, and Timing Waveforms	18-98
Examining Timing Path Details in a Data Table	18-99
Loading Timing Paths in a New Path Data Table	18-100
Creating Path Data Histograms	18-101
Viewing Timing Reports	18-102
Saving the Timing Path Data	18-103
Reloading Timing Paths in a Path Data Table	18-103
Inspecting Timing Path Elements	18-104
Loading Paths Into a Path Inspector Window	18-107
Finding Text in the Path Inspector	18-108
Viewing Timing Waveforms	18-108
Configuring the Path Inspector	18-109
Saving the Path Element Data	18-112

Viewing Object Attributes	18-112
Viewing the Current Selection	18-113
Querying Objects	18-114
Copying Text on the Query Panel	18-115
Configuring the Query Panel	18-116
Viewing Object Properties	18-117
Managing Attribute Groups	18-119
Saving and Restoring Attribute Groups	18-122
Creating Custom Attribute Groups	18-122
Modifying Attribute Groups	18-123
Copying and Renaming Attribute Groups	18-124
Examining Object Attributes in a Data Table	18-124
Loading Objects in an Object Attribute Table	18-126
Creating Object Attribute Histograms	18-127
Viewing Object Reports	18-128
Saving the Object Attribute Data	18-129
Recalculating Timing Paths	18-129
Comparing Normal and Recalculated Paths	18-129
Comparing Normal and Recalculated Path Pins	18-130
Analyzing Signal Integrity	18-131
Examining Delta Delay	18-133
Examining Path Delta Delay	18-134
Examining Bump Voltage	18-136
Examining Accumulated Bump Voltage	18-139
Examining Crosstalk Coupling	18-140
Examining Noise Slack	18-143
Examining Noise Bump	18-144
Examining Accumulated Noise Bump	18-146
Examining Noise Immunity Curves	18-147
Viewing Propagated Waveforms for Recalculated Paths	18-149
Analyzing Timing Paths from Multiple Scenarios	18-151
Interactive Multi-Scenario Analysis Flow	18-151
Viewing the Timing Paths in the Path Analyzer Window	18-154
Specifying a User-Defined Value for the Slack	18-156
Menu Command Reference	18-158
File Menu Commands	18-159

View Menu Commands	18-160
Select Menu Commands	18-162
Design Menu Commands	18-163
Timing Menu Commands.	18-164
Highlight Menu Commands	18-165
Crosstalk Menu Commands	18-166
Noise Menu Commands	18-167
Power Menu Commands	18-167
Clock Menu Commands.	18-168
Schematic Menu Commands.	18-169
ECO Menu Commands	18-171
Window Menu Commands.	18-172
Help Menu Commands	18-173

19. ECO Flow

Setting Options for ECO	19-3
Configuring the ECO for Multiple Libraries	19-3
Excluding the Usage of Specific Library Cells From ECO	19-3
The dont_use Application Attribute	19-4
The pt_dont_use User-Defined Attribute	19-4
Handling Multiply Instantiated Modules	19-4
Fixing Design Rule and Timing Violations With ECO Guidance.	19-5
Recommended Automated Fixing Flow.	19-6
Fixing Design Rule Violations	19-6
Fixing Setup and Hold Timing Violations.	19-7
Examining the ECO Results and the Reasons for Unfixed Violations	19-8
Resizing Sequential Cells	19-9
Prioritizing the Fixing of Timing Violations Over Design Rule Violations	19-10
Multi-Scenario ECO Fixing With Constrained Resources	19-11
Minimum Host Requirement.	19-11
Disk Space Requirement	19-11
Runtime Versus Number of Hosts	19-11
Optimizing DMSA Scripts for Faster Turnaround Time.	19-11
Performing Power Recovery	19-13
Cell Swapping for Leakage Power Reduction	19-15
Cell Swapping Based on the Library Cell Names.	19-15

Cell Swapping Based on a User-Defined Attribute.....	19-16
Performing Power Recovery With DMSA.....	19-17
Physically-Aware ECO	19-18
Physically-Aware ECO Setup Fixing Through Buffer Insertion	19-21
Physically-Aware ECO for Multivoltage Designs	19-22
Manual Netlist Editing	19-23
Finding Alternative Library Cells	19-24
“What-If” Incremental Analysis	19-25
Automatic Uniquifying of Blocks.....	19-26
Resolving Library Cells	19-26
Estimating Delay Changes.....	19-27
Customizing Estimation Columns.....	19-30
Sizing Cells	19-32
Inserting and Removing Buffers	19-33
Swapping Cells	19-33
Renaming Cells or Nets.....	19-34
User Function Class Support.....	19-35
Netlist Editing in Multiple Scenarios.....	19-35
Writing Change Lists.....	19-36
Implementing the Changes With the Galaxy Design Platform	19-36
ECO Flow With StarRC Incremental Parasitic Extraction	19-37
20. Timing Models for Hierarchical Analysis	
Overview of Timing Models.....	20-2
Synopsys Logic Libraries	20-4
Quick Timing Models	20-4
Quick Timing Model Design Flow	20-5
Quick Timing Model Parameters	20-5
Commands to Set Global Parameters	20-6
Commands to Specify the Model Information.....	20-7
Computing Arc Delays	20-8
Creating a Quick Timing Model	20-10
Create the Model and Set Global Parameters	20-11
Specify Model Information	20-12
View the New Model.....	20-12

Save the Model.....	20-13
Using a Quick Timing Model in a Design.....	20-13
Quick Timing Model Command Summary.....	20-14
Interface Logic Models	20-14
Benefits, Flow, and Limitations of ILMs	20-16
Generating, Validating, and Using an ILM.....	20-16
Generating an ILM	20-17
Example of Generating an ILM.....	20-18
Specific Setups with ILM	20-19
Recommended Options for the <code>create_ilm</code> Command	20-19
Validating an ILM	20-20
Model Validating Issues and Resolutions.....	20-24
Using the ILM in a Top-Level Netlist	20-25
Shielded PrimeTime SI ILM Flow	20-26
Generating PrimeTime SI ILM	20-26
Model Validating for PrimeTime SI ILM	20-28
Using a PrimeTime SI ILM in a Top-Level Netlist	20-29
Non-Shielded PrimeTime SI ILM Flow.....	20-29
Hierarchical Crosstalk Analysis	20-30
Generating the Block Context.....	20-33
Context Files.....	20-34
Top-Level Design Files	20-36
Blocks Below the Top Level	20-37
Block-Level Analysis and Timing Model Generation.....	20-38
Top-Level Analysis	20-41
Analysis Iterations Using Annotated Arrival and Slew	20-42
Using an ILM.....	20-43
Extracted Timing Models	20-44
Model Extraction Overview	20-45
Timing Model Extraction Requirements.....	20-46
Timing Model Extraction Process	20-46
Extracting Timing Paths.....	20-48
Boundary Nets	20-49
Internal Nets	20-49
Paths From Inputs to Registers	20-50
Paths From Inputs to Outputs.....	20-50
Paths From Registers to Outputs	20-50
Paths From Registers to Registers.....	20-51
Clock Paths	20-51

Interface Latch Structures	20-51
Minimum Pulse Width and Minimum Period	20-52
False Paths	20-53
Clock-Gating Checks	20-53
Back-Annotated Delays	20-53
Block Scope	20-54
Noise Characteristics	20-54
Other Extracted Information	20-55
Limitations of Model Extraction	20-56
Limitations of Extracted Models in Design Compiler	20-58
Limitations of Extracted Models in .lib Format	20-59
Extracted Model Types	20-59
Extraction Variables	20-59
Variable Setting Guidelines	20-61
Delay Table Generation	20-61
Preparing for Extraction	20-63
Setting the Extraction Variables	20-63
Setting the Operating Conditions	20-64
Defining the Wire Load Models	20-64
Defining Clocks	20-65
Setting Up Latch Extraction	20-65
Identifying False Paths	20-66
Removing Internal Input and Output Delays	20-66
Controlling Mode Information in the Extracted Model	20-66
Performing Timing Checks	20-67
Preparing for Crosstalk Analysis	20-67
Model Extraction Options	20-68
Extracted Model Examples	20-68
Extracting Clock-Gating Checks	20-72
Extraction of Combinational Paths Through Gating Logic	20-72
Extraction of Clock-Gating Checks As No-Change Arcs	20-73
Extracting Constant Values	20-74
Extracting Timing Exceptions	20-74
Restricting the Types of Arcs Extracted	20-78
Back-Annotated Delay and Layout Information	20-79
Back-Annotated Delay on Nets and Cells	20-79
Guidelines for Back-Annotation	20-81
Performing Model Extraction	20-82
Loading and Linking the Design	20-82
Preparing to Extract the Model	20-82

Generating the Model	20-83
PrimeTime User-Defined Attributes in .lib	20-83
Merging Extracted Models	20-84
Extracting Internal Pins	20-87
Clocks on Internal Pins of a Design	20-87
Generated Clocks	20-87
Check Pins for Clock Networks	20-88
Check Pins for Bidirectional Ports	20-89
Model Validation	20-90
Model Validation Overview	20-90
Automatic Model Validation	20-90
Manual Model Validation Using Commands	20-93
Slew Propagation	20-94
Viewing the write_interface_timing Report	20-95
Arc Types	20-97
Time Borrowing by Transparent Latches	20-97
Viewing the compare_interface_timing Report	20-98
Comparison Tolerance	20-100
Absolute Tolerance	20-101
Percentage Tolerance	20-101
Debugging Timing Arc Comparison Failures	20-101
Validation Flow With Timing Arc Debugging	20-102
Debugging Report Example	20-103
Causes of Validation Failure	20-104
Debugging Other Comparison Failures	20-105
Transition Time	20-105
Capacitance	20-105
Design Rules	20-105
Hierarchical Scope Checking	20-106
Scope Checking Overview	20-106
Types of Block Scope Checking	20-109
Clock Arrival, Transition, and Waveform	20-110
Interclock Skew and Uncertainty	20-110
Data Input and Output Conditions for Crosstalk	20-111
Block Scope Files	20-112
Generating a Block Scope File	20-112
Reporting Block Scope Files	20-112
Updating Block Scope Data	20-114
Checking the Block Scope at the Chip Level	20-114

Context Characterization	20-117
Context Characterization Overview	20-117
Setting Synthesis or Optimization Constraints	20-117
Performing Subdesign Timing Analysis	20-118
Deriving the Context of a Subdesign	20-118
Clock Information	20-119
Input and Output Delay Times	20-119
Point-to-Point Timing Exceptions	20-119
Constant Logic Values on Inputs	20-120
Input Drive Strength and Port Capacitance	20-120
Wire Load Models	20-121
Design Rule Checks	20-121
Annotated Delays and Parasitics	20-121
Input Delay and Port Capacitance	20-123
Writing Physical Information	20-127
Reporting the Timing Context	20-127
Generating Scripts for Characterized Contexts	20-127
Removing Context Information	20-128
Limitations of Context Characterization	20-128
21. Using PrimeTime With SPICE	
Generating a SPICE Deck With the write_spice_deck Command	21-2
Writing a SPICE Deck for a Timing Path	21-2
Writing a SPICE Deck for a Timing Arc	21-3
Library Sensitization in write_spice_deck	21-6
Library Driver Waveform	21-8
Additional Required Information for SPICE Simulation	21-8
Example of write_spice_deck Output	21-9
Limitations of Using write_spice_deck for SPICE Correlation	21-11
Correlating PrimeTime and SPICE Results With the Simulation Link	21-11
Summary of Simulation Link Commands	21-12
Path-Based Uncoupled SPICE Analysis	21-13
Arc-Based Coupled SPICE Analysis	21-15
22. Using PrimeTime With Design Compiler	
Features Specific to PrimeTime	22-2
Timing Analysis Differences	22-2

Paths	22-2
Time Borrowing for Hold Checks	22-2
Segmenting Paths: Load-Dependent Delays	22-3
Transition Time	22-3
current_design Command	22-3
Command Scripts	22-3
Sharing Design Compiler and PrimeTime Scripts	22-4
Synopsys Design Constraints Formatted Script Files	22-5
Checking the Syntax of the SDC File	22-5
Reading .db Files With Back-Annotated Data.	22-5
Path Groups in Design Compiler	22-6
23. Attributes	
Using Attributes	23-2
Creating User-Defined Attributes.	23-2
Importing User-Defined Attributes	23-3
Saving Design Attributes.	23-5
Using Paths to Generate Custom Reports	23-6
Using Arcs to Generate Custom Reports	23-8
Creating a Collection of Library Arcs	23-9
Reporting Library Data and Driver Information	23-9

Glossary

Preface

This preface includes the following sections:

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

About This Guide

The Synopsys PrimeTime® Suite provides comprehensive full-chip analysis of static timing, signal integrity, statistical timing, and full-chip power in a single integrated environment.

The *PrimeTime User Guide* describes the analysis flow using the PrimeTime, PrimeTime SI, and PrimeTime ADV tools. For information about other PrimeTime add-on tools, see the following companion guides:

- *PrimeTime PX User Guide* – Static and dynamic full-chip power analysis.
 - *PrimeTime GCA User Guide* – Constraint Checking and Debugging.
-

Audience

This guide is for engineers who use PrimeTime for static timing and signal integrity analysis. Readers should have some familiarity with crosstalk and signal integrity principles.

Related Publications

For additional information about the PrimeTime Suite, 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:

- Design Compiler®
 - IC Compiler™ and IC Compiler™ II
 - Library Compiler™
 - NanoTime
 - SiliconSmart®
 - StarRC™
-

Release Notes

For information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs), see the *PrimeTime Suite Release Notes* on the SolvNet site.

To see the *PrimeTime Suite Release Notes*,

1. Go to the SolvNet Download Center located at the following address:
<https://solvnet.synopsys.com/DownloadCenter>
2. Select PrimeTime Suite, 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>report_timing</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>report_timing path_collection</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>pt_shell> report_timing</code>
[]	Denotes optional arguments in syntax, such as <code>update_timing [-full]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</code>
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 PrimeTime

The PrimeTime Suite performs full-chip, gate-level static timing analysis, which is an essential part of the design and analysis flow for chip designs. The tool exhaustively validates the timing performance of a design by checking all paths for timing violations, without using logic simulation or test vectors.

For an introduction to the PrimeTime tool, see

- [PrimeTime Features](#)
- [Using PrimeTime in the Implementation Flow](#)
- [Overview of Static Timing Analysis](#)

PrimeTime Features

To learn about the capabilities of the PrimeTime Suite, see

- [Types of Design Checking](#)
 - [Analysis Features](#)
 - [Supported Timing Models](#)
 - [PrimeTime Add-On Tools](#)
-

Types of Design Checking

The PrimeTime tool performs the following types of design checking:

- Setup, hold, recovery, and removal constraints
 - User-specified data-to-data timing constraints
 - Clock-gating setup and hold constraints
 - Minimum period and minimum pulse width for clocks
 - Design rules (minimum and maximum transition time, capacitance, and fanout)
-

Analysis Features

The PrimeTime tool supports a wide range of advanced timing analysis features, including the following:

- Multiple clocks and clock frequencies
- Multicycle path timing exceptions
- False path timing exceptions
- Transparent latch analysis and time borrowing
- Simultaneous minimum and maximum delay analysis for setup and hold constraints
- Analysis with on-chip variation (OCV) of process, voltage, and temperature (PVT) conditions
- Case analysis of constants or specific transitions applied to specified inputs
- Mode analysis with module-specific operating modes, such as read mode or write mode for a RAM module

- Bottleneck analysis to report cells that cause the most timing violations
- Crosstalk analysis between physically adjacent nets
- ECO analysis that generates a change list by using inserted buffers, resized cells, and swapped cells

Supported Timing Models

PrimeTime supports the use of timing models to represent chip submodules. A timing model contains information about the timing characteristics, but not the logical functionality, of a submodule.

PrimeTime can generate a timing model from a submodule netlist, and then use that model in place of the original netlist for timing analysis at higher levels of hierarchy. This technique makes whole-chip analysis run much faster.

Another use of timing models is to protect intellectual property. If you supply a chip submodule to a customer for integration into the customer's larger chip, you can provide the timing model without the original netlist. This method allows the customer to perform accurate timing analysis with the submodule, without having access to the netlist.

PrimeTime supports the following types of timing models:

- Quick timing model – An approximate timing model created in PrimeTime using a sequence of PrimeTime commands. This type of model is useful early in the design cycle, when a netlist is not yet available for a submodule.
- Extracted timing model – A timing-only model extracted by PrimeTime from a gate-level netlist. This type of model discards all of the logic of the original netlist and replaces it with a set of timing arcs between clocks, inputs, and outputs.
- Interface logic model – A structural timing model extracted by PrimeTime from a gate-level netlist. This type of model preserves the interface logic of the original netlist and discards the internal register-to-register logic that has already been verified at the module level.
- Liberty model – A timing model defined in a descriptive language, either written manually or translated from a timing description in another form.

See Also

- [Timing Models for Hierarchical Analysis](#)

PrimeTime Add-On Tools

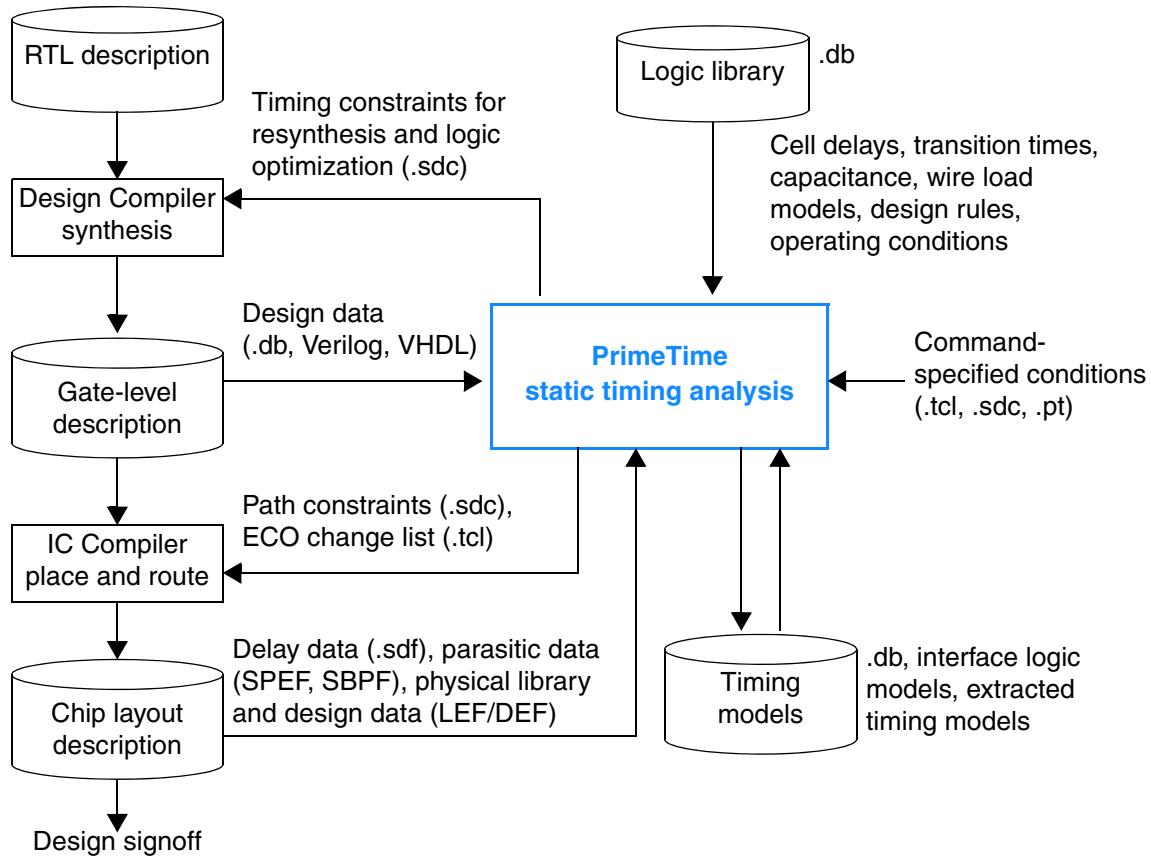
The PrimeTime Suite includes the following add-on tools; to use these tools, you must have the appropriate licenses:

- PrimeTime SI – Performs signal integrity analysis. It calculates the timing effects of cross-coupled capacitance between nets and includes the resulting delay changes in the PrimeTime analysis reports. It also calculates the logic effects of crosstalk noise and reports conditions that could lead to functional failure. For more information, see [Signal Integrity Analysis](#).
- PrimeTime ADV – Provides advanced features such as parametric on-chip variation, advanced latch analysis, multi-input switching analysis. It also performs advanced ECO fixing capabilities such as physical ECO and power recovery.
- PrimeTime GCA – Performs constraint analysis and debugging of full-chip or block-level netlists. You can use PrimeTime GCA as a standalone tool or invoke a subset of PrimeTime GCA features within PrimeTime. For more information, see the *PrimeTime GCA User Guide*.
- PrimeTime PX – Accurately analyzes full-chip power dissipation of cell-based designs. It provides vector-free and vector-based peak power and average power analysis. The vectors to PrimeTime PX are either RTL or gate-level simulation results in Value Change Dump (VCD) format or Switching Activity Interchange Format (SAIF). PrimeTime PX provides multivoltage and power domain analysis. It also has an integrated graphical user interface (GUI) for visual power debugging. For more information, see the *PrimeTime PX User Guide*.

Using PrimeTime in the Implementation Flow

As part of the Synopsys Galaxy™ Design Platform, PrimeTime works well with the Design Compiler synthesis tool and the IC Compiler place-and-route tool. These tools use many of the same libraries, databases, and commands, as shown in [Figure 1-1](#). PrimeTime can also operate as a standalone static timing analyzer in other design flows.

Figure 1-1 Implementation Flow Using PrimeTime



Starting from an RTL design description, the Design Compiler tool generates a gate-level design description. The PrimeTime tool reads this description and verifies the design timing using information provided in the logic library.

If the tool finds any timing violations, the design needs to be resynthesized using new timing constraints (generated by PrimeTime) to fix the conditions that are causing the timing violations.

When the gate-level design is free of timing violations, you can proceed to placement and routing. This produces a chip layout database from which accurate delay information or detailed parasitic information can be extracted. This data, when back-annotated on the design in PrimeTime, results in a physically-accurate timing analysis.

If there are timing violations, PrimeTime can generate an engineering change order (ECO) change list to fix the violations. After the timing violations have been resolved, PrimeTime can also generate an ECO change list that optimizes area and power. IC Compiler uses the ECO change list to implement the changes.

A successful validation of the circuit timing at this point leads to signoff of the completed design.

Compatibility With Design Compiler and IC Compiler

The PrimeTime static timing analysis tool is designed to work well with the Synopsys Design Compiler synthesis tool and the IC Compiler place-and-route tool. These tools are compatible in the following ways:

- They use the same logic libraries and read the same design data files in .db and .ddc formats.
- They support the Synopsys Design Constraints (SDC) format for specifying design intent, including the timing and area constraints for a design.
- They share many of the same commands, such as the `create_clock`, `set_input_delay`, and `report_timing` commands. Shared commands are identical or very similar in operation.
- They share the same delay calculation algorithms and generally produce identical delay results.
- They generate very similar timing reports.
- PrimeTime can capture the timing environment of a synthesizable subcircuit and write this timing environment as a series of Design Compiler commands. You can use the resulting script in Design Compiler to define the timing constraints for synthesis or logic optimization of the subcircuit.
- PrimeTime can provide engineering change order (ECO) guidance to IC Compiler by generating change lists that fix timing violations and optimize area and power.

Although Design Compiler and IC Compiler have their own built-in static timing analysis capability, PrimeTime has better speed, capacity, and flexibility for static timing analysis, and offers many features not supported by the other tools.

See Also

- [Using PrimeTime With Design Compiler](#)

Overview of Static Timing Analysis

Static timing analysis is a method of validating the timing performance of a design by checking all possible paths for timing violations. To check a design for violations, PrimeTime breaks the design down into a set of timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface.

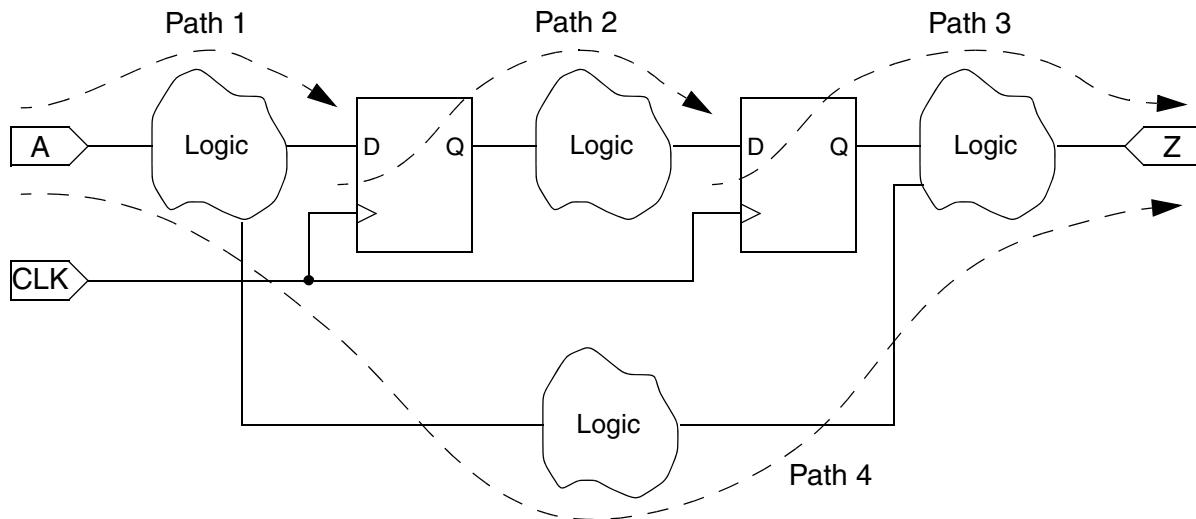
Another way to perform timing analysis is to use dynamic simulation, which determines the full behavior of the circuit for a given set of input stimulus vectors. Compared with dynamic simulation, static timing analysis is much faster because it is not necessary to simulate the logical operation of the circuit. It is also more thorough because it checks all timing paths, not just the logical conditions that are sensitized by a particular set of test vectors. However, static timing analysis can only check the timing, not the functionality, of a circuit design.

Timing Paths

The first step performed by PrimeTime for timing analysis is to break the design down into a set of timing paths. Each path has a startpoint and an endpoint. The startpoint is a place in the design where data is launched by a clock edge. The data is propagated through combinational logic in the path and then captured at the endpoint by another clock edge.

The startpoint of a path is a clock pin of a sequential element, or possibly an input port of the design (because the input data can be launched from some external source). The endpoint of a path is a data input pin of a sequential element, or possibly an output port of the design (because the output data can be captured by some external sink).

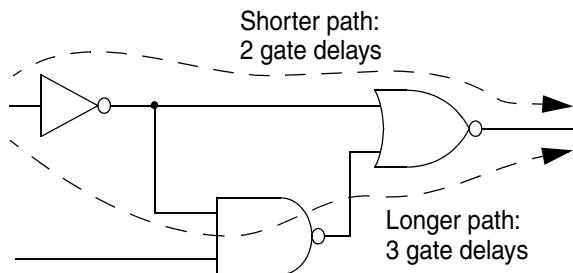
[Figure 1-2](#) shows an example of a simple design and the data paths contained in that design.

Figure 1-2 Timing Paths

In this figure, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point:

- Path 1 starts at an input port and ends at the data input of a sequential element.
- Path 2 starts at the clock pin of a sequential element and ends at the data input of a sequential element.
- Path 3 starts at the clock pin of a sequential element and ends at an output port.
- Path 4 starts at an input port and ends at an output port.

A combinational logic cloud might contain multiple paths, as shown in [Figure 1-3](#). PrimeTime uses the longest path to calculate a maximum delay or the shortest path to calculate a minimum delay.

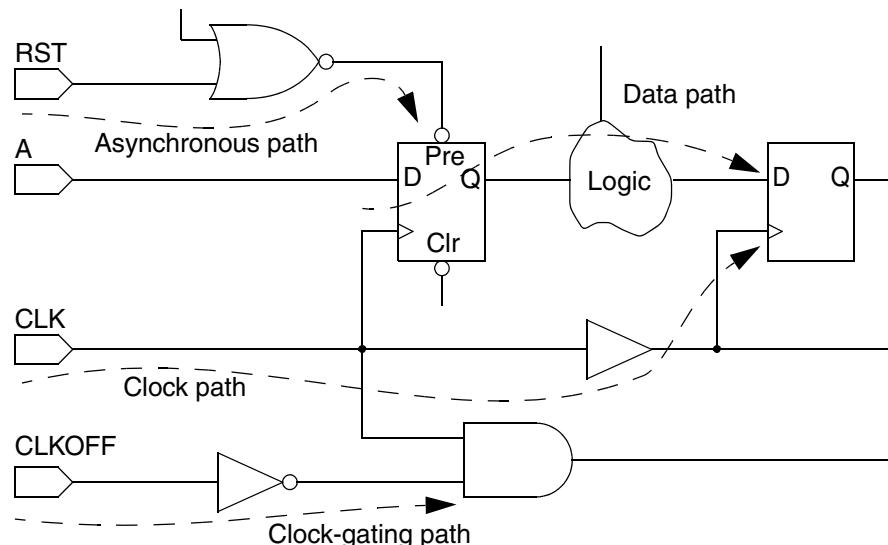
Figure 1-3 Multiple Paths Through Combinational Logic

In addition to the data paths just described, PrimeTime considers other types of paths for timing analysis, such as the following:

- Clock path (a path from a clock input port or cell pin, through one or more buffers or inverters, to the clock pin of a sequential element) for data setup and hold checks
- Clock-gating path (a path from an input port to a clock-gating element) for clock-gating setup and hold checks
- Asynchronous path (a path from an input port to an asynchronous set or clear pin of a sequential element) for recovery and removal checks

[Figure 1-4](#) shows some examples of these types of paths.

Figure 1-4 Path Types



Delay Calculation

After breaking down a design into a set of timing paths, PrimeTime calculates the delay along each path. The total delay of a path is the sum of all cell and net delays in the path.

The method of delay calculation depends on whether chip layout has been completed. Before layout, the chip topography is unknown, so PrimeTime must estimate the net delays using wire load models.

After layout, an external tool can accurately determine the delays and write them to a Standard Delay Format (SDF) file. PrimeTime can read the SDF file and back-annotate the design with the delay information for layout-accurate timing analysis. PrimeTime can also accept a detailed description of parasitic capacitors and resistors in the interconnection network, and then accurately calculate net delays based on that information.

Cell Delay

Cell delay is the amount of delay from input to output of a logic gate in a path. In the absence of back-annotated delay information from an SDF file, PrimeTime calculates the cell delay from delay tables provided in the logic library for the cell.

Typically, a delay table lists the amount of delay as a function of one or more variables, such as input transition time and output load capacitance. Based on these table entries, PrimeTime calculates each cell delay. When necessary, PrimeTime uses interpolation or extrapolation of table values to obtain a delay value for the current conditions specified for the design.

Net Delay

Net delay is the amount of delay from the output of a cell to the input of the next cell in a timing path. This delay is caused by the parasitic capacitance of the interconnection between the two cells, combined with net resistance and the limited drive strength of the cell driving the net.

PrimeTime can calculate net delays by the following methods:

- By using specific time values back-annotated from an SDF file
- By using detailed parasitic resistance and capacitance data back-annotated from file in RSPF, SPEF, or SBPF format
- By estimating delays from a wire load model

A wire load model attempts to predict the capacitance and resistance of nets in the absence of back-annotated delay information or parasitic data. The logic library provides statistical wire load models for estimating parasitic resistance and capacitance based on the number of fanout pins on each net.

Constraint Checking

After PrimeTime determines the timing paths and calculates the path delays, it can check for violations of timing constraints, such as setup and hold constraints.

A setup constraint specifies how much time is necessary for data to be available at the input of a sequential device before the clock edge that captures the data in the device. This constraint enforces a maximum delay on the data path relative to the clock path.

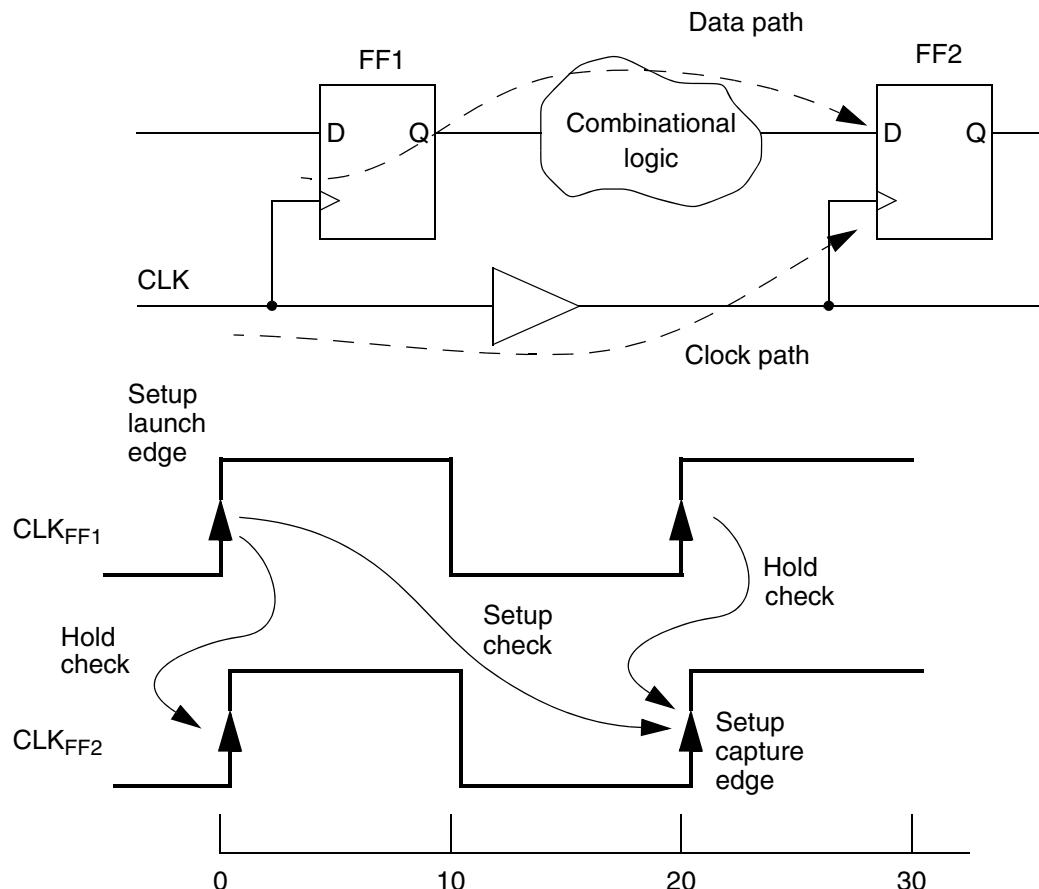
A hold constraint specifies how much time is necessary for data to be stable at the input of a sequential device after the clock edge that captures the data in the device. This constraint enforces a minimum delay on the data path relative to the clock path.

In addition to setup and hold constraints, PrimeTime can also check recovery/removal constraints, data-to-data constraints, clock-gating setup/hold constraints, and minimum pulse width for clock signals.

The amount of time by which a violation is avoided is called the slack. For example, for a setup constraint, if a signal must reach a cell input at no later than 8 ns and is determined to arrive at 5 ns, the slack is 3 ns. A slack of 0 means that the constraint is just barely satisfied. A negative slack indicates a timing violation.

Setup and Hold Checking for Flip-Flops

[Figure 1-5](#) shows how PrimeTime checks setup and hold constraints for a flip-flop in the absence of timing exceptions that apply to the data path.

Figure 1-5 Setup and Hold Checks

For this example, assume that the flip-flops are defined in the logic library to have a minimum setup time of 1.0 time units and a minimum hold time of 0.0 time units. The clock period is defined in PrimeTime to be 10 time units. (The time unit size, such as ns or ps, is specified in the logic library.)

By default, PrimeTime assumes that signals are to be propagated through each data path in one clock cycle. Therefore, when PrimeTime performs a setup check, it verifies that the data path delay is small enough so that the data launched from FF1 reaches FF2 within one clock cycle, and arrives at least 1.0 time unit before the data gets captured by the next clock edge at FF2. If the data path delay is too long, it is reported as a timing violation. For this setup check, PrimeTime considers the longest possible delay along the data path and the shortest possible delay along the clock path between FF1 and FF2.

When PrimeTime performs a hold check, it verifies that the data launched from FF1 reaches FF2 no sooner than the capture clock edge for the previous clock cycle. This check ensures that the data already existing at the input of FF2 remains stable long enough after the clock

edge that captures data for the previous cycle. For this hold check, PrimeTime considers the shortest possible delay along the data path and the longest possible delay along the clock path between FF1 and FF2. A hold violation can occur if the clock path has a long delay.

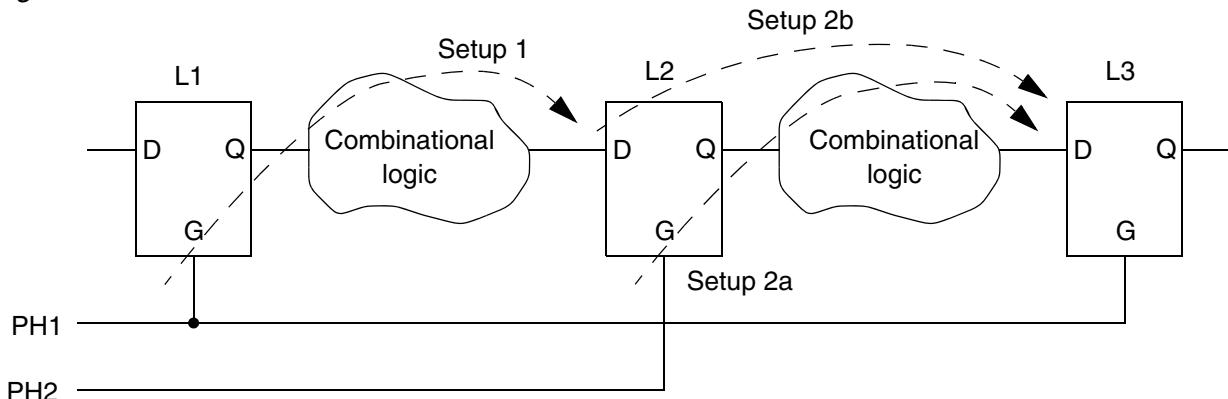
Note:

For more examples of setup and hold calculations, including paths that use different clocks at the path startpoint and endpoint, see [Timing Exceptions](#).

Setup and Hold Checking for Latches

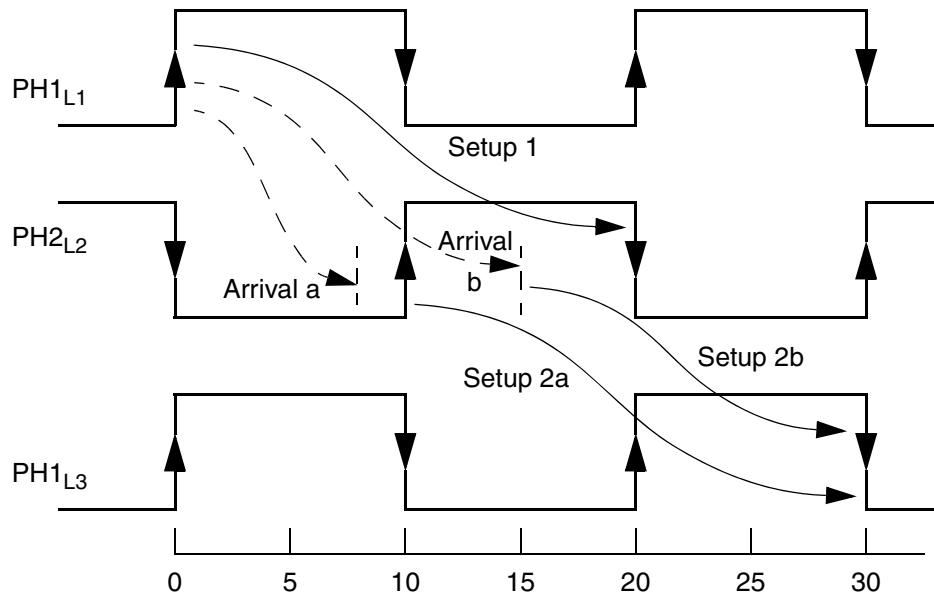
Latch-based designs typically use two-phase, nonoverlapping clocks to control successive registers in a data path. In these cases, PrimeTime can use time borrowing to lessen the constraints on successive paths. For example, consider the two-phase, latch-based path shown in [Figure 1-6](#). All three latches are level-sensitive, with the gate active when the G input is high. L1 and L3 are controlled by PH1, and L2 is controlled by PH2. A rising edge launches data from the latch output, and a falling edge captures data at the latch input. For this example, consider the latch setup and delay times to be zero.

Figure 1-6 Latch-Based Paths



[Figure 1-7](#) shows how PrimeTime performs setup checks between these latches. For the path from L1 to L2, the rising edge of PH1 launches the data. The data must arrive at L2 before the closing edge of PH2 at time = 20. This timing requirement is labeled Setup 1. Depending on the amount of delay between L1 and L2, the data might arrive either before or after the opening edge of PH2 at time = 10, as indicated by the dashed-line arrows labeled “Arrival a” and “Arrival b” in the timing diagram. Arrival after time = 20 is a timing violation.

Figure 1-7 Time Borrowing in Latch-Based Paths



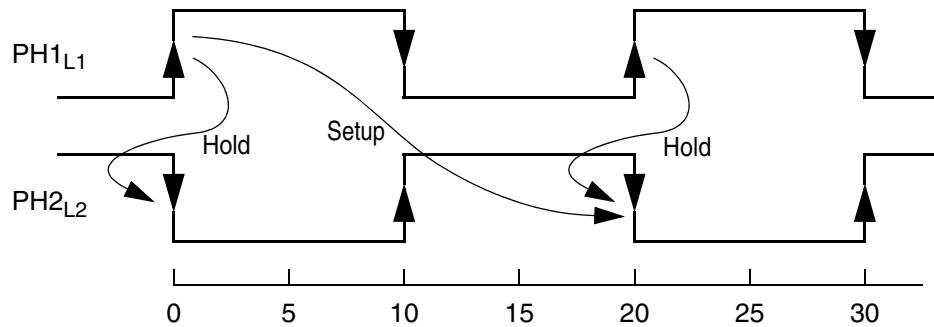
If the data arrives at L2 before the opening edge of PH2 at time = 10 (Arrival a), the data for the next path from L2 to L3 gets launched by the opening edge of PH2 at time = 10, just as a synchronous flip-flop operates. In this case, no time is borrowed from the second path. This timing requirement for L2 to L3 is labeled Setup 2a.

If the data arrives after the opening edge of PH2 (Arrival b), the first path (from L1 to L2) borrows time from the second path (from L2 to L3). In that case, the launch of data for the second path occurs not at the opening edge, but at the data arrival time at L2, at some time between the opening and closing edges of PH2. This timing requirement is labeled Setup 2b. When borrowing occurs, the path originates at the D pin rather than the G pin of L2.

For the first path (from L1 to L2), PrimeTime reports the setup slack as zero if borrowing occurs. The slack is positive if the data arrives before the opening edge at time=10, or negative (a violation) if the data arrives after the closing edge at time=20.

To perform hold checking, PrimeTime considers the launch and capture edges relative to the setup check. It verifies that data launched at the startpoint does not reach the endpoint too quickly, thereby ensuring that data launched in the previous cycle is latched and not overwritten by the new data. This is depicted in [Figure 1-8](#).

Figure 1-8 Hold Checks in Latch-Based Paths



Timing Exceptions

When certain paths are not intended to operate according to the default setup/hold behavior assumed by PrimeTime, you should specify those paths as timing exceptions. Otherwise, PrimeTime might incorrectly report those paths as having timing violations.

PrimeTime lets you specify the following types of timing exceptions:

- False path – A path that is never sensitized due to the logic configuration, expected data sequence, or operating mode.
- Multicycle path – A path designed to take more than one clock cycle from launch to capture.
- Minimum or maximum delay path – A path that must meet a delay constraint that you specify explicitly as a time value.

2

Getting Started

To get started with using the PrimeTime tool for static timing analysis, see

- [Before You Begin](#)
- [Setup Files](#)
- [Starting a PrimeTime Session](#)
- [Working With Licenses](#)
- [Entering pt_shell Commands](#)
- [Getting Help on the Command Line](#)
- [Using Tcl/Tk in PrimeTime](#)
- [The PrimeTime Static Timing Analysis Flow](#)
- [Ending a PrimeTime Session](#)

Before You Begin

Before you can use the PrimeTime tool, you must install the software and licenses for your site. For information about installation and licensing, see the documentation that comes with the software release.

Setup Files

When you start a PrimeTime session, the tool executes the commands specified in the PrimeTime setup files. In a setup file, you can set variables, the design environment, and your preferred working options.

The name of the setup file is `.synopsys_pt.setup`. If more than one of the following directories contains a `.synopsys_pt.setup` file, the tool executes the files in this order:

1. The admin/setup subdirectory of the Synopsys installation directory. For example, if the installation directory is `/usr/synopsys`, the setup file path is `/usr/synopsys/admin/setup/.synopsys_pt.setup`. Typically, this file contains setup information for all users at your site.
2. Your home directory – Typically, this setup file specifies your personal preferred working configuration.
3. The current working directory from which you started PrimeTime – Typically, this setup file specifies the environment for the current project.

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

Starting a PrimeTime Session

To start a PrimeTime session, enter the following command at the UNIX or Linux shell prompt:

```
% pt_shell
```

By default, this command starts the tool in the command-line interface (pt_shell):

```
PrimeTime (R)
Version J-2014.12 -- Dec 8, 2014
Copyright (c) 1988-2014 Synopsys, Inc.
ALL RIGHTS RESERVED
```

This program is proprietary and confidential ...

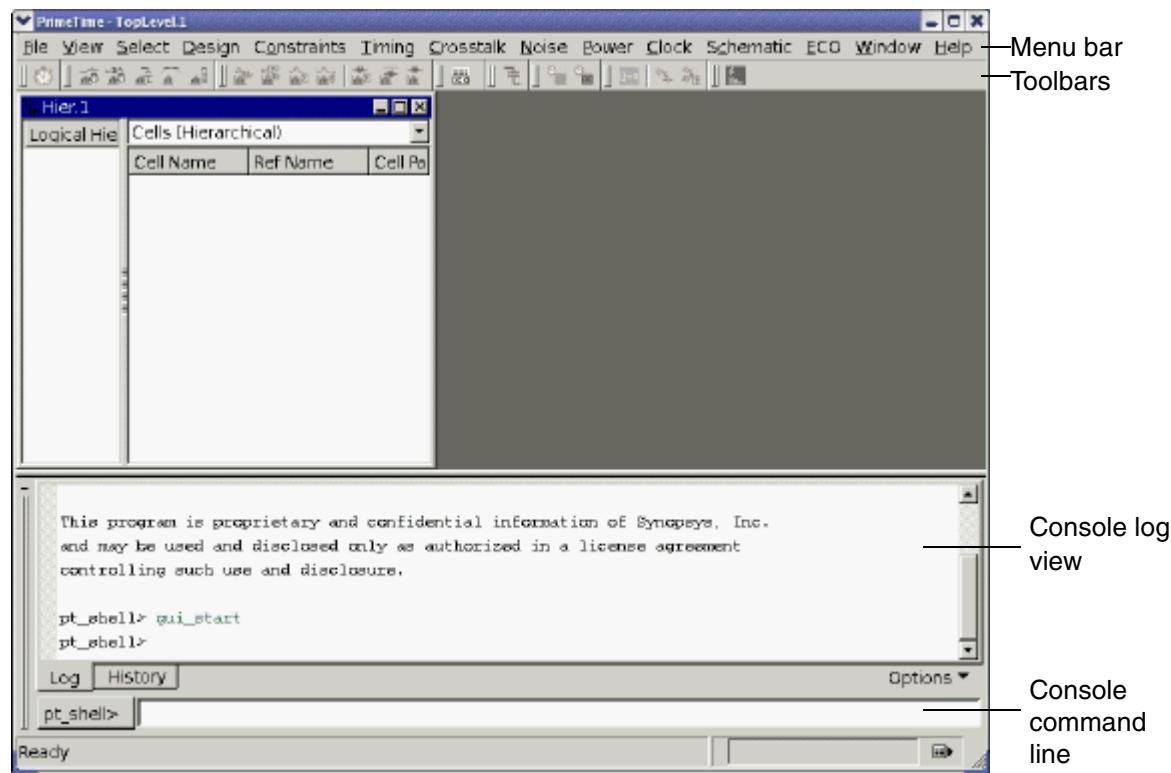
pt_shell>

To start the PrimeTime graphical user interface (GUI) instead of the command-line interface, enter:

```
% pt_shell -gui
```

This opens the PrimeTime main window, as shown in [Figure 2-1](#). You can enter pt_shell commands on the console command line at the bottom of the window. PrimeTime displays system messages in the console log view.

Figure 2-1 Initial PrimeTime Main Window



For more information about using the PrimeTime GUI, see [Graphical User Interface](#).

Working With Licenses

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

- [Listing the Licenses in Use](#)
 - [Checking Out Licenses](#)
 - [Enabling License Queuing](#)
 - [Releasing Licenses](#)
-

Listing the Licenses in Use

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

```
pt_shell> list_licenses
Licenses in use:
    PrimeTime (1)
    PrimeTime-SI (1)
```

To list the licenses that are checked out, use the `license_users` command. For example,

```
pt_shell> license_users
homer@eng1 PrimeTime, PrimeTime-PX
marge@eng2 PrimeTime, PrimeTime-SI, PrimeTime-ADV
2 users listed.
```

Checking Out Licenses

When you invoke the PrimeTime tool, the Synopsys Common Licensing software automatically checks out the appropriate license.

If you know the tools and interfaces you need, 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. For example,

```
pt_shell> get_license PrimeTime-SI
Information: Checked out license 'PrimeTime-SI' (PT-019)
```

After a license is checked out, it remains checked out until you release it or exit `pt_shell`.

Enabling License Queuing

If all PrimeTime licenses are in use, you can wait for available licenses by using license queuing. To enable license queuing, set the `SNPSLMD_QUEUE` environment variable to `true`. Then when launching `pt_shell`, the tool displays the following message:

Information: License queuing is enabled. (PT-018)

If you enable license queuing, you can run into a situation where two processes wait indefinitely for a license that the other process owns. To prevent this situation, set the following environment variables:

- `SNPS_MAX_WAITTIME` – This variable specifies the maximum time that a process waits to acquire the first key license. The default wait time is 259,200 seconds (72 hours). If a license is still not available after the specified time, the tool shows the following message:

Information: Timeout while waiting for feature 'PrimeTime'.
(PT-017)

- `SNPS_MAX_QUEUEETIME` – This variable specifies the maximum queue time for checking out subsequent licenses within the same `pt_shell` process, after you have successfully checked out the first license to start `pt_shell`. The default queue time is 28,800 seconds (eight hours). If the license is still not available after the specified time, the tool shows the following message:

Information: Timeout while waiting for feature 'PrimeTime SI'.
(PT-017)

As you take your design through the PrimeTime analysis flow, the queuing functionality might display other status messages, such as the following:

Information: Started queuing for feature 'PrimeTime SI'. (PT-015)

Information: Still waiting for feature 'PrimeTime SI'. (PT-016)

Information: Successfully checked out feature 'PrimeTime SI'. (PT-014)

Releasing Licenses

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

```
pt_shell> remove_license PrimeTime-SI
```

Entering pt_shell Commands

You interact with the tool by using pt_shell commands, which are based on the Tool Command Language (Tcl). The PrimeTime command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. To run the PrimeTime tool, you can

- Enter individual commands interactively at the pt_shell prompt
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl command scripts, which are text files containing pt_shell commands

To see the Tcl/Tk version used by your version of PrimeTime, enter

```
pt_shell> printvar tcl_version  
tcl_version = "8.5"
```

For more information about Tcl features that are supported in the PrimeTime tool, see

- [Tcl Packages and Autoload](#)
- [TclPro Toolkit](#)
- [Support of the incr Tcl Extension](#)

For general information about the Tcl command-line interface, see *Using Tcl With Synopsys Tools*, available on [SolvNet](#).

Tcl Packages and Autoload

PrimeTime supports the standard Tcl package and autoload facilities. However, the load command is not supported, so packages that require shared libraries cannot be used. PrimeTime is shipped with the standard implementations of the packages that are part of the Tcl distribution. For reference, you can find these packages below the root of the Synopsys installation in the auxx/tcllib/lib/tcl8.5 directory.

Add a new Tcl package to PrimeTime either by installing the package into the Synopsys tree or by adding a new directory to the `auto_path` variable in the application startup script (`.synopsys_pt.setup`).

PrimeTime provides these default locations for loading packages into the application:

- For application-specific Tcl packages, auxx/tcllib/primetime
- For packages that work with all Tcl-based Synopsys tools, auxx/tcllib/snps_tcl

For example, if you have a Tcl package called mycompanyPT that contains PrimeTime reporting facilities used by mycompany, you create a directory called mycompanyPT under the auxx/tcllib/primetime directory and then put the pkgIndex.tcl file and Tcl source files for the package into that directory. You can use the package in your PrimeTime scripts by using the following command:

```
package require ecompanyPT
```

For detailed information about Tcl packages, see Tcl reference books or go to <http://tcl.activestate.com/doc>.

TclPro Toolkit

TclPro is an open-source toolkit for Tcl programming. With the TclPro tools, you can do the following:

- [Check the Syntax in Scripts With the TclPro Checker](#)
- [Create Bytecode-Compiled Scripts With the TclPro Compiler](#)
- [Debug Scripts With the TclPro Debugger](#)

For more information about the TclPro tools, go to <http://tcl.sourceforge.net>.

Install TclPro Tools

To use any TclPro tools other than the syntax checker, you must install TclPro on your system. After you have installed TclPro on your system, specify the path to the installed program by setting the SNPS_TCLPRO_HOME environment variable. For example, if you installed TclPro version 1.5 at /u/tclpro1.5, set the SNPS_TCLPRO_HOME environment variable to point to that directory. PrimeTime uses this variable as a base for launching some of the TclPro tools. In addition, other Synopsys applications use this variable to link to the TclPro tools.

Check the Syntax in Scripts With the TclPro Checker

The Synopsys Syntax Checker, based on the TclPro Checker (procheck), helps you find syntax and semantic errors in your Tcl scripts. Everything that you need for syntax and semantic checking is shipped with PrimeTime; it is not necessary to download the TclPro tools to access syntax checking.

The Synopsys Syntax Checker, snps_checker, checks the following:

- Unknown options
- Ambiguous option abbreviation

- Use of exclusive options
- Missing required options
- Validation of literal option values for numerical range (range, <=, >=)
- Validation of one-of-string (keyword) options
- Recursion into constructs that have script arguments, such as the `redirect` and `foreach_in_collection` commands
- Warning of duplicate options overriding previous values

Running the Synopsys Syntax Checker

There are two ways to run `snps_checker` in the Synopsys environment. You can launch `snps_checker` from PrimeTime, or run it standalone.

To run `snps_checker` standalone, use the wrapper scripts provided with the PrimeTime installation. Running `snps_checker` directly does not work. For each platform there is a script in the appropriate bin directory. For PrimeTime you can find the script in the `sparcOS5/syn/bin/ptprocheck`, `linux/syn/bin/ptprocheck`, and so on.

To launch `snps_checker` from PrimeTime, you need to load a package provided with the installation, which is loaded as follows:

```
package require snpsTclPro
```

This makes the `check_script` command available. Pass the name of the script you want to check to the `check_script` command.

The following example is a script with errors that is tested in `snps_checker`:

```
create_clock [get_ports CLK] -period
create_clock [get_ports CLK] -period -12.2
sort_collection
set paths [get_timing_paths -nworst 10 -delay_type mx_fall -r]
my_report -from [sort_collection \
    [sort_collection $a b] {b c d} -x]
foreach_in_collection x $objects {
    query_objects $x
    report_timing -through $x -through $y -from a -from b -to z > r.out
}
all_fanout -from X -clock_tree
puts [pwd xyz]
```

After running the script, the extensions shipped with your release are loaded. In the example output it shows the Synopsys extensions being loaded. Each line in the output shows where a syntax or semantic error was detected. The offending command is shown with a caret (^) below the character that begins the offending token. [Example 2-1](#) shows the output from `snps_checker`.

Example 2-1 snps_checker Output

```
% /synopsys/2001.08/sparcOS5/syn/bin/ptprocheck ex1.tcl
Synopsys Tcl Syntax Checker - Version 1.0

Loading snps_tcl.pcx...
Loading primetime.pcx...
scanning: /disk/scripts/ex1.tcl
checking: /disk/scripts/ex1.tcl
/disk/scripts/ex1.tcl:1 (warnUndefProc) undefined
procedure:
get_ports
get_ports CLK
^
/disk/scripts/ex1.tcl:1 (SnpsE-MisVal) Value not specified
for
'create_clock -period'
create_clock [get_ports CLK] -period
^
/disk/scripts/ex1.tcl:2 (SnpsE-BadRange) Value -12.2 for
'create_clock -period' must be >= 0.000000
create_clock [get_ports CLK] -period -12.2
^
/disk/scripts/ex1.tcl:3 (SnpsE-MisReq) Missing required
positional options for sort_collection: collection criteria
sort_collection
^
/disk/scripts/ex1.tcl:4 (badKey) invalid keyword "mx_fall"
must
be: max min min_max max_rise max_fall min_rise min_fall
get_timing_paths -nworst 10 -delay_type mx_fall -r
^
/disk/scripts/ex1.tcl:4 (SnpsE-AmbOpt) Ambiguous option
'get_timing_paths -r'
get_timing_paths -nworst 10 -delay_type mx_fall -r
^
/disk/scripts/ex1.tcl:5 (warnUndefProc) undefined
procedure:
my_report
my_report -from [sort_collection \
^
/disk/scripts/ex1.tcl:5 (SnpsE-UnkOpt) Unknown option
'sort_collection -x'
sort_collection \
[sort_collection $a b] {b c d} -x
^
/disk/scripts/ex1.tcl:9 (SnpsW-DupOver) Duplicate option
'report_timing -from' overrides previous value
report_timing -through $x -through $y -from a -from b -to z > r.out
^
/disk/scripts/ex1.tcl:11 (SnpsE-Excl) Can only specify one
of
these options for all_fanout: -from -clock_tree
```

```
all_fanout -from X -clock_tree
^
/disk/scripts/ex1.tcl:12 (numArgs) wrong # args
pwd xyz
^
```

Limitations of the Synopsys Syntax Checker

The Synopsys Syntax Checker has the following limitations:

- Command abbreviation is not checked. Abbreviated commands show up as undefined procedures.
- Aliases created with the `alias` command are not expanded and show up as undefined procedures.
- A few checks done when the application is running might not be checked using the `snps_checker` environment. For example, some cases where one option requires another option are not checked.
- Script size is an issue with `snps_checker` and `TclPro 1.3` and `1.5`. Scripts up to a few thousand lines can be reasonably checked, but beyond that, CPU time becomes a factor. Do not try to check extremely large scripts using `snps_checker`.
- PrimeTime allows you to specify Verilog-style bus names on the command line without rigid quoting, for instance, `A[0]`. This format with indexes from 0 to 255 is checked. Wildcards `*` and `%` are also checked. Other forms, including ranges as `A[15:0]` show as undefined procedures, unless represented as `{A[15:0]}`.
- User-defined procedures enhanced with the `define_proc_attributes` command are not checked. Because such procedures are declared with the `args` argument, no semantic errors are reported.

Create Bytecode-Compiled Scripts With the `TclPro Compiler`

You can create bytecode-compiled scripts by using the `TclPro Compiler` (`procomp`). Bytecode-compiled scripts have the following advantages over ASCII scripts:

- Efficient to load
- Secure because the contents are not readable
- Body of compiled Tcl procedures is hidden

PrimeTime can load bytecode-compiled scripts. To load a bytecode-compiled script, use the `source` command. You do not need any files other than the application to load bytecode-compiled scripts.

Debug Scripts With the TclPro Debugger

The TclPro debugger (predebug) is the most complex tool in the package. The debugger is similar to most source code debuggers. You can step over lines in the script, step into procedures, set breakpoints, and so on. It is not standalone; it requires the application to be running while you debug the script.

Before running the TclPro debugger, you must specify the path to the TclPro installation by setting the `SNPS_TCLPRO_HOME` environment variable. To debug scripts, use the `debug_script` command.

To launch prodebug from PrimeTime, you need to load a package provided with the installation. The package is as follows:

```
package require snpsTclPro
```

This makes the `debug_script` command available. Pass the name of the script you want to debug to the `debug_script` command. This command launches prodebug, and connects PrimeTime to it. The script is instrumented, and then you are ready for debugging. You can make subsequent calls to instrument the script by sourcing the file with the `source` command or with the `debug_script` command. For more information, see the TclPro documentation shipped with the debugger.

Support of the incr Tcl Extension

PrimeTime supports the incr Tcl (itcl) extension, which adds object-oriented programming constructs to Tcl.

For details about incr Tcl, see the Tcl Developer Xchange website:

<http://www.tcl.tk/man/itcl3.1/index.html>

Getting Help on the Command Line

To get help when running the PrimeTime tool, use the following resources:

- Command help
 - To list all PrimeTime commands, organized by command group, enter
`pt_shell> help`
 - To show a brief description of a command, enter
`pt_shell> help command_name`

- To show all options and arguments for a command, enter
`pt_shell> command_name -help`
- Man pages
 To display the man page of a command, variable, or message, enter
`pt_shell> man command_variable_or_message_name`

Using Tcl/Tk in PrimeTime

The PrimeTime command interface is based on Tool Command Language (Tcl) and the Tk toolkit, which is used in many other Synopsys tools. For general information about Tcl and its usage in Synopsys command shells, see *Using Tcl With Synopsys Tools*, available on [SolvNet](#).

To see the Tcl/Tk version used by your version of PrimeTime, enter

```
pt_shell> printvar tcl_version
```

For more information about Tcl features that are supported in the PrimeTime tool, see

- [Tcl Packages and Autoload](#)
- [TclPro Toolkit](#)

The PrimeTime Static Timing Analysis Flow

To perform PrimeTime static timing analysis, follow the typical flow outlined in [Table 2-1](#).

Table 2-1 Typical PrimeTime Static Timing Analysis Flow

Step	Task	Typical commands	Related topics
1	Read in the design data, which includes a gate-level netlist and associated logic libraries.	<code>set search_path</code> <code>set link_path</code> <code>read_db</code> <code>read_verilog</code> <code>link_design</code>	Design Data

Table 2-1 Typical PrimeTime Static Timing Analysis Flow (Continued)

Step	Task	Typical commands	Related topics
2	Specify timing and design rule constraints.	<code>set_input_delay</code> <code>set_output_delay</code> <code>set_min_pulse_width</code> <code>set_max_capacitance</code> <code>set_min_capacitance</code> <code>set_max_fanout</code> <code>set_max_transition</code>	Design Constraints
3	Specify clock characteristics.	<code>create_clock</code> <code>set_clock_uncertainty</code> <code>set_clock_latency</code> <code>set_clock_transition</code>	Clocks
4	Specify timing exceptions.	<code>set_multicycle_path</code> <code>set_false_path</code> <code>set_disable_timing</code>	Timing Paths and Exceptions
5	Specify the environment and analysis conditions such as operating conditions and delay models.	<code>set_operating_conditions</code> <code>set_driving_cell</code> <code>set_load</code> <code>set_wire_load_model</code>	Operating Conditions, Delay Calculation
6	Specify case and mode analysis settings.	<code>set_case_analysis</code> <code>set_mode</code>	Case and Mode Analysis
7	Back-annotate delay and parasitics.	<code>read_sdf</code> <code>read_parasitics</code>	Back-Annotation
8	(Optional) Apply variation.	<code>read_aocvm</code> <code>set_aocvm_coefficient</code> <code>set_aocvm_table_group</code> <code>set_ocvm_table_group</code> <code>set_timing_derate</code>	Variation

Table 2-1 Typical PrimeTime Static Timing Analysis Flow (Continued)

Step	Task	Typical commands	Related topics
9	Specify power information.	<pre>create_power_domain create_supply_net create_supply_set create_supply_port connect_supply_net set_voltage</pre>	Multivoltage Design Flow
10	(Optional) Specify options and data for signal integrity analysis.	<pre>set_app_var si_enable_analysis true read_parasitics -keep_capacitive_coupling</pre>	Signal Integrity Analysis
11	(Optional) Apply options for specific design techniques.	<pre>set_latch_loop_breaker set_latch_loop_breaker get_latch_loop_groups set_multi_input_ switching_coefficient</pre>	Advanced Analysis Techniques , PrimeTime ADV Advanced Latch Analysis , Multi-Input Switching Analysis , Scaling for Multirail Cells , Fast Multidrive Delay Analysis , Parallel Driver Reduction

Table 2-1 Typical PrimeTime Static Timing Analysis Flow (Continued)

Step	Task	Typical commands	Related topics
12	Check the design data and analysis setup.	check_timing check_constraints report_design report_port report_net report_clock report_wire_load report_path_group report_cell report_hierarchy report_reference report_lib	Checking the Design and Analysis Setup
13	Perform a full timing analysis and examine the results.	report_global_timing report_timing report_constraint report_bottleneck report_analysis_coverage report_delay_calculation update_timing	Reporting and Debugging Analysis Results , Graphical User Interface
14	(Optional) Perform ECO to fix timing violations and recover power.	set_eco_options fix_eco_drc fix_eco_timing fix_eco_power write_changes	ECO Flow
15	Save the PrimeTime session.	save_session	Saving a PrimeTime Session

The PrimeTime reference methodology (<https://solvnet.synopsys.com/rmgen>) provides scripts that implement this flow.

Ending a PrimeTime Session

You can end the session and exit the tool at any time. To learn more, see

- [Saving a PrimeTime Session](#)
 - [Exiting a PrimeTime Session](#)
 - [Accessing the Session History in the Command Log File](#)
-

Saving a PrimeTime Session

Before ending a PrimeTime working session, you might want to save the data of the current session. If you need to examine the analysis results later, save the session with the `save_session` command. To later restore the session, use the `restore_session` command, which takes you to the same point in the analysis.

Saving and restoring the session is useful in the following situations:

- You use a script to run a PrimeTime session overnight. The script uses `save_session` after the final `update_timing` command. Later, you can restore the session and examine the results using `gui_start`, `report_delay_calculation`, `report_timing`, and so on.
- You save the current state of the analysis as a checkpoint, which you can restore later in case of an error or unexpected change in the analysis environment.
- You save the current session and then restore it multiple times to apply different chip operating modes. This is an alternative to saving and applying the timing data in SDF format.

A saved and restored session retains the following information:

- Linked design and loaded libraries
- Clocks, timing exceptions, and other constraints
- Operating conditions
- Back-annotated SDF delays and parasitics
- Variable settings
- Netlist edits (`insert_buffer`, `size_cell`, `swap_cell`)
- Analysis data
- Cross-coupled delay data and noise data

The save and restore feature is intended only as a PrimeTime session tool, not a database archiving tool or a method for storing test cases. The tool does not save or restore the following information:

- Collections of derived data types
- Tcl procedures and command history
- GUI state (timing path tables, schematics, histograms, and so on)
- Quick timing model generation state (between the `create_qtm_model` and `save_qtm_model` commands)
- Context characterization state (between the `characterize_context` and `write_context` commands)

Although you can use different platforms to save and restore a session, you must use the same PrimeTime version, including service packs.

Exiting a PrimeTime Session

To exit a PrimeTime session, enter the `quit` or `exit` command at the `pt_shell` prompt:

```
pt_shell> exit
Timing updates: 2 (1 implicit, 1 explicit)
    (0 incremental, 1 full, 1 logical)
Noise updates: 0 (0 implicit, 0 explicit) (0 incremental, 0 full)
Maximum memory usage for this session: 318.43 MB
CPU usage for this session: 2 seconds (2.06 seconds aggregate)
Elapsed time for this session: 47 seconds
Diagnostics summary: 2 errors, 1 warning, 3 informations
```

Thank you for using `pt_shell`!

Accessing the Session History in the Command Log File

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

The tool creates the `pt_shell_command.log` file in the current working directory. Any existing log file with the same name is overwritten. Before you start a new PrimeTime session, ensure that you rename any log file that you want to keep.

To specify a different name for the command log file, set the `sh_command_log_file` variable in your setup file. You cannot change this variable during a working session.

3

Managing Performance and Capacity

To learn how to manage the performance and capacity of the PrimeTime tool, see

- [High Capacity Mode](#)
- [Fast Analysis Mode](#)
- [Threaded Multicore Analysis](#)
- [Distributed Multi-Scenario Analysis](#)
- [Memory and CPU Resource Usage Reports](#)
- [Performance Profiling of Tcl Scripts](#)

High Capacity Mode

By default, PrimeTime runs in high capacity mode, which reduces the peak memory footprint and makes runtime trade-offs between performance and capacity, while producing the same results as normal analysis. High capacity mode is compatible with all analysis flows, including flat and hierarchical flows; however, it is most useful with clock reconvergence pessimism removal (CRPR) enabled.

To use high capacity mode, you do not need to change your existing scripts or flows. In high capacity mode, the tool temporarily stores data to a local disk partition specified by the `pt_tmp_dir` variable; the default is the `/tmp` directory. When you exit the session, the consumed disk space is automatically released. For effective use of this mode, the available capacity of the local disk on the host machine is as large as the physical RAM.

To specify the effort level of high capacity mode, set the `sh_high_capacity_effort` variable to `low`, `medium`, `high`, or `default` (equivalent to the `medium` level). You must set this variable before using the `set_program_options` command. This variable simply specifies the tradeoff between capacity and performance; it does not affect the analysis results.

To disable high capacity mode, use the `set_program_options` command. The `sh_high_capacity_enabled` read-only variable shows whether high capacity mode is enabled. For example:

```
pt_shell> set_program_options -disable_high_capacity
Information: high capacity analysis mode disabled. (PTHC-001)
pt_shell> printvar sh_high_capacity_enabled
sh_high_capacity_enabled = "false"
```

Fast Analysis Mode

Fast analysis mode speeds up analysis while maintaining reasonable accuracy. This is useful during early analysis runs when accuracy is less critical. Fast analysis mode takes precedence over other options or variables to provide higher performance. For example, default reporting focuses only on violating paths. In fast analysis mode, path-based analysis is turned off, and the `report_timing` command uses graph-based analysis.

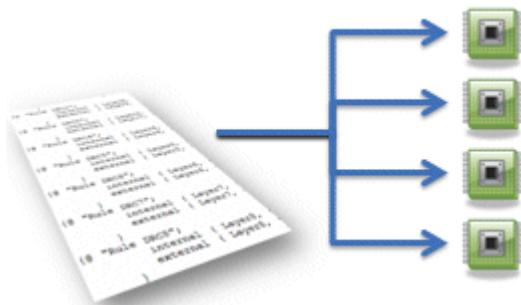
By default, fast analysis mode is disabled. To enable fast analysis mode, use the `set_program_options` command with the `-enable_fast_analysis` option at the beginning of your script, before loading designs and libraries. To determine whether fast analysis mode is enabled, check the setting of the `sh_fast_analysis_mode_enabled` read-only variable.

When accuracy is critical, such as during late stage timing and ECO closure runs, ensure that fast analysis mode is disabled.

Threaded Multicore Analysis

PrimeTime can perform threaded multicore analysis, which improves performance on a shared memory server by running multiple threads on the cores available on that server, as shown in [Figure 3-1](#).

Figure 3-1 Threaded Multicore Analysis



When using threaded multicore analysis, PrimeTime executes multiple time-consuming algorithms in parallel to yield a faster run without requiring any script changes. PrimeTime runs the following commands in parallel: `get_timing_paths`, `read_parasitics`, `read_verilog`, `report_analysis_coverage`, `report_constraint`, `report_timing`, `save_session`, `update_timing`, `update_noise`, and `write_sdf`.

To learn how to use threaded multicore analysis, see

- [Configuring Threaded Multicore Analysis](#)
- [Executing Commands in Parallel](#)
- [Running Threaded Multicore Path-Based Analysis](#)

Configuring Threaded Multicore Analysis

By default, threaded multicore analysis is enabled.

The following factors limit the number of cores used:

- The `set_host_options` command with the `-max_cores` option
For example, for single-core analysis, specify `-max_cores 1`.
- The physical number of cores available on the server
For example, in a dual core server, threads are launched to use only the two available cores.
- The 16-core maximum per PrimeTime license

When you use the `read_parasitics` command, parasitic reading is performed using a separate process launched within PrimeTime. The parasitic reading is performed in parallel with other commands, such as the `read_sdc` command. This parasitic reading process is transparent and requires no explicit control to be enabled. However, you can turn it off by using the `set_host_options -max_cores 1` command. To get the maximum benefit out of the separate parasitic reading process, use the `read_parasitics` command immediately after linking the design, which is the recommended usage for this command rather than using it later in the flow.

To control over threading in multicore analysis, use the `multi_core_allow_overthreading` variable. When the variable is set to `true` (the default), simultaneously active threads can exceed the maximum limit set for the CPU cores. To guarantee that the process cores utilization does not exceed the maximum core limit, set the variable to `false`; this could result in reduced multicore performance.

The parasitic reading process is enabled only in high capacity mode. To disable the process, use the `set_program_options -disable_high_capacity` command. The temporary files written during the parasitic reading process are large parasitic files that are stored in the `pt_tmp_dir` directory. These files are automatically deleted by PrimeTime at the end of the session. The log of the parasitic commands goes to a file specified by the `parasitics_log_file` variable; the default is the `parasitics_command.log` file in the current working directory.

Executing Commands in Parallel

Additional performance improvements can be obtained by using other parallelization techniques such as parallel command execution. You can use parallel command execution to execute post-timing update commands in parallel with each other. You can run Tcl procedures, post-timing update reporting, and analysis commands using parallel command execution. To ensure effectiveness, launch the longest running command first.

When using parallel command execution, the maximum number of parallel commands executed at one time is determined by the `-max_cores` option of the `set_host_options` command.

Note:

You cannot use parallel command execution for commands that update the design, such as ECO, timing, or noise updates.

To execute commands in parallel, use the following commands in your scripts:

- [The parallel_execute Command](#)
- [The redirect -bg Command](#)
- [The parallel_foreach_in_collection Command](#)

The parallel_execute Command

Specify the `parallel_execute` command with the commands that you want to execute in parallel. For example,

```
update_timing -full
parallel_execute {
    report_cmd1 rpt_file1
    report_cmd2 rpt_file2
    report_cmd3 rpt_file3
}
...
```

The tool blocks the `pt_shell` until all of the embedded jobs are completed.

If you specify `set_host_options -max_cores 1`, PrimeTime runs in single-core analysis mode; all commands within the `parallel_execute` command are executed serially in the specified order.

The redirect -bg Command

If you use the `redirect -bg` command, the tool activates a post-update command in the background. The main `pt_shell` keeps running while the `redirect -bg` command queues the jobs and waits for available cores. For example,

```
update_timing -full
redirect -bg -file rpt_file1 {report_cmd1}
redirect -bg -file rpt_file2 {report_cmd2}
redirect -bg -file rpt_file3 {report_cmd3}
...
```

If you specify `set_host_options -max_cores 1`, PrimeTime runs in single-core analysis mode; the `redirect -bg` command runs the commands in the foreground, as if the `-bg` option was not specified.

Note:

If the foreground execution encounters `exit` and there is at least one active background `redirect -bg` command still executing, the `exit` request is blocked until all background commands complete.

The parallel_foreach_in_collection Command

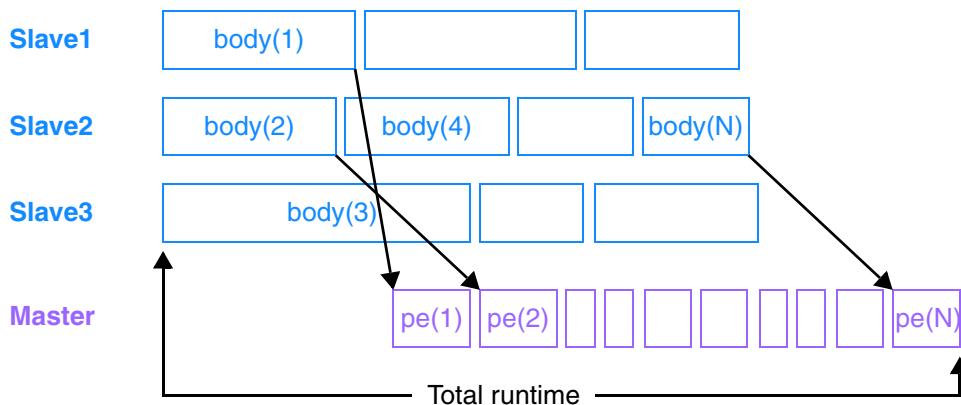
To speed up iterations over collections of objects, use the `parallel_foreach_in_collection` command. This command is similar to the `foreach_in_collection` command but with the addition of parallel processing on multiple

cores. You can use this command with the `post_eval` command, which specifies a script that is executed by the master process, as shown in the following syntax:

```
parallel_FOREACH_in_collection iterator_variable collections {
    body_script_executed_by_slaves_in_parallel
    ...
    post_eval {
        script_executed_by_master_in_sequence
        ...
    }
}
```

The example in [Figure 3-2](#) shows the `parallel_FOREACH_in_collection` command execution, where the processing time proceeds from left to right. The blue boxes represent iterations of the body script executed in parallel by slave processes. The `parallel_FOREACH_in_collection` command automatically schedules and balances the execution of N iterations to optimize the utilization of available cores.

Figure 3-2 Execution of parallel_FOREACH_in_collection



While executing the body script, the slave processes submit the script specified by the `post_eval` command to the master process; this is represented in [Figure 3-2](#) by the purple boxes. The master process executes the `post_eval` commands in sequence. Therefore, for maximum performance, keep `post_eval` commands as fast as possible and limited to only parts of the script that cannot be parallelized, such as changing constraints, setting user-defined attributes, and modifying persistent variables to aggregate results. It is also important to avoid excessive variable transfer, in particular when dealing with large lists and arrays.

The example in [Figure 3-3](#) shows how to use the `parallel_FOREACH_in_collection` command to find and report the endpoint with the most violating paths.

Figure 3-3 Example of parallel_foreach_in_collection and post_eval Commands

```

set endpoints [add_to_collection [all_registers -data_pins -async_pins]
              [all_outputs] ]
set most_violators 0
set ep_with_most_violators {}
parallel_foreach_in_collection endpoint $endpoints {
    set paths [get_timing_paths -to $endpoint -nw 1000
               -slack_lesser_than 0]
    set num_violators [sizeof_collection $paths] Body script, executed by slaves in parallel
}

post_eval {
    if { $num_violators > $most_violators } {
        set most_violators $num_violators
        set ep_with_most_violators $endpoint
    }
}
echo "endpoint [get_object_name $ep_with_most_violators]
      has $most_violators violators"
post_eval (PE) script, executed by the master in sequence

```

For information about converting an existing `foreach_in_collection` loop to a `parallel_foreach_in_collection` loop, see the man page for the `parallel_foreach_in_collection` command.

Running Threaded Multicore Path-Based Analysis

PrimeTime runs threaded multicore analysis for path-based analysis when you use the `-pba_mode` path and `-pba_mode` exhaustive options. This feature is supported for all command variations of path-specific and exhaustive recalculation. For example,

- `report_timing -pba_mode exhaustive ...`
- `report_timing -pba_mode path ...`
- `report_timing -pba_mode path $path_collection`
- `get_timing_paths -pba_mode exhaustive ...`
- `get_timing_paths -pba_mode path ...`
- `get_timing_paths -pba_mode path $path_collection`

If multiple cores are not available, PrimeTime runs the standard single-core path-based analysis.

Distributed Multi-Scenario Analysis

Verifying a chip design requires several PrimeTime runs to check correct operation under different operating conditions (such as extreme temperatures and operating voltages) and different operating modes (such as mission or test mode). A specific combination of operating conditions and operating modes for a given chip design is called a *scenario*.

The number of scenarios for a design is

$$\text{Scenarios} = [\text{Sets of operating conditions}] \times [\text{Modes}]$$

PrimeTime can analyze several scenarios in parallel with *distributed multi-scenario analysis* (DMSA). Instead of analyzing each scenario in sequence, DMSA uses a master PrimeTime process that sets up, executes, and controls multiple slave processes—one for each scenario. You can distribute the processing of scenarios onto different hosts running in parallel, thus reducing the overall turnaround time to analyze the timing results from multiple scenarios. In addition, total runtime is reduced when you share common data between different scenarios.

A single script can control many scenarios, making it easier to set up and manage different sets of analyses. This capability does not use *multithreading* (breaking a single process into pieces that can be executed in parallel). Instead, it provides an efficient way to specify the analysis of different operating conditions and operating modes for a given design and to distribute those analyzes onto different hosts.

To learn how to use DMSA, see

- [Definition of Terms](#)
- [Overview of the DMSA Flow](#)
- [Distributed Processing Setup](#)
- [Common Image Generation](#)
- [Modeling Support](#)
- [Achieving Optimal Performance](#)
- [Manipulating Variables](#)
- [Merged Reporting](#)
- [Saving and Restoring Your Session](#)
- [License Resource Management](#)
- [Database Support](#)
- [Controlling Fault Handling](#)

- [Messages and Log Files](#)
 - [Limitations](#)
 - [Single and Multiple Scenario Constraint Reports](#)
 - [DMSA Commands, Options, and Variables](#)
-

Definition of Terms

The following terms describe aspects of distributed processing:

baseline image

Image that is produced by combining the netlist image and the common data files for a scenario.

command focus

Current set of scenarios to which analysis commands are applied. The command focus can consist of all scenarios in the session or just a subset of those scenarios.

current image

Image that is automatically saved to disk when there are more scenarios than hosts, and the slave process must switch to work on another scenario.

master

Manages the distribution of scenario analysis processes.

scenario

Specific combination of operating conditions and operating modes.

session

Current set of scenarios selected for analysis.

slave

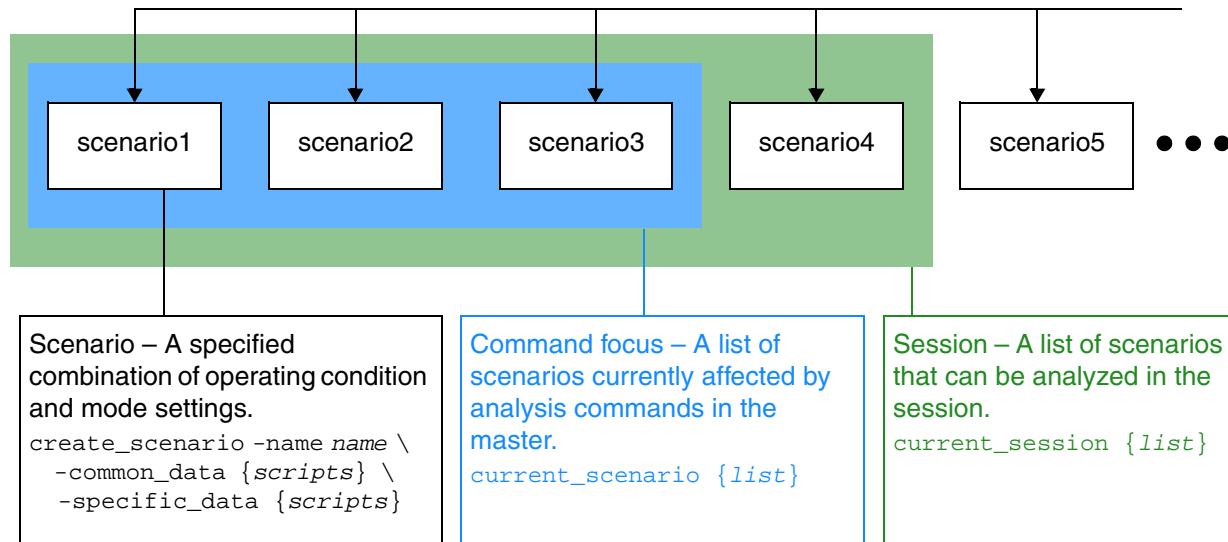
Started and controlled by the master to perform timing analysis for one scenario.

task

Self-contained piece of work defined by the master for a slave to execute.

[Figure 3-4](#) shows the relationships between the scenarios, session, and command focus.

Figure 3-4 Scenarios, Session, and Command Focus



A scenario describes the specific combination of operating conditions and operating modes to use when analyzing the design specified by the configuration. There is no limit to the number of scenarios that you can create (subject to memory limitations of the master). Each scenario is created using the `create_scenario` command. The command specifies the scenario name and the names of the scripts that apply the analysis conditions and mode settings for the scenario.

The scripts are divided into two groups: common-data scripts and specific-data scripts. The common-data scripts are shared between two or more scenarios, whereas the specific-data scripts are specific to the particular scenario and are not shared. This grouping helps the master process manage tasks and to share information between different scenarios, minimizing the amount of duplicated effort for different scenarios.

A session describes the set of scenarios you want to analyze in parallel. You can select the scenarios for the current session using the `current_session` command. The current session can consist of all defined scenarios or just a subset of those scenarios.

The command focus is the set of scenarios affected by PrimeTime analysis commands entered at the PrimeTime prompt in the master process. The command focus can consist of all scenarios in the current session or just a subset of those scenarios. A subset of the scenarios in the current session can be specified using the `current_scenario` command. By default, all scenarios in the current session are in command focus, unless you change them.

Overview of the DMSA Flow

To learn about the basics of the multi-scenario flow, see

- [Preparing to Run DMSA](#)
- [DMSA Usage Flow](#)

Preparing to Run DMSA

Before you start your multi-scenario analysis, you must set the search path and create a `.synopsys_pt.setup` file:

- [Setting Your Search Path](#)
- [.synopsys_pt.setup File](#)

Setting Your Search Path

In multi-scenario analysis, you can set the `search_path` variable only at the master. When reading in the search path, the master resolves all relative paths in the context of the master. The master then automatically sets the fully resolved search path at the slave. For example, you might launch the master in the `/remote1/test/ms` directory, and set the `search_path` variable with the following command:

```
set search_path ". . . ./scripts"
```

The master automatically sets the search path of the slave to the following:

```
/remote1/test/ms /remote1/test /remote1/ms/scripts
```

The recommended flow in multi-scenario analysis is to set the search path to specify the location of

- All files for your scenarios and configurations
- All Tcl scripts and netlist, SDF, library, and parasitic files to be read in a slave context

Note:

For best results, avoid using relative paths in slave context scripts.

.synopsys_pt.setup File

The master and slaves source the same set of `.synopsys_pt.setup` files in the following order:

1. PrimeTime install setup file at

```
install_dir/admin/setup/.synopsys_pt.setup
```

2. Setup file in your home directory at
 `~/.synopsys_pt.setup`
3. Setup file in the master launch directory at
 `$sh_launch_dir/.synopsys_pt.setup`

To control whether commands are executed in the current `pt_shell` mode, set the `pt_shell_mode` variable to `primetime`, `primetime_master`, or `primetime_slave`. For example,

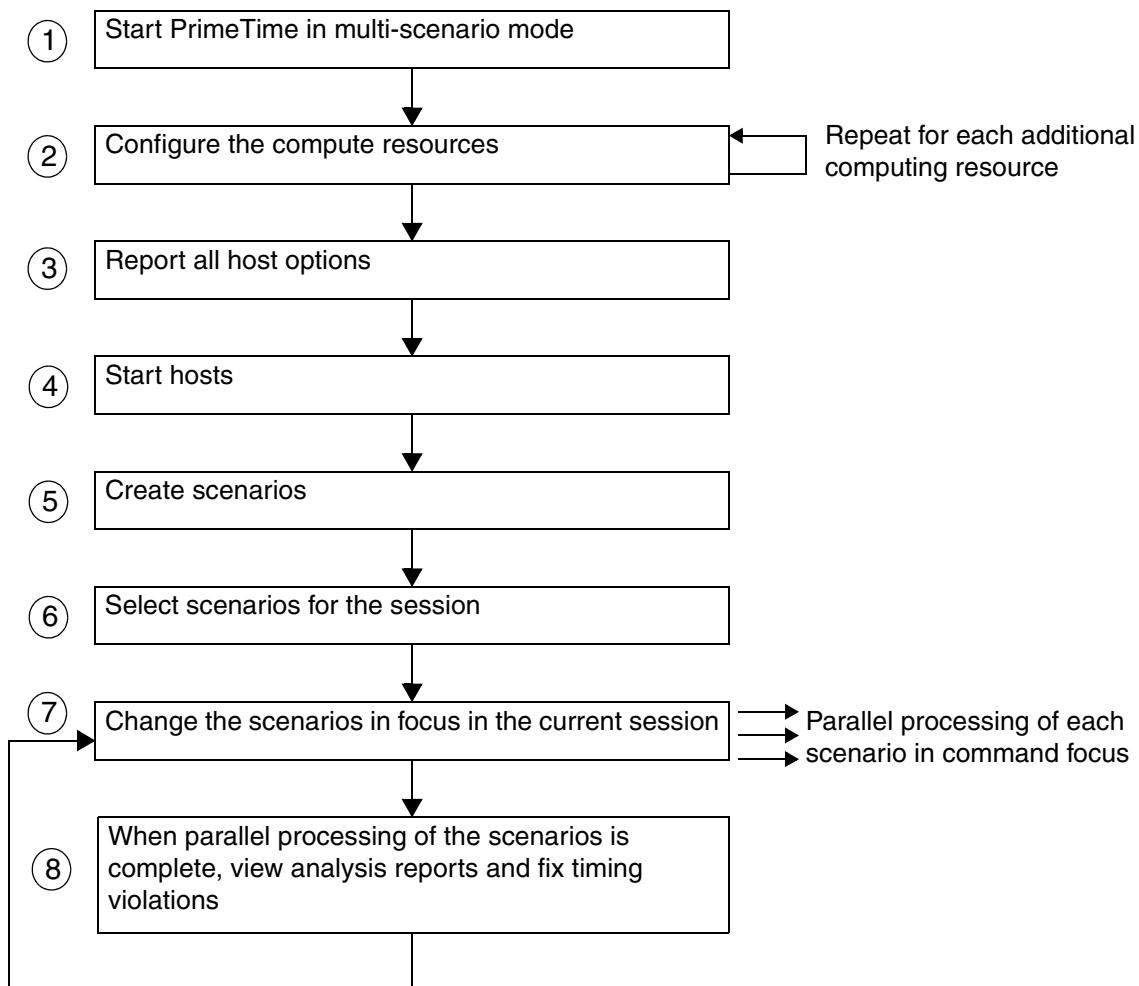
```
if { $pt_shell_mode == "primetime_master" } {  
    set multi_scenario_working_directory "./work"  
}
```

DMSA Usage Flow

You can use the DMSA capability for timing analysis in PrimeTime and PrimeTime SI as well as for power analysis in PrimeTime PX. This dramatically reduces the turnaround time. For more information about using DMSA in PrimeTime PX, see the “Distributed Peak Power Analysis” section in the *PrimeTime PX User Guide*.

From the `pt_shell` prompt of the master PrimeTime process, you can initiate and control up to 256 slave processes (see [Limitations](#)). [Figure 3-5](#) shows the sequence of steps in a typical multi-scenario analysis.

Figure 3-5 Distributed Processing Usage Flow



A typical multi-scenario analysis has the following steps:

- Start PrimeTime in the multi-scenario analysis mode by using the `pt_shell` command with the `-multi_scenario` option.
- Configure the computer resources that you want to use during the timing update and reporting phases by using the `set_host_options` command. This command does not start the host, but rather sets up the host options for that host. In the following example, the host option is named `ptopt030`, `rsh` is used to connect to the host, and a maximum of three cores per process are specified for the compute resources:

```
pt_shell> set_host_options -name my_opts -max_cores 3 \
    -num_processes 4 -submit_command "/usr/bin/rsh -n" ptopt030
```

Control the maximum number of cores by specifying the remote core count with the `set_host_options` command in the master script, as shown in the previous example.

3. Use the `report_host_usage` command to report all host options that are set. This command also reports peak memory and CPU usage for the local process and all distributed processes that are already online. The report displays the host options specified, status of the distributed processes, number of CPU cores each process uses, and licenses used by the distributed hosts.

To display information about a specific host, use the `host_option_names` option. If you have not defined any host options using the `set_host_options` command, all host options are displayed.

The following report shows the host options created by the `set_host_options` command and also the real time data associated with the processes:

```
pt_shell> report_host_usage
*****
Report : host_usage
...
*****
Options Name      Host          32Bit      Num Processes
-----
my_opts          ptopt030      N           4
Usage limits (cores)
Options Name                      #      Effective
-----
(local process)                  -          2
my_opts5
-----
Total                           2
Memory usage
Options Name      #      Peak mem (MB)
-----
(local process)   -      24.93
Performance
Options Name      #      CPU time (s)      Elapsed Time (s)
-----
(local process)   -          3                  25
```

4. You must use the `start_hosts` command to request compute resources and bring the hosts online. Otherwise, the `current_session` command fails.

Note:

You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before starting compute resources.

For more information about these variables, see [Starting PrimeTime for Distributed Processing](#).

To provide more information about the hosts, such as the status and process information, use the `report_host_usage` command after starting the distributed processes. For example,

```
pt_shell> set_host_options -name my_opts -max_cores 3 \
                     -num_processes 4 -submit_command "/usr/bin/rsh -n" ptopt030
1
pt_shell> report_host_usage
*****
Report : host_usage
...
*****
Options Name      Host          32Bit      Num Processes
-----
my_opts           ptopt030      N          4
Options Name      #      Host Name      Job ID      Process ID      Status
-----
my_opts           1      ptopt030      28857      28857      ONLINE
                  2      ptopt030      28912      28912      ONLINE
                  3      ptopt030      28966      28966      ONLINE
                  4      ptopt030      29020      29020      ONLINE
Usage limits (cores)
Options Name      #      Effective
-----
(local process)      -      2
                      1      -
                      2      -
                      3      -
                      4      -
-----
Total                           2
Memory usage
Options Name      #      Peak Memory (MB)
-----
(local process)    -      24.93
my_opts           1      26.31
                  2      26.31
                  3      26.31
                  4      26.31
```

Performance

Options Name	#	CPU Time (s)	Elapsed Time (s)
(local process)	-	3	25
my_opts	1	1	14
	2	3	25
	3	3	25
	4	1	15

5. Create the scenarios with the `create_scenario` command. Each `create_scenario` command specifies a scenario name and the PrimeTime script files that apply the conditions for that scenario.

Note:

You must complete steps 1 through 5 before you can go to the next step. Otherwise, the `current_session` command fails.

6. Select the scenarios for the session using the `current_session` command. The command specifies a list of scenarios previously created with the `create_scenario` command.
7. (Optional) Change the scenarios in the current session that are in command focus, using the `current_scenario` command. The command specifies a list of scenarios previously selected with the `current_session` command. By default, all scenarios in the current session are in command focus.
8. View the analysis report and fix validation issues.
- Start processing the scenarios by executing the `remote_execute` command or performing a merged report command at the master. For more information about merged reports, see [Merged Reporting](#).
 - When the processing of all scenarios by the slave processes is complete, you can view the analysis reports. Locate the reports generated by the `remote_execute` command under the directory you specified with the `multi_scenario_working_directory` variable, as described in [Distributed Processing Setup](#). Alternatively, if you issue the `remote_execute` command with the `-verbose` option, all information is displayed directly to the console at the master. The output of all merged reporting commands is displayed directly in the console at the master.
 - Use ECO commands to fix timing and design rule violations. For more information about ECO commands, see [ECO Flow](#).

Distributed Processing Setup

For distributed processing of multiple scenarios, you must first invoke PrimeTime in DMSA mode. You then manage your compute resource, define the working directory, create the scenarios, and specify the current session and command focus. For details, see

- [Starting PrimeTime for Distributed Processing](#)
- [Managing Compute Resources](#)
- [Creating Scenarios](#)
- [Specifying the Current Session and Command Focus](#)
- [Checking the Setup](#)
- [Executing Commands Remotely](#)
- [Script Example](#)

Starting PrimeTime for Distributed Processing

To start PrimeTime in distributed multi-scenario analysis (DMSA) mode, use the `-multi_scenario` option when you start the PrimeTime shell. For example,

```
% pt_shell -multi_scenario
```

PrimeTime starts up and displays the `pt_shell` prompt just as in a single scenario session. A distributed multi-scenario analysis is carried out by one master PrimeTime process and multiple PrimeTime slave processes. The master and slave processes interact with each other using full-duplex network communications.

The master process generates analysis tasks and manages the distribution of those tasks to the slave processes. It does not perform timing analysis and does not hold any data other than the netlist and the multi-scenario data entered. The command set of the master is therefore restricted to commands for carrying out master features.

To specify the directory where working files for distributed processing are stored, use the `multi_scenario_working_directory` variable. Set this variable to the name of a directory that is write-accessible by you, the master process, and all slave processes. Typically, the slave processes are running on different hosts than the master; therefore, the working directory needs to be network accessible. If you do not explicitly set this variable, the current working directory of the master is used.

Note:

You must set the `multi_scenario_working_directory` variable before starting compute resources.

Specify the file in which to write all error, warning, and information messages issued by the slaves using the `multi_scenario_merged_error_log` variable. Any error, warning, or information message that is issued by more than one slave is merged into a single entry in the merged error log. Each entry contains the scenario name from which the data originated. If a file is specified for this variable, it appears in the directory specified by the `multi_scenario_working_directory` variable. The `multi_scenario_merged_error_log` variable limits the number of messages of a particular type written to the log on a per task basis. The default is 100. Therefore, when using the default, a maximum of 100 RC messages is written to the merged error log per task. For more information, see [Merged Error Log](#).

Note:

You must set the `multi_scenario_merged_error_log` variable before starting compute resources.

Upon startup, every PrimeTime process, both master and slave, looks for and sources the `.synopsys_pt.setup` setup file just as in normal mode. Each process also writes its own separate log file. For more information, see [Log Files](#).

Managing Compute Resources

Dynamically manage your compute resources and allocate slave processes by adding hosts. Managed computing resources enables PrimeTime to allocate pooled computing resources as they become available. If an LSF, GRD or proprietary (generic) computing management capability is available for load balancing, you can allocate hosts from these systems to the compute resources used for processing scenarios.

The distributed processing mode supports the following hardware management architectures:

- LSF (Load Sharing Facility from Platform Computing, Inc.)
- Grid computing (Global Resource Director from Gridware, Inc.)
- Generic computing farm

Unmanaged computing resources are computed servers/workstations that are not load balanced. They can be added to the compute resources used for processing scenarios and are internally load balanced during DMSA.

Setting Up Distributed Host Options

You can allocate slave hosts as compute resources with the `set_host_options` command. Each slave process invoked by the master is a single PrimeTime session that is accessible only by the master process. It is not possible to manually launch a PrimeTime process in slave mode.

When the master begins the DMSA on the slaves, it generates tasks for the slave processes to execute. Any one task can apply to only one scenario and any one slave process can be configured for only one scenario at a time. If there are more scenarios than slave processes to run them, some slave processes have to swap out one scenario for another so that all tasks get executed.

The process of swapping out one scenario for another is an expensive, time-consuming operation. It is also important, however, to avoid overloading hosts by specifying more processes on a host than there are available CPUs in the host. If more processes are specified than there are CPUs available, a penalty is incurred for swapping in and out slave processes on a CPU. Therefore, the slave processes do not run concurrently.

Specify the maximum number of cores each slave process can use by running the `set_host_options` command at the master. You can also specify this command in `.synopsys_pt.setup` file that the remote processes execute; however, it is better to use the command at the master. If the command is set in both the file and at the master, the maximum cores specification in the `.synopsys_pt.setup` file takes precedence.

For optimal system performance of DMSA, match the number of slave processes, scenarios in command focus, and available CPUs. Optimal system performance is achieved when every scenario is executed in its own slave process on its own CPU.

Note:

You can use the `set_host_options` command to specify the host options for the slave processes. However, the processes are only started when the `start_hosts` command is called.

To remove all of the host options and terminate all processes, use the `remove_host_options` command with no arguments. You can also remove a list of predefined host options by specifying the host option names. For example, if there are host options called `host1`, `host2`, and `host3` and you want to remove `host1` and `host3`, use this command:

```
pt_shell> remove_host_options {host1 host3}
```

Note:

By default, all processes are stopped and the host options are removed when you use the `remove_host_options` command.

Configure the Distributed Environment

Configure the distributed environment by using the `set_distributed_parameters` command. For user-defined configurations to take effect, you must issue the `set_distributed_parameters` command before the `start_hosts` command.

Starting the Compute Resources

Use the `start_hosts` command to start hosts specified by the `set_host_options` command. The `start_hosts` command requests compute resources and brings the hosts online. When the `start_hosts` command returns, execution proceeds if at least one host is available. As additional remote processes become available during the analysis, they are dynamically made available for use by the analysis. If no remote processes were available when the `start_hosts` command returns, an information message is issued.

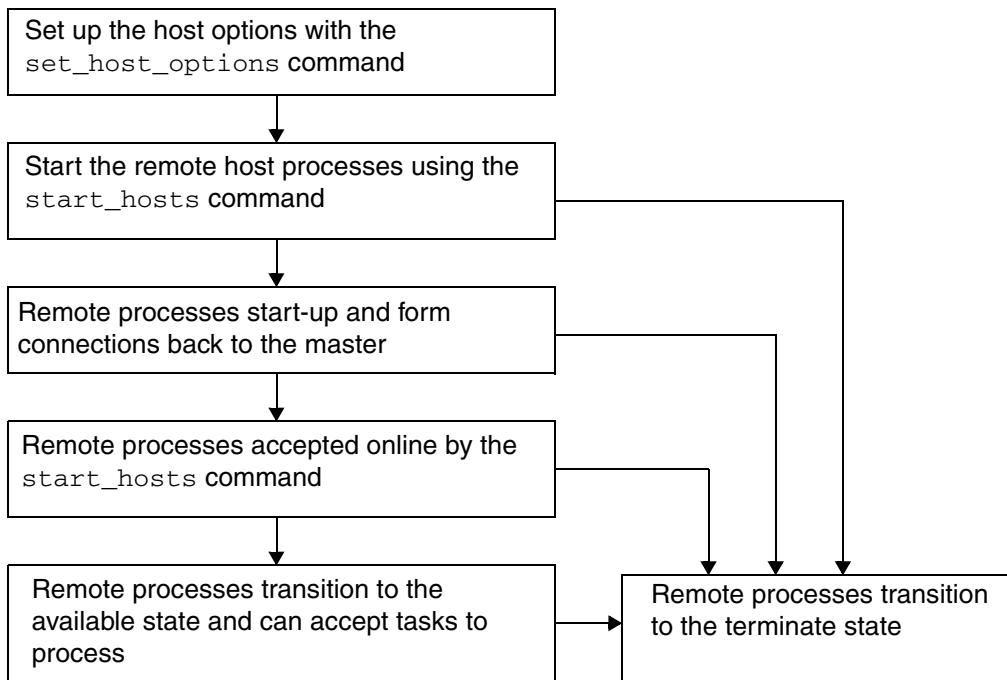
The `start_hosts` command completes when one of the following conditions is met:

- All remote host processes become available.
- If specified, the `-min_hosts` minimum number of remote host processes becomes available.
- The time-out expires.

To control the wait time for remote processes, use the `-timeout` and `-min_hosts` options of the `start_hosts` command. To specify how long the `start_hosts` command waits for remote processes to become available, use the `-timeout` option; the default timeout is 21,600 seconds (6 hours). Use the `-min_hosts` option to specify the minimum number of hosts needed to enable the `start_hosts` command to return and allow execution to proceed.

Although you can set the session to begin analysis when just one host is available, remember that the number of available hosts directly affects the ability of PrimeTime to simultaneously analyze scenarios. When there is an imbalance between scenarios and hosts, expensive image swapping occurs until sufficient hosts become available for each scenario to occupy a remote host. If you set the `-min_hosts` option to a low value, or you set the `-timeout` option to a small value, an imbalance is likely and leads to image swapping until sufficient hosts become available.

When the `start_hosts` command is called, the previously added hosts begin to transition through the states of their lifecycle shown in [Figure 3-6](#).

Figure 3-6 Lifecycle States

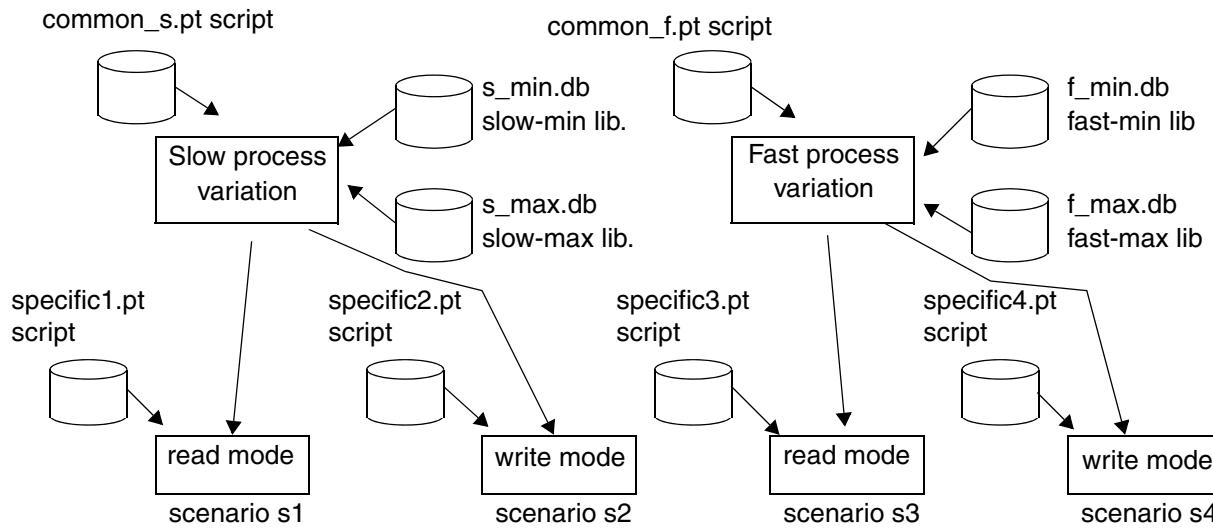
Creating Scenarios

A scenario is a specific combination of operating conditions and operating modes for a given configuration. Create a scenario with the `create_scenario` command. For example,

```
pt_shell> create_scenario -name scen1 \
    -common_data {common_s.pt} \
    -specific_data {specific1.pt}
```

This command specifies a scenario name, the common-data script files, and the specific-data script files. The scripts associated with the scenario are divided into two categories: common data (shared by two or more scenarios) and specific data (not shared by any other scenarios). Common-data scripts are included in the baseline image generation process, whereas specific-data scripts are applied only to the specified scenario. Organizing scripts into categories allows the master to share design data between groups of scenarios where appropriate, thereby reducing the total runtime for all scenarios. The baseline image generation process is described in greater detail in the [Common Image Generation](#). For example, consider the scenarios shown in [Figure 3-7](#) that shows the types of scripts that can be applied as common-data and specific-data scripts.

Figure 3-7 Multi-Scenario Setup Example



This design is to be analyzed at two process variation extremes, called slow and fast, and two operating modes, called read and write. Thus, there are four scenarios to be analyzed: slow-read, slow-write, fast-read, and fast-write. You can add more conditions and modes by creating several scenarios as indicated in [Figure 3-7](#).

The `common_all.pt` script reads in the design and applies constraints that are common to all scenarios. The `common_s.pt` script is shared between some, but not all, scenarios. The `specific1.pt` script directly contributes to specifying conditions of an individual scenario and is not shared at all between different scenarios.

Use the `-image` option of the `create_scenario` command to create scenarios from a previously saved scenario image. This image can be generated from a standard PrimeTime session, a session saved from the DMSA master, or a session saved from the current image generated during a DMSA session.

Using Common or Specific Script Scenarios

To create a scenario using a common or specific script, use the `create_scenario` command with the `-common_data` or `-specific_data` options, respectively. When you do this, you must create voltage scaling groups just before running the `link_design` in one of the scripts used to set up a scenario.

If voltage scaling groups are created in a script that is part of a scenario's `-specific_data` specification, only that scenario sees those scaling groups. If voltage scaling groups are created in a script that is part of a scenario's `-common_data` specification and another scenario shares the same common specification, both of those scenarios see the same voltage scaling groups.

Removing Scenarios

To remove a scenario previously created with the `create_scenario` command, use the `remove_scenario` command:

```
pt_shell> remove_scenario s1
```

Specifying the Current Session and Command Focus

The current session consists of a set of scenarios that you want to analyze. Use the `current_session` command to establish the current session. For example,

```
pt_shell> current_session {s2 s3 s4}
```

The command lists the scenarios to be part of the current session, all of which must be valid scenario names already defined by the `create_scenario` command. By default, all listed scenarios are in command focus. The command focus is the set of scenarios in the current session to which analysis commands are applied.

After using `current_session`, you can further narrow the command focus by using the `current_scenario` command to select a subset of the scenarios in the current session:

```
pt_shell> current_scenario {s3}
```

This command is useful during interactive analysis. For example, you might want to modify the timing characteristics of a path in one scenario by annotating a load on a particular net, and then run a new delay calculation on the net's driving cell or recheck the timing of the whole path for that scenario alone.

After narrowing the command focus, you can restore the command focus to all scenarios in the session by using the `-all` option:

```
pt_shell> current_scenario -all
```

Note:

Changing the *current-session* list requires images to be rebuilt.

Checking the Setup

To check the distributed processing setup before you begin DMSA, use the `report_multi_scenario_design` command. This command creates a detailed report about user-defined multi-scenario objects and attributes.

Executing Commands Remotely

After you set up the analysis with the `create_scenario`, `current_session`, and (optionally) `current_scenario` commands, you can start analysis of the scenarios in command focus.

The `remote_execute` command is used in the master to explicitly evaluate commands, variables, and expressions in the slave context. There is a slight difference in the way that you can use the command, depending on whether you are executing commands in batch or interactive mode.

In batch mode, the commands are contained in scripts. For example, you might execute the following command at the UNIX prompt to invoke PrimeTime and execute a multi-scenario analysis in batch mode:

```
% pt_shell -multi_scenario -file script.tcl
```

For a complete example of the `script.tcl` file, see [Script Example](#). To use the `remote_execute` command, enclose the list of commands in a Tcl string. Separate the command with semicolons so that they execute one at a time rather than as a group. To evaluate subexpressions and variables remotely, generate the command string using curly braces. For example,

```
remote_execute {  
    # variable all_in evaluated at slave  
    report_timing -from $all_in -to [all_outputs]  
}
```

In this example, the `report_timing` command is evaluated at the slave, and the `all_in` variable and the `all_outputs` expression are evaluated in a slave context.

To evaluate expressions and variables locally at the master, enclose the command string in quotation marks. All master evaluations must return a string. For example,

```
remote_execute "  
    # variable all_out evaluated at master  
    report_timing -to $all_out  
"
```

In this example, the `report_timing` command is evaluated at the slave, and the `all_out` variable is evaluated at the master. Upon issuing the `remote_execute` command, the master generates tasks for execution in all scenarios in command focus.

[Example 3-1](#) shows how to use the `remote_execute` command. [Example 3-2](#) shows that if you use the `-pre_commands` option with the `remote_execute` command, the listed commands are executed before the command specified for remote execution. [Example 3-3](#)

shows that if you use the `-post_commands` option, the listed commands are executed after the commands specified for remote execution.

Example 3-1 Slave and Master Context Variables

```
remote_execute {set x 10}
# Send tasks for execution in all scenarios in command focus
set x 20
remote_execute {report_timing -nworst $x}
# Leads to report_timing -nworst 10 getting executed at the slaves

remote_execute "report_timing -nworst $x"
# Leads to report_timing -nworst 20 getting executed at the slaves
```

Example 3-2 Using the -pre_commands Option With the remote_execute Command

```
remote_execute -pre_commands {cmd1; cmd2; cmd3}
"report_timing"
# On the slave host, execute cmd1, cmd2, and cmd3 before
# executing report_timing on the master
```

Example 3-3 Using the -post_commands Option With the remote_execute Command

```
remote_execute -post_commands {cmd1; cmd2; cmd3}
"report_timing"
# On the slave host, execute cmd1, cmd2, and cmd3 after
# executing report_timing on the master
```

You can use the `remote_execute` command with the `-verbose` option to return all slave data to the master terminal screen, instead of piping it to the `out.log` file in the working directory of the DMSA file system hierarchy.

The ability to execute netlist editing commands remotely extends to all PrimeTime commands, except when using the following commands:

- Explicit `save_session` (slave only)
- Explicit `restore_session`
- `remove_design`

Script Example

The following script is a typical multi-scenario master script.

```
set multi_scenario_working_directory "./ms_session_1"
set multi_scenario_merged_error_log "merged_errors.log"

script.tcl
#####
# Start of specifying the set-up
# Start of specifying the first task
#####
```

```
set search_path "./scripts"

# Add 2 32-bit hosts to the compute resources; one host
# from platinum1 and another host from platinum2
set_host_options -32bit -num_processes 1 platinum1
set_host_options -32bit -num_processes 1 platinum2

# Generates a detailed report that describes all host options
# that you have added.
report_host_usage

# Start the compute resources for processing the scenarios
start_hosts

# Create the scenarios
create_scenario \
    -name scenario1 \
    -common_data common_data_scenarios_s1_s3.tcl \
    -specific_data specific_data_scenario_s1.tcl
create_scenario \
    -name scenario2 \
    -common_data common_data_scenarios_s2_s4.tcl \
    -specific_data specific_data_scenario_s2.tcl

create_scenario \
    -name scenario3 \
    -common_data common_data_scenarios_s1_s3.tcl \
    -specific_data specific_data_scenario_s3.tcl

create_scenario \
    -name scenario4 \
    -common_data common_data_scenarios_s2_s4.tcl \
    -specific_data specific_data_scenario_s4.tcl

#####
# End of specifying the setup
#####
# Set the scenarios for the current session
current_session {scenario1 scenario2}

# cmd1,cmd2,cmd3, are placeholders for typical PrimeTime commands or
# scripts that need to be executed on scenario1 and scenario2
#
# remote_execute -pre_commands {"cmd1" "cmd2" "cmd3"} {report_timing}
# For example,

remote_execute -pre_commands {source constraint_group1.tcl\
    source constraint_group2.tcl} {report_timing}
```

```
#####
# End of specifying the first task
#####
quit
# analysis is finished
# all slaves are inactivated and licenses are returned
```

A task is a self-contained block of work that a slave process needs to execute to perform some function. A slave process starts execution of a task for the specified scenario as soon as it gets the licenses appropriate to that task. Otherwise, it waits for the licenses it requires.

This script performs the following functions:

1. The `initial set` command sets the search path at the master (used for finding scripts).
2. The `set_host_options` command specifies the host options for the compute resources.
3. The `report_host_usage` command generates a detailed report that describes all of the host options that you have added.
4. The `start_hosts` command starts the host processes specified by the `set_host_options` command.
5. The `create_scenario` commands create four scenarios named s1 through s4. Each command specifies the common-data and specific-data scripts for the scenario.
6. The `current_session` command selects s3 and s4 for the current session, which is also the command focus by default. (You can use the `current_scenario` command to narrow the focus further.)
7. The `remote_execute -pre_commands` command begins the task generation process and execution of those tasks in scenarios s3 and s4. The resulting logs contain all the output from the commands that were executed.

Because the `-pre_commands` option is specified, the two `source` commands are executed before the `report_timing` command. The command list can contain any valid PrimeTime commands or sourcing of scripts that contain valid PrimeTime commands (excluding all commands that alter the netlist or save or restore).

8. The `quit` command terminates the slave processes and master processes in an orderly manner and returns all the checked-out PrimeTime licenses back to the central license pool.
9. Locate the report and log files in the full file paths listed later, assuming the master `pt_shell` was launched from the directory `/rpt/new`.
 - Master command log:
`/rpt/new/pt_shell_command.log`

- Errors generated by the slaves and returned to the master and merged for reporting purposes:
`/rpt/new/ms_session_1/merged_errors.log`
- Slave output log for work done for the session, that is, netlist image generation:
`/rpt/new/ms_session_1/default_session/out.log`
- Slave output logs generated for s1 and s2:
`/rpt/new/ms_session_1/s3/out.log`
`/rpt/new/ms_session_1/s4/out.log`
- Slave command logs:
`/rpt/new/ms_session_1/pt_shell_command_log/`
 `platinum1_pt_shell_command.log`
`/rpt/new/ms_session_1/pt_shell_command_log/`
 `platinum2_pt_shell_command.log`

Common Image Generation

In the `script.tcl` example, the main task that is user-directed consists of `cmd1`, `cmd2`, `cmd3` and the `report_timing` command. However, there are other internal tasks that need to happen before you can execute the task containing these commands. This process is called common image generation. The following are the processes that take place before executing your commands but after specifying all scenarios, sessions, and so on.

The first set of tasks the master generates and delegates to arbitrary slave processes is to generate the baseline images. Every scenario in command focus requires access to a baseline image that consists of the netlist image and the common data files for that scenario. The slave process assigned to execute a baseline image generation task executes the common data files for that scenario in the order in which they were specified at the time the scenario was created using the `create_scenario` command. After this, the slave process generates the baseline image for that scenario.

Before any other types of tasks can proceed in the scenario that is having its baseline image generated, the baseline image generation process must complete successfully. Given this dependence, failure of this process results in failure of the multi-scenario analysis only for those scenarios that depend on that failed image generation. The MS-020 warning message is issued and resources are reduced. Scenarios that have access to a successfully generated baseline image can proceed normally.

For each scenario in command focus that provides a baseline image, that image is written to the `scenario_name/baseline_image` directory under the multi-scenario working directory.

The third set of tasks that the master generates and delegates to slave processes is the execution of the user-specified commands in the scenarios that are in command focus. The

slave process assigned a command execution task uses the baseline image for the scenario in which these commands need to be executed (this configures the slave process for the scenario), executes the specific data files for that scenario, and then executes the commands specified.

Where there are more scenarios in command focus than processes to execute the tasks for those scenarios, the slave process needs to save out a current image for one scenario while it proceeds with task execution for a different scenario. If this swapping is necessary, the slave process saves the current image in the scenario_name/current_image directory under the multi-scenario working directory.

Modeling Support

DMSA supports interface logic models and extracted timing models without requiring any special setup. Additionally, you do not need to change any modeling arrangements made to your design. For more information about modeling support in PrimeTime, see [Timing Models for Hierarchical Analysis](#).

Achieving Optimal Performance

It is important to remember that the automatic reconfiguring of slave processes to run different tasks in different scenarios can be an expensive operation; minimize automatic reconfiguring to maximize system throughput. The multi-scenario process operates to minimize the swapping out of scenarios, but is subject to the limitations imposed by the number of scenarios in command focus and the number of slave processes added. If there is a significant imbalance of scenarios to slave processes, as a whole the system performance is degraded.

You can achieve optimal performance from DMSA if there is a slave process for every scenario in command focus, a CPU available to execute each slave process, and a license for each feature needed for every slave process. This results in all slave processes executing concurrently, allowing maximum throughout for the system.

Manipulating Variables

You can control the value of a variable in either the master or any of its slaves during DMSA. To learn how to manipulate variables, see

- [Master Context Variables](#)
- [Slave Context Variables and Expressions](#)
- [Setting Distributed Variables](#)

- [Getting Distributed Variable Values](#)
- [Merging Distributed Variable Values](#)

Master Context Variables

Here is an example of a master context variable.

```
set x {"ffa/CP ffb/CP ffc/CP"}
report_timing -from $x
```

Notice how x is forced to be a string.

Slave Context Variables and Expressions

The master has no knowledge of variables residing on the slave, but can pass variable or expression strings to the slave for remote evaluation. To pass a token to the slave for remote evaluation, use curly braces. For example, say you had the following scenarios:

- Scenario 1: x has value of ff1/CP
- Scenario 2: x has value of ff2/CP

The `report_timing -from {$x}` command reports the worst paths from all paths starting at ff1/CP in the context of scenario 1, and all paths starting at ff2/CP in the context of scenario 2.

Setting Distributed Variables

With the `set_distributed_variables` command you can set the values of variables at the slaves from the master. The variable you set can be a Tcl variable, collection, array, or list.

To set variables in multiple scenarios, you first create a Tcl array of the desired values for those scenarios, then use the `set_distributed_variables` command to distribute the settings to the scenarios. For example, suppose that you have two scenarios, s1 and s2, running under the control of a master. From the master, you want to set the value of two variables x and y at the slave level, so that x=10 and y=0 in s1, and x=0 and y=15 in s2. You create an array, indexed by scenario name, containing the values to set in the slaves. From the master, execute the following commands:

```
pt_shell> array set x {s1 10 s2 0}
pt_shell> array set y {s1 0 s2 15}
pt_shell> set_distributed_variables {x y}
```

This sends the values specified in the arrays x and y from the master to the slaves, and creates the variables x and y at the slaves if they do not already exist, and sets them to the desired values.

When variables are pushed from the master into the scenarios, nonarray variables at the master are supported. The specified variable is created at each of the scenarios. This avoids the need for creating a temporary array at the master to push a simple variable value. For example,

```
pt_shell> set test_var 123
123
pt_shell> set_distributed_variables test_var
```

Getting Distributed Variable Values

Using the `get_distributed_variables` command provides the ability to get the values of variables set in scenarios in command focus. The variables retrieved by the command can be a Tcl variable, collection, array, or list. The retrieved values are returned to the master process and stored in a Tcl array, indexed by scenario name. For example, the following session retrieves slack attribute values from two scenarios:

```
current_session {scen1 scen2}

remote_execute {
    set pin_slack1 [get_attribute -class pin UI/Z slack];
    set pin_slack2 [get_attribute -class pin U2/Z slack]
}

get_distributed_variables {
    pin_slack1 pin_slack2
}
```

The session returns two arrays, `pin_slack1` and `pin_slack2`. Indexing `pin_slack1(scen1)` returns the scalar value of slack at U1/Z in context of scenario `scen1`. Indexing `pin_slack1(scen2)` returns the scalar value of slack at U1/Z in the context of scenario `scen2`. When retrieving a collection, you must indicate what attributes you are interested in for the command to generate the desired list. The following session retrieves timing paths from two scenarios:

```
current_session {scen1 scen2}
remote_execute {
    set mypaths [get_timing_paths -nworst 100]
}
get_distributed_variables mypaths -attributes (slack)
```

The session returns an array called `mypaths`. Indexing `mypaths(scen1)` yields a collection of the 100 worst paths from scenario `scen1`, ordered by slack. Similarly, indexing `paths(scen2)` yields a collection of the 100 worst paths from `scen2`.

When retrieving variables at the master, any existing array variables are updated with the retrieved values.

Merging Distributed Variable Values

You can use the `-merge_type` and `-null_merge_method` options of the `get_distributed_variables` command to merge variable values that come from the scenarios as they are retrieved. This is especially useful when you are writing customized DMSA scripts.

By default, the `get_distributed_variables` command brings back a scenario variable as an array. With the `-merge_type` option, the values can be merged back into a variable during retrieval. The merge types that are available are `min`, `max`, `average`, `total`, `list`, `sum`, and `unique_list`. For example, to keep the numerically smallest value of the `worst_slack` from all scenarios, use these commands:

```
pt_shell> get_distributed_variables {worst_slack} -merge_type min
1
pt_shell> echo $worst_slack
-0.7081
```

You can also merge arrays of one or more dimensions and keep the worst value for each array entry. For example, if you have the following scenarios:

```
scenario best_case:
set slacks(min,pinA) -0.2044
set slacks(max,pinA) 0.3084

scenario type_case:
set slacks(min,pinA) -0.0523
set slacks(max,pinA) 0.0901
scenario worst_case:
set slacks(min,pinA) 0.1015
set slacks(max,pinA) -0.7081
```

To obtain the merge results listed, use these commands:

```
pt_shell> get_distributed_variables {slack} -merge_type min
1
pt_shell> echo $slacks(min, pinA)
-0.2044
pt_shell> echo $slacks(max, pinA)
-0.7081
```

The `-merge_type` option can also merge together lists of values from the scenarios. For example, to obtain a unique list of all clock names from every scenario, use these commands:

```
pt_shell> remote_execute {set clock_names \
    [get_object_name [all_clocks]]}
pt_shell> get_distributed_variables clock_names -merge_type list
1
pt_shell> echo $clock_names
CLK33 CLK66 CLK50 CLK100 ATPGCLOCK JTAGCLK
```

The merged list is built from the list contents from each scenario with any duplicate list entries of identical values omitted.

To merge lists of object name lists in a multi-scenario analysis flow, a specialized `unique_list` merging mode is available. This mode is useful when you want to bring back multiple lists of design object names, such as cell, pin, or net names, and there might be small variations between the scenario results due to differing constants, constraints, or interface logic model (ILM) netlists, or differences in the design. Where possible, the `unique_list` merging mode keeps the sublists separate so that gaps can be filled, but merges these sublists together as needed. Using this merging mode is similar to applying the list merging type to each of the sublists, but additionally merges together any sublists that share common items. For example, consider following variables,

```
scenario best_case:  
set cells {{U1 U2} {U4 U5} {U7 U8}}  
  
scenario type_case:  
set cells {{U2 U3} {U6 U7}}  
  
scenario worst_case:  
set cells {{U1} {U5 U6}}
```

Retrieving these variables with the `unique_list` merging mode gives the following result:

```
{{U1 U2 U3} {U4 U5 U6 U7 U8}}
```

The merged list is built from the list contents from each scenario, with any duplicate list entries of identical value omitted.

The `-null_merge_method` option specifies the behavior when a null (empty) value of {} is encountered during the numerical merge operations. The available options are

- `ignore` (the default) – Ignores any null values and processes only non-null values.
- `error` – Terminates the merging process and issues an error.
- `override` – Null values win the merging process, and a null is returned.

For example, you might have the libraries in the `type_case` scenario misconfigured such that the `BUFFX2` buffer cell cannot be resolved, and the value for this array entry was set to

null. With the default null merging method of `ignore`, the null is ignored and the remaining `BUFFX2` values are averaged as shown in the following example.

```
scenario best_case:
set buffer_delay(BUFFX1) 0.105
set buffer_delay(BUFFX2) 0.210

scenario type_case:
set buffer_delay(BUFFX1) 0.130
set buffer_delay(BUFFX2) {}

scenario worst_case:
set buffer_delay(BUFFX1) 0.140
set buffer_delay(BUFFX2) 0.280
```

If it is possible that some data entries can be null, you can specify the `-null_merge_method override` option to detect these cases of incomplete data array entries. In this situation, the null wins during the numerical merging process and overrides the other numerical values for that array entry, allowing the missing data to be detected in a controlled manner. For example:

```
pt_shell> get_distributed_variables {buffer_delay} -merge_type average \
          -null_merge_method override
1
pt_shell> echo $buffer_delay(BUFFX2)
pt_shell> echo $buffer_delay(BUFFX1)
0.125
```

Alternatively, you can use the null merging method of `error` so that the `get_distributed_variables` command terminates with an error. For example,

```
pt_shell> get_distributed_variables {buffer_delay} -merge_type average \
          -null_merge_method error
Error: Found null value in merged_object when -null_merge_method was
      set to error (ZDPP-022)
```

The error null merging method can be used as an assertion to trap cases that should never happen. For example, if the array data is always complete, it can be used to assert an error if some data is null for some unexpected reason.

Note:

A `null` data value is different from when a value is undefined. If array entries are undefined, the `-null_merge_method` option does not apply and the merging process operates normally on the remaining data values. For example, in the `type_case` scenario, if the scenario scripting omitted the `BUFFX2` array entry rather than setting it to `null`, it would be impossible to detect any missing data after applying the merging process.

Merged Reporting

Executing reporting commands in multi-scenario analysis can generate large amounts of data and detailed information about each scenario. The sheer quantity of data can make it difficult to identify the critical information. To manage these large data sets, and to help you identify critical issues across the design, PrimeTime provides the merged reporting capability.

Merged reporting is a mechanism that works with all the scenarios in command focus to automatically eliminate redundant data across scenarios and sort the data in order of criticality. You can treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. The merge reporting capability applies to the `get_timing_paths`, `report_analysis_coverage`, `report_clock_timing`, `report_constraint`, `report_si_bottleneck`, and `report_timing` commands.

Using Merged Reporting

To use merged reporting for a timing report, issue the `report_timing` command at the master as you would in a single scenario session of PrimeTime. Executing the command in this way is called executing the command in the master context. This results in the master and slave processes working together to execute the `report_timing` command in all the scenarios in command focus, to produce a final merged report that is displayed at the master console. Each of the reports include the scenario name from which it came.

Generating Merged Reports

When issuing commands in a master context, the commands are operating in a distributed manner, which necessitates specifying some of their options in a slightly different way compared to a single scenario session.

Using commands in DMSA is similar to using these same commands in a single scenario session of PrimeTime. All of the options exist in the merged mode. However, you must first invoke the `-multi_scenario` option to use the `-pre_commands` and `-post_commands` options. These options allow you to specify a string of commands to execute in the slave context before and after executing a merged report command. In situations where there are more scenarios than slave processes, using these options can reduce the amount of swapping out of scenarios by performing the `-pre_commands` or `-post_commands` options and the merged reporting command in series, as opposed to multiple swapping out and restoring the same scenarios for each command.

To learn more about the reporting commands, see

- [`get_timing_paths`](#)
- [`report_analysis_coverage`](#)

- [report_clock_timing](#)
- [report_constraint](#)
- [report_si_bottleneck](#)
- [report_timing](#)

get_timing_paths

Using the `get_timing_paths` command from the master process returns a collection of timing path objects from each scenario, based on the parameters you set in the command. The timing path collection is condensed to meet the requirements you specify (such as, using `-nworst` or `-max paths`). It returns a merged output collection of the worst timing paths across all scenarios according to the reporting criteria set.

To find the scenario a particular timing path object belongs to, use the `scenario` attribute on the timing path object itself. To generate a list of attributes you are interested in, use the `-attributes` option with the `get_timing_paths` command.

report_analysis_coverage

The `report_analysis_coverage` command reports the coverage of timing checks over all active scenarios in the current session. The timing check is defined by the constrained pin, the related pin, and the type of the constraint. The timing check status is reported

- As violated if the calculated signal propagation time along the particular data path does not meet timing requirements (constraints)
- As met if timing requirements are satisfied
- As untested if the timing check was skipped

Timing checks are included in the Untested category only if they are untested in all scenarios. If a timing check is tested in at least one scenario, the timing check is reported as tested (showing in either the Violated or Met category) because it has been successfully exercised in some scenario. This allows meaningful reporting to be performed across scenarios with significantly differing behavior (for example, functional versus test modes).

The output for the merged reports is similar to the single scenario analysis coverage reports; however, the merged reports contain a list of scenarios. [Example 3-4](#) shows the summary merged analysis coverage report.

Example 3-4 Summary Merged Analysis Coverage Report

```
pt_shell> report_analysis_coverage
*****
Report : analysis_coverage
Design : multi-scenario design
...
*****
Scenarios: SLOW_OPER, NOM_OPER, FAST_TEST

Type of Check      Total      Met      Violated      Untested
-----
setup             8          3 (37.5%)    3 (37.5%)    2 ( 25%)
hold              8          3 (37.5%)    3 (37.5%)    2 ( 25%)
-----
All Checks        16         6 (37.5%)    6 (37.5%)    4 ( 25%)
```

If a timing check is untested in all scenarios in the command focus, instead of a list of scenarios, the all tag is listed in the scenario column; however, if a timing check is tested in at least one scenario, the check is reported as tested, either violated or met, because it has been successfully exercised in some scenarios. [Example 3-5](#) shows a report where some scenarios are met and others are untested.

Example 3-5 Detailed Merged Analysis Coverage Report

```
pt_shell> report_analysis_coverage -status_details {untested met} \
           -sort_by slack
*****
Report : analysis_coverage
           -status_details {untested met}
           -sort_by slack
Design : multi-scenario design
...
*****
Scenarios: scen1, scen2, scen3

Type of Check      Total      Met      Violated      Untested
-----
setup             15          2 ( 13%)    12 ( 80%)    1 ( 7%)
hold              15          12 ( 80%)    2 ( 13%)    1 ( 7%)
-----
All Checks        30         14 ( 47%)    14 ( 47%)    2 ( 7%)
```

Constrained Pin	Related Pin	Clock	Check Type	Scenario	Slack	Reason
ffd/CR	CP(rise)		hold	scen2 scen1	untested	constant_disabled
ffd/CR	CP(rise)		setup	scen2 scen1	untested	constant_disabled
ffa/D	CP(rise)	clk2	hold	scen2	0.14	
ffa/D	CP(rise)	CLK	hold	scen2	0.14	
ffb/D	CP(rise)	CLK	hold	scen2	0.14	
ffb/D	CP(rise)	clk2	hold	scen2	0.14	
ffd/D	CP(rise)	CLK	hold	scen2	0.14	
ffd/D	CP(rise)	clk2	hold	scen2	0.14	
ffa/D	CP(rise)	clk3	hold	scen1	0.15	
ffb/D	CP(rise)	clk3	hold	scen1	0.15	
ffc/D	CP(rise)	clk3	hold	scen1	0.15	
ffc/D	CP(rise)	CLK	hold	scen1	0.15	
ffd/D	CP(rise)	clk3	hold	scen1	0.15	
ffd/CR	CP(rise)	CLK	setup	scen3	9.10	
ffd/CR	CP(rise)	clk2	setup	scen3	9.15	
ffc/D	CP(rise)	clk2	hold	scen2	9.40	

report_clock_timing

The `report_clock_timing` command executed by the master process returns a merged report that displays the timing attributes of clock networks in a design. The output for the merged `report_clock_timing` report is similar to the single-core analysis `report_clock_timing` report; however, the merged report contains a list of scenarios, which are displayed under the Scen column. [Example 3-6](#) shows an example of the output of the merged `report_clock_timing` command.

Example 3-6 Merged report_clock_timing Report

```
pt_shell> report_clock_timing -type latency -nworst 3
*****
Report : clock timing
          -type latency
          -launch
          -nworst 3
          -setup
Design   : multi-scenario design
...
*****
Scenarios: scen1, scen2
Clock: CLKA
          --- Latency ---
Clock Pin      Scen   Trans   Source   Network   Total
-----
1_1/CP         scen1   0.00    0.11    25.36    25.47    rp-+
f1_3/CP        scen2   0.00    0.11    23.96    24.07    rp-+
gclk_ff/CP     scen1   0.00    0.10    22.96    23.06    rpi-+
```

report_constraint

The `report_constraint` command executed by the master process returns a merged report that displays constraint-based information for all scenarios in command focus. It reports whether or not a constraint is violated or met, by how much that constraint is violated, and which design object is the worst violator.

The merging process for the `report_constraint` command for DMSA treats all the scenarios in command focus as if they are a single virtual PrimeTime instance. When an object is constrained in more than one scenario, these constraints are considered duplicates. PrimeTime reports the worst violating instance of each relevant object. For example, for a `max_capacitance` report, PrimeTime reports on the most critical instance for each pin. When there are multiple instances of the same pin across multiple scenarios, PrimeTime retains the worst violator and uses the scenario name as a tie-breaker to ensure determinism. When you specify the `-verbose` or `-all_violators` option, PrimeTime reports the scenario name for the constraint violators.

Collection handling using the `report_constraint` command occurs in the same way as the `report_timing` command. For an example of the `report_timing` command collection handling, see [Example 3-10](#).

Note:

There are many different types of constraints and reporting styles available when using the `report_constraint` command. For more information about the differences between single-core and multi-scenario reports, see [Single and Multiple Scenario Constraint Reports](#).

[Example 3-7](#) shows the constraint report output for the merged DMSA constraint report.

Example 3-7 PrimeTime DMSA Merged Constraint Report

```
pt_shell> report_constraint -all_violators -path_type slack_only
*****
Report : constraint
          -all_violators
          -path_type slack_only
Design  : multi-scenario design
...
*****
max_delay/setup ('default' group)

Endpoint           Scenario      Slack
-----             -----
QA                  scen1       -1.80  (VIOLATED)
QB                  scen1       -1.79  (VIOLATED)

max_delay/setup ('CLK' group)
```

Endpoint	Scenario	Slack
<hr/>		
ffd/D	scen1	-4.12 (VIOLATED)
ffc/D	scen2	-4.01 (VIOLATED)
ffb/D	scen1	-3.73 (VIOLATED)
ffa/D	scen2	-3.60 (VIOLATED)
min_delay/hold ('CLK' group)		
Endpoint	Scenario	Slack
<hr/>		
ffd/CR	scen1	-0.40 (VIOLATED)

If you specify the `report_constraint` command in summary mode for DMSA when handling setup checks, the report shows the worst setup constraint for all scenarios per group. The hold information displays the sum of the worst total endpoint cost per clock group over all scenarios. [Example 3-7](#) shows the summary output for the merged DMSA constraint report shown in [Example 3-8](#).

Example 3-8 PrimeTime DMSA Merged Summary Report

```
pt_shell> report_constraint
*****
Report : constraint
Design : multi_scenario design
...
*****

```

Group (max_delay/setup)	Cost	Weight	Weighted Cost
default	1.80	-	1.80
CLK	4.12	-	4.12
max_delay/setup			5.92

Group (min_delay/hold)	Cost	Weight	Weighted Cost
CLK	0.40	-	0.40
min_delay/hold			0.40

Constraint	Cost
max_delay/setup	5.92 (VIOLATED)
min_delay/hold	0.40 (VIOLATED)

[Example 3-9](#) shows the output of an all-violators constraint report with the `max_capacitance` option for a multi-scenario analysis:

Example 3-9 PrimeTime DMSA Merged Constraint Report

```
pt_shell> report_constraint -all_violators -path_type slack_only \
    -max_capacitance
*****
Report : constraint
    -all_violators
    -path_type slack_only
    -max_capacitance
Design : multi_scenario design
...
*****
Scenarios: scen1 scen2
max_capacitance
Pin      Scenario   Capacitance Required Capacitance Actual Slack
-----  

RESET    scen2      0.40        7.00      -6.60 (VIOLATED)
ffa/QN   scen2      0.40        6.00      -5.60 (VIOLATED)
ffb/QN   scen1      0.50        5.00      -4.50 (VIOLATED)
ffc/QN   scen1      0.50        4.00      -3.50 (VIOLATED)
o/Z      scen1      0.50        4.00      -3.50 (VIOLATED)
p/Z      scen1      0.50        4.00      -3.50 (VIOLATED)
```

report_si_bottleneck

To locate nets that are most critical in crosstalk delay, use the `report_si_bottleneck` command. With it, you need minimal net repair effort to identify and fix most problems. Across all scenarios, sorting and printing is performed with duplicates pruned to leave a unique set of nets across the scenarios.

The following example executes the `report_si_bottleneck` command at the master.

```
pt_shell> report_si_bottleneck \
    -cost_type total_victim_delay_bump -max \
    -slack_lesser_than 2 -significant_digits 6
*****
Report : si_bottleneck
    -cost_type total_victim_delay_bump
    -slack_lesser_than 2
    -max_nets 20
    -significant_digits 6
    -minimum_active_aggressors 1
    -max
...
*****
```

Bottleneck Cost: total_victim_delay_bump

Net	Scenario Name	Cost
<hr/>		
OutFirstReg	s1	0.919074
Reg	s2	0.639971
PreNet	s1	0.367737
Comb	s4	0.021904
InReg	s3	0.000039
InComb	s2	0.000033

report_timing

The `report_timing` command executed by the master process returns a merged report that eliminates redundant data across scenarios and sorts the data in order of slack, effectively treating the scenarios in the command focus as a single analysis. The merging process allows you to treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. To do this, PrimeTime reports the worst paths from all scenarios while maintaining the limits imposed by the `-nworst`, `-max_paths`, and other options of the `report_timing` command.

When the same path is reported from multiple scenarios, PrimeTime keeps only the most critical instance of that path in the merged report and shows the scenario in which that instance of the path was the most critical. This way, the resulting report is more evenly spread across the design instead of being focused on one portion of the design that is critical in all scenarios.

Note:

You can disable this capability with the `-dont_merge_duplicates` option of the `report_timing` command. For more information, see [Handling Duplicate Paths](#).

PrimeTime considers two paths from two scenarios to be instances of the same path if the two paths meet all of the following criteria:

- Path group
- Sequence of pins along the data portion of the path
- Transitions at every pin along the data portion of the path
- Launch clock
- Capture clock
- Constraint type

The following example shows the merged `report_timing` command that was issued at the master and a multi-scenario analysis run with two scenarios, `func_bc` and `func_wc`:

```
pt_shell> report_timing -delay_type min -nworst 2 \
    -max_paths 2 -group mrx_clk_pad_I
```

```

Start of Master/Slave Task Processing
-----
Started    : Command execution in scenario 'func_wc'
Started    : Command execution in scenario 'func_bc'
Succeeded  : Command execution in scenario 'func_bc'
Succeeded  : Command execution in scenario 'func_wc'
-----
End of Master/Slave Task Processing

Startpoint: txethmac1/WillTransmit_reg
            (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
Endpoint: WillTransmit_q_reg
            (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
Path Group: mrx_clk_pad_i
Path Type: min
Scenario: func_bc
Min Data Paths Derating Factor : 1.00
Min Clock Paths Derating Factor : 1.00
Max Clock Paths Derating Factor : 1.05

Point IncrPath
-----
clock mrx_clk_pad_i (rise edge)          0.00      0.00
library hold time -                      0.06      0.67
data required time                       0.67
-----
data required time                       0.67
data arrival time                        -0.76
-----
slack (MET)                            0.10

Startpoint: txethmac1/WillTransmit_reg
            (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
Endpoint: WillTransmit_q_reg
            (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
Path Group: mrx_clk_pad_i
Path Type: min
Scenario: func_wc
Min Data Paths Derating Factor : 0.95
Min Clock Paths Derating Factor : 0.95
Max Clock Paths Derating Factor : 1.00
Point IncrPath
-----
clock mrx_clk_pad_i (rise edge)          0.00      0.00
clock network delay (propagated)        1.0       51.05
txethmac1/WillTransmit_reg/CP (_SDFCNQD1) 0.00      1.05 r
txethmac1/WillTransmit_reg/Q (_SDFCNQD1) 0.44      1.49 f
txethmac1/WillTransmit (eth_txethmac_test_1) 0.00      1.49 f
WillTransmit_q_reg/D (_SDFQD1)           0.00      1.49 f
data arrival time1.49

clock mrx_clk_pad_i (rise edge)          0.00      0.00
clock network delay (propagated)        1.31      1.31

```

```

clock reconvergence pessimism          0.00      1.31
WillTransmit_q_reg/CP (_SDFQD1)       1.31 r
library hold time                     0.06      1.37
data required time                   1.37
-----
data required time                   1.37
data arrival time                    -1.49
-----
slack (MET)                         0.12

```

For information about how to use the `report_timing` options to control the output reports, see

- [Standard Option Handling](#)
- [Complex Option Handling](#)
- [Handling Duplicate Paths](#)

Standard Option Handling

All of the options of the `report_timing` command are available for merged reporting, with a few exceptions (see [Limitations](#)). Provided collections are not being passed to the options, they can be specified at the master just as in a single scenario session of PrimeTime. For example,

```

pt_shell> report_timing -nworst 10
pt_shell> report_timing -nworst 10 -max_paths 15
pt_shell> report_timing -delay_type max_rise \
           -path_type full_clock -nosplit
pt_shell> report_timing -from UI/CP -rise_to U2/D \
           -significant_digits 5

```

As in the `remote_execute` command, to evaluate subexpressions and variables remotely, generate the options using curly braces. For example,

```
report_timing -from {$all_in} -to {[all_outputs]}
```

In this example, the `report_timing` command is a merged reporting command that collates data from the slave and generates a merged report at the master. The `all_in` variable and the `all_outputs` expression are evaluated in a slave context.

To evaluate expressions and variables locally at the master, enclose the command string in quotation marks. All master evaluations must return a string. For example,

```
report_timing -to "$all_out"
```

Use the `-pre_commands` option so the collection is generated in the same task as the `report_timing` command is executed. The merged `report_timing` command at the

master then refers to the explicit collection using the name you specified. See [Example 3-10](#).

Example 3-10 Example of Explicit and Implicit Collection

```
pt_shell> report_timing
           -pre_commands {set start_pins [get_pins U1/*]}
           -from {$start_pins}
```

The implicit collection is referred to in the merged `report_timing` command at the master using curly braces around the slave expression. At the slave, the implicit collection is generated and passed to the `-from` option of the `report_timing` command. For example,

```
pt_shell> report_timing -from {[get_pins U1/A]}
```

Complex Option Handling

To allow for evaluating master and slaves variables and expressions from the multi-scenario master, the following options have been altered at the multi-scenario master:

-from	-rise_from	-fall_from
-through	-rise_through	-fall_through
-to	-rise_to	-fall_to
-exclude	-rise_exclude	-fall_exclude

In a single scenario session of PrimeTime, these variables accept a list as an argument; however, in multi-scenario merged reporting, these variables accept a string as an argument. For example,

```
# Single scenario session of PrimeTime
report_timing -from {ffa ffb} # in this case, the argument
                           # to -from is a list.

# Multi-scenario merged reporting
report_timing -from {ffa ffb} # in this case, the argument
                           # to -from is treated as a string.
```

Handling Duplicate Paths

The `-dont_merge_duplicates` option is available with the `report_timing` command only if you invoke PrimeTime with the `-multi_scenario` option. This option affects the way in which paths from multiple scenarios are merged. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. When you use this option, PrimeTime does not merge duplicate instances of the same path into a single instance; however, it shows all critical instances of the path from all scenarios.

Note:

The number of paths reported is limited by the `-nworst` and `-max_paths` options and other options of the `report_timing` command. Therefore, when you use the `-dont_merge_duplicates` option, the resulting merged report might not be evenly

spread out across the design. The report might be focused on the portion of the design that is critical in each scenario.

Saving and Restoring Your Session

You can find the images generated by the master-issued `save_session` command in the `$sh_launch_dir/images/scen1` and `$sh_launch_dir/images/scen2` directories. You can then reload these images into a single scenario PrimeTime session.

The save and restore features allow you to explicitly restore a saved image into a DMSA scenario. This is achieved with the addition of the `create_scenario -image` option. By using the `-image` option, you can load any single-core analysis PrimeTime image that has been produced by the `save_session` command (either inside or outside of DMSA). Load this image into DMSA directly as a scenario. You can then operate upon it in the normal way within the context of DMSA.

License Resource Management

By default, the master takes care of its own licensing regardless of the licenses required to perform multi-scenario analysis. After the master is launched, it acquires any needed PrimeTime licenses from the license pool in the same way as a single scenario PrimeTime session. The master then dynamically distributes the licenses to the slave processes. To specify the maximum number of licenses of a particular feature that the master can check out from the license pool for multi-scenario processing, use the `set_multi_scenario_license_limit` command. Any licenses that the master checks out are available for slave usage. For instance, five PrimeTime licenses allow one master and five slaves to operate concurrently.

Setting the limit on the number of licenses to use does not cause PrimeTime to check out any licenses from the central pool. The licenses are checked out after the slaves begin processing tasks. To specify that the licenses are checked out immediately, use the `-force` option. In the following example, the limit on the number of PrimeTime licenses to use is set to five. The master process checks out no more than four additional licenses (but might check out fewer depending on the availability in the main licensing pool) making up to a maximum of five licenses available for slave usage:

```
pt_shell> set_multi_scenario_license_limit -feature PrimeTime 5
```

If any analysis being performed involves crosstalk analysis, you must set the limit on the number of PrimeTime SI licenses to use during multi-scenario analysis. If the limit is not set before the analysis begins, the analysis proceeds just as in PrimeTime, without the crosstalk features. For example, the following command sets the limit on the number of PrimeTime SI licenses for slave usage to six. Since the master has not checked out a PrimeTime SI

license, it checks out up to six PrimeTime SI licenses for slave usage during multi-scenario analysis:

```
pt_shell> set_multi_scenario_license_limit -feature PrimeTime-SI 6
```

Since the master dynamically allocates licenses to the slave processes on an as-needed basis, a slave process uses a license only when it is actively executing a task. This is important in the context of maximizing the overall performance of the multi-scenario mechanism. Therefore, it is possible to have more slave processes added than licenses available for them to use. The master distributes tasks across all of the slave processes, and as each begins to execute a task, it checks out the appropriate licenses from the master for the duration of time it takes to execute the task. After the task is complete, the slave process returns the licenses to the master for other slave processes to use.

The benefit of this mechanism is that if you have more slave processes than licenses, you can maximize the multi-scenario system throughput by minimizing the expensive process involved in swapping out a scenario.

For optimal system performance of DMSA, match the license limit to the number of slave processes added. In this way, all slave processes can execute concurrently, thereby maximizing the system throughput.

DMSA provides the following license management capabilities:

- [Incremental License Handling](#) – Lowers the license limit; any licenses over the limit are returned to the license server pool.
- [License Pooling](#) – Checks out licenses for different features from different license servers.
- [License Autoreduction](#) – Instructs the master to automatically reduce the number of licenses for features checked out to a minimum when those licenses are not needed.
- [License Queuing](#) – Waits for licenses to become available if all of them are currently in use.

Incremental License Handling

By default, the master does not allow you to reduce the licensing limit for a feature. With this configuration, when the master checks out licenses of a feature, they are not returned to the central license server until the master exits. You can configure the master to enable license reduction by enabling incremental license handling, which is not enabled by default. To enable incremental license handling, set the following environment variable:

```
% setenv SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING 1
```

If you do not set this environment variable or set it to anything other than 1, the master is not able to check licenses back into the central license server until it exits. Changing the environment variable from within the master session has no effect.

When you enable incremental license handling, you can use the `set_multi_scenario_license_limit` command to both increase and decrease the license upper limit of a feature. When the upper limit is being lowered, the upper limit is immediately reduced and the master checks in sufficient licenses of the feature to the central license server, if needed, to meet the new target limit. If no licenses have been checked out from the central license server or the number checked out is less than the new limit, the command lowers the upper limit.

Note:

There must always be a minimum of 1 license checked out for all features in use at all times. Therefore, you cannot use the `set_multi_scenario_license_limit` command to reduce the upper limit to 0.

License Pooling

License pooling provides the ability to check out licenses of different feature from different license servers. You can activate license pooling by enabling incremental license handling. Before the master is launched, set the

`SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING` environment variable to 1. This variable also enables incremental license check in, which allows PrimeTime to automatically return unused licenses during the timing update. Setting the `SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING` environment variable to anything other than 1 disables incremental license handling.

Note:

All licenses of a single feature must be provided by a single license server. The ability to check out licenses for the same feature from multiple license servers, which is called license spanning, is currently unsupported.

License Autoreduction

License autoreduction instructs the master to automatically reduce the number of licenses for features checked out to a minimum level when those licenses are not being used to perform work. You can enable or disable license autoreduction at any time from within the master by setting the `enable_license_auto_reduction` global variable to `true` to enable or `false` to disable. When autoreduction is enabled, if a scenario task is completed and no additional scenarios are waiting for the license, the license is returned to the central license pool.

It is important to use this feature carefully because although it allows licenses to return to the central license server, there is no guarantee that subsequent analysis can reacquire those licenses for further use. If the master cannot reacquire the licenses or can only partially reacquire them, the level of concurrency in the DMSA is reduced. This can affect performance and time-to-results. Also, the time needed to repeatedly check in and check out the licenses themselves can significantly slow down interactive work, such as merged reporting or the execution of Tcl scripts or procedures that communicate heavily with the

scenarios. In these cases, it might be desirable to enable autoreduction during the initial timing update and then disable it during interactive work as follows:

```
# get all scenarios to update their timing
set enable_license_auto_reduction true
remote_execute {update_timing}

# generate merged reports
set enable_license_auto_reduction false
report_timing ...
report_qor ;# Tcl proc from SolvNet (Doc ID 008602)
dmsa_fix_hold ... ;# Tcl proc from SolvNet (Doc ID 018510)
```

License Queuing

If all licenses are currently in use, you can wait for available licenses by enabling license queuing. To do this, set the `SNPSLMD_QUEUE` environment variable to `true` in the user environment.

If you enable license queuing, also specify the maximum queue wait time (in seconds) by setting the `SNPS_MAX_QUEUEETIME` environment variable. In DMSA, you should set this variable to a low value.

When queuing for licenses, the master process is blocked until the requested license is available or when the maximum queue time expires. In the meantime, the slave processes with available licenses can proceed; when these slave processes finish, the master reassigns the available licenses to other slaves.

Database Support

File reading and support at the slave is supported without restriction. This means that support for the `read_ddc` and `read_milkyway` commands in DMSA is identical to the support for these commands in PrimeTime.

Controlling Fault Handling

In DMSA, PrimeTime supports fault handling, which you can control by setting the `multi_scenario_fault_handling` variable to either `ignore` or `exit`. When the variable is set to `ignore` (the default) and a critical fault occurs in a slave process, the abnormally terminated scenarios and the scenarios that depend on those that terminated are excluded from further analysis within the context of the current session. A warning message is

displayed showing which scenarios abnormally terminated. The session then proceeds to analyze the remaining scenarios that are in command focus. The following situations might occur:

- If all scenarios in command focus of the current session abnormally terminate, any subsequent distributed command issued at the master fails, and an error message explains that there are no scenarios in command focus. You need to change the command focus within the context of the current session and analyze the selected scenarios.
- If all scenarios in the current session abnormally terminate, the current session terminates and you receive a warning saying the session is removed. Any subsequent distributed commands issued at the master causes an error message explaining that there is no current session.
- Critical faults cause resources to go offline. If the resources available are reduced to the point where none are online, the session must terminate and you receive an error message explaining that the current session has terminated.

The `exit` option of the `multi_scenario_fault_handling` variable is available for backward compatibility. When you select this option, if a critical fault occurs in a slave process, the active command completes its task and then the master process terminates the entire analysis. The process exits gracefully and an information message is displayed explaining what occurred.

Merged Reporting Commands

Merged reporting is a mechanism that works with all the scenarios in command focus to eliminate redundant data automatically across scenarios and sort the data in order of criticality. This allows you to treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. Due to the distributed feature, if one or more slave processes terminate abnormally, the relevant information for the reporting command is not available. Therefore, when a scenario terminates abnormally, you receive a warning message, and PrimeTime removes this scenario from the current session. PrimeTime then automatically reissues the reporting command to the remaining scenarios in command focus. Any reporting command issued afterwards is applied only to the remaining scenarios in command focus and any abnormally terminated scenarios are excluded from further analysis.

Netlist Editing Commands

Before a netlist change is committed to any scenario from the master, it must be possible to commit the change to all scenarios. Thus, PrimeTime first checks that the netlist change could be successful in all scenarios and next it commits the change. If a scenario terminates abnormally while PrimeTime is checking the possibility of successfully executing the netlist change, this scenario is removed from the command focus, and the netlist editing command

is automatically reissued to the remaining scenarios in command focus. If all of the checking processes succeeded in all scenarios, but there was a scenario that terminated abnormally when PrimeTime went to commit the change, a warning message is issued, and all the remaining scenarios in command focus apply the netlist change. Subsequent netlist editing commands are applied only to the remaining scenarios in command focus and any abnormally terminated scenarios are excluded from further analysis.

Using the `remote_execute` Command

DMSA supports execution of any number of commands on all scenarios in command focus by using the `remote_execute` command. If the `remote_execute` command encounters a scenario that terminated abnormally, a warning message is issued, the scenario is removed from the current session, and subsequent calls to the command apply only to the remaining scenarios in command focus. Abnormal termination of one or more scenarios has no effect on command execution in other scenarios.

Other Commands

The following commands do not come under the categories that have been listed, but they are handled in a similar way when an abnormal termination occurs.

- `get_distributed_variables` command

When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `get_timing_paths` command

When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `save_session` command

When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `set_distributed_variables` command

When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. The command executes as normal for the remaining scenarios in command focus.

Messages and Log Files

Multi-scenario analysis has the potential to generate large amounts of data. To avoid overloading you with data at the master, all data is written to a set of log files. A limited set of interactive messages are displayed at the master.

If you use the `-verbose` option, the `remote_execute` command sends all data to the master terminal.

To learn about the types of messages and log files available in DMSA, see

- [Interactive Messages](#)
- [Progress Messages](#)
- [User Control of Task Execution Status Messages](#)
- [Error Messages](#)
- [Warning Messages](#)
- [Log Files](#)
- [Command Output Redirection](#)

Interactive Messages

While a task is executing on the slaves, the slaves send messages back to the master to indicate their progress.

Progress Messages

As the slave progresses through each stage of the analysis, the slaves send confidence messages to update you on the progress of the analysis. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-----
Started   : Netlist image generation for session 'session'
Succeeded : Netlist image generation for session 'session'
Started   : Baseline image generation for scenario 'scen1'
Succeeded : Baseline image generation for scenario 'scen1'
```

User Control of Task Execution Status Messages

You can control the verbosity of DMSA task execution status messages. You can set the `multi_scenario_message_verbosity_level` variable to one of two values, default and

low. When you set to the default, all task execution status messages are displayed. For example,

```
Start of Master/Slave Task Processing
-----
Started    : Command execution in scenario 'wc'
Started    : Command execution in scenario 'bc'
Succeeded  : Command execution in scenario 'bc'
Succeeded  : Command execution in scenario 'wc'
-----
End of Master/Slave Task Processing
```

When you set the variable to low, only error, fatal, and failure messages are displayed. You might find this useful for Tcl scripts where more control over the output is desired. If this variable is changed by a Tcl script, it is desirable to change the variable back to its original value after completing the script or procedure to avoid confusion.

Error Messages

When an error occurs on a slave for a given scenario, the error is reported in full at the master together with the name of that scenario. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-----
Started    : Netlist image generation for session 'session'
Error      : Problem in read_verilog. No designs were read
(DBR-011)  (default_session)
```

Warning Messages

When a warning occurs on a slave for a given scenario, the warning is reported at the master only if it is the first time that this warning has occurred in that scenario. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-----
Started    : Netlist image generation for session 'session'
Succeeded  : Netlist image generation for session 'session'
Started    : Baseline image generation for scenario 'scen1'
Warning    : RC-009 has occurred on scenario 'scen1'
```

Log Files

You can examine all information generated by the slaves in the log, output log, command log, or merged error log file. The root working directory for all multi-scenario analysis data, including log files, is determined by the setting of the `multi_scenario_working_directory` variable. If you do not explicitly set this variable, it is set to the current directory from which the PrimeTime master was invoked. You must have write permission to the working directory, and the directory must be accessible by the master and to all slaves.

Output Log

Each scenario has its own output log file, which is located in the scenario working directory. For example, for a scenario named `s1`, the log is the following:

```
multi_scenario_working_directory/s1/out.log
```

You cannot set or change the name of the output log. The output log for a particular scenario contains all errors, warnings, and information messages, which is all of the information that you normally see in the terminal window of a conventional single analysis PrimeTime run.

Command Log

Each remote process has its own command log file that you can find in the `command_log` directory under the working directory. For example, for a process launched on a host named `srv1` that is the sixth process to be launched, the command log would be in the following file:

```
multi_scenario_working_directory/command_log/srv1_6_pt_shell_command.log
```

You cannot set or change the name of the command log of remote processes. The command log for a particular process contains all commands executed on that process. You can locate the masters command log in the following file:

```
$sh_launch_dir/pt_shell_command.log
```

The `sh_source_logging` variable, when set to `true`, causes individual commands from sourced scripts to be written to the command log file.

Merged Error Log

Large numbers of errors, warnings, and information messages can occur during a multi-scenario analysis. To help you debug these messages, you can set the merged error log variable at the master. For example, the following command creates a merged error file in `multi_scenario_working_directory/error.log`:

```
pt_shell> set_app_var multi_scenario_merged_error_log "error.log"
```

When this variable is set, all error messages from the start of a particular task to the end of the task are merged together. If the same error message occurs on a number of scenarios,

the message is printed to the merged error file only one time, with a list of the scenarios in which the message occurred.

Note:

You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before issuing the `start_hosts` command.

Command Output Redirection

Command output redirection is supported in multi-scenario analysis. Be sure to redirect output to different destination files for each process. Otherwise, an error occurs when two different processes attempt to control the same file at the same time.

Note:

When using output redirection, be sure use a relative path. Do not use an absolute path; if you do, all slave processes attempt to write to the same file, which could lead to read/write conflicts and cause the slave processes to fail.

As with the previous log file example, the value of the `multi_scenario_working_directory` variable is used along with the scenario name to determine the output destination file name. The following examples demonstrate the file naming conventions:

Example 1: Interactive redirection

```
pt_shell> set x myfile  
pt_shell> remote_execute {report_timing} > $x.out
```

The output of the `remote_execute` command is directed to a file named `myfile.out` in the current working directory. The `$x` variable is evaluated only at the master process.

Example 2: Script-based redirection

The `multi_scenario_working_directory` variable has been set to `/rpt/new`, and there are two scenarios in command focus, `s3` and `s4`. The master issues the `remote_execute { "source myscript.tcl" }` command, which sources the `myscript.tcl` Tcl script in the scenarios in command focus. The `myscript.tcl` script contains the following line:

```
report_timing > rt.log
```

The output of the `report_timing` command goes into the following files:

```
/rpt/new/s3/rt.log  
/rpt/new/s4/rt.log
```

Limitations

The DMSA feature has the following known functional limitations and issues:

- Maximum number of slave processes

Currently, the maximum number of slave processes that the master process can control is 256 concurrent hosts.

- GUI is unsupported

Single and Multiple Scenario Constraint Reports

To learn about the differences in the reports generated by the `report_constraint` command for single and multiple scenario analysis, see

- [DMSA Variables](#)
- [Printing Styles](#)
- [Reporting Modes](#)

Constraint Merging Operations

There are four fundamental operations performed to gather constraint information for merging reports. [Table 3-1](#) describes how these operations are currently defined in single-core analysis mode and how they are supported in the multi-scenario mode of PrimeTime.

Table 3-1 Constraint Merging Operation

Name	Single scenario analysis	Distributed multi-scenario analysis
Constraint summation	Adds all weighted constraint costs of a particular constraint together.	Adds the worst constraint cost at each violating endpoint, pin, and net over all scenarios in command focus. When multiple scenarios have the same violating endpoints, pins, or nets, use the worst slack of these scenarios.
Worst constraint per group	Finds the worst cost per path group of a particular constraint.	Finds the worst cost per path group of a particular constraint across all scenarios. Groups are considered duplicate if they have the same name. If there is a tie between scenarios, the scenario name is used as a tie-breaker

Table 3-1 Constraint Merging Operation (Continued)

Name	Single scenario analysis	Distributed multi-scenario analysis
Worst constraint	Finds the worst cost of a particular constraint.	Finds the worst cost of a particular constraint across all scenarios. If there is a tie between scenario names, group names, or both, the group name and then the scenario name is used as tie-breakers.
Sort constraints	Sorts all violators of a particular constraint from best to worst.	Sorts all violators of a particular constraint across all scenarios from best to worse. For each constraint type remove any duplicate constraints. A constraint is considered duplicate if the same constraint is specified on the same pin. Two pins are considered duplicate if they have the same leaf name and hierarchy. For example, a/b/c, d, a/b, c/d are different. If there is a tie between constraints, the scenario name is used as the tie-breaker

Printing Styles

There are a number of printing styles you can specify to write out an individual constraint report. [Table 3-2](#) describes how these printing styles are currently defined in the single-core analysis mode and how they are defined in the multi-scenario mode of PrimeTime.

Table 3-2 Printing Styles

Printing style	Single scenario analysis	Distributed multi-scenario analysis																		
Group summary	<table> <thead> <tr> <th>Group (max_delay/setup)</th> <th>Cost</th> <th>Weight</th> <th>Weighted Cost</th> </tr> </thead> <tbody> <tr> <td>CLK</td> <td>0.00</td> <td>1.00</td> <td>0.00</td> </tr> </tbody> </table>	Group (max_delay/setup)	Cost	Weight	Weighted Cost	CLK	0.00	1.00	0.00	<table> <thead> <tr> <th>Group (max_delay/setup)</th> <th>Cost</th> <th>Weight</th> <th>Weighted Cost</th> </tr> </thead> <tbody> <tr> <td>CLK</td> <td>0.00</td> <td>1.00</td> <td>0.00</td> </tr> </tbody> </table>	Group (max_delay/setup)	Cost	Weight	Weighted Cost	CLK	0.00	1.00	0.00		
Group (max_delay/setup)	Cost	Weight	Weighted Cost																	
CLK	0.00	1.00	0.00																	
Group (max_delay/setup)	Cost	Weight	Weighted Cost																	
CLK	0.00	1.00	0.00																	
Constraint summary	<table> <thead> <tr> <th>Constraint</th> <th>Cost</th> </tr> </thead> <tbody> <tr> <td>max_delay/setup</td> <td>2.46 (VIOLATED)</td> </tr> </tbody> </table>	Constraint	Cost	max_delay/setup	2.46 (VIOLATED)	<table> <thead> <tr> <th>Constraint</th> <th>Cost</th> </tr> </thead> <tbody> <tr> <td>max_delay/setup</td> <td>2.46 (VIOLATED)</td> </tr> </tbody> </table>	Constraint	Cost	max_delay/setup	2.46 (VIOLATED)										
Constraint	Cost																			
max_delay/setup	2.46 (VIOLATED)																			
Constraint	Cost																			
max_delay/setup	2.46 (VIOLATED)																			
Object summary	<table> <thead> <tr> <th>Pin</th> <th>pulse width</th> <th>pulse width</th> <th>Slack</th> </tr> </thead> <tbody> <tr> <td>nand1/Z</td> <td>10.00</td> <td>9.58</td> <td>-0</td> </tr> </tbody> </table>	Pin	pulse width	pulse width	Slack	nand1/Z	10.00	9.58	-0	<table> <thead> <tr> <th>Pin</th> <th>Scenario</th> <th>pulse width</th> <th>pulse width</th> <th>Slack</th> </tr> </thead> <tbody> <tr> <td>nand1/Z</td> <td>scen1</td> <td>10.00</td> <td>9.58</td> <td>-0</td> </tr> </tbody> </table>	Pin	Scenario	pulse width	pulse width	Slack	nand1/Z	scen1	10.00	9.58	-0
Pin	pulse width	pulse width	Slack																	
nand1/Z	10.00	9.58	-0																	
Pin	Scenario	pulse width	pulse width	Slack																
nand1/Z	scen1	10.00	9.58	-0																

Table 3-2 Printing Styles (Continued)

Printing style	Single scenario analysis	Distributed multi-scenario analysis																																																																																																
Path verbose	<p>Startpoint: c (input port) Endpoint: z3 (output port) Path Group: default Path Type: max</p> <table> <thead> <tr> <th>Point</th> <th>Incr</th> <th>Path</th> </tr> </thead> <tbody> <tr> <td>input external delay</td> <td>0.00</td> <td>0.00 r</td> </tr> <tr> <td>c (in)</td> <td>0.00</td> <td>0.00 r</td> </tr> <tr> <td>U6/Z (IV)</td> <td>1.34</td> <td>1.34 f</td> </tr> <tr> <td>U2/Z (NR2)</td> <td>3.35</td> <td>4.69 r</td> </tr> <tr> <td>U15/Z (AO7)</td> <td>0.87</td> <td>5.56 f</td> </tr> <tr> <td>U24/Z (AO3)</td> <td>1.02</td> <td>6.57 r</td> </tr> <tr> <td>z3 (out)</td> <td>0.00</td> <td>6.57 r</td> </tr> <tr> <td>data arrival time</td> <td></td> <td>6.57</td> </tr> <tr> <td>max_delay</td> <td>6.50</td> <td>6.50</td> </tr> <tr> <td>output external delay</td> <td>0.00</td> <td>6.50</td> </tr> <tr> <td>data required time</td> <td>6.50</td> <td></td> </tr> <tr> <td></td> <td></td> <td>-----</td> </tr> <tr> <td>data required time</td> <td></td> <td>6.50</td> </tr> <tr> <td>data arrival time</td> <td></td> <td>-6.57</td> </tr> <tr> <td>slack (VIOLATED)</td> <td></td> <td>-0.07</td> </tr> </tbody> </table>	Point	Incr	Path	input external delay	0.00	0.00 r	c (in)	0.00	0.00 r	U6/Z (IV)	1.34	1.34 f	U2/Z (NR2)	3.35	4.69 r	U15/Z (AO7)	0.87	5.56 f	U24/Z (AO3)	1.02	6.57 r	z3 (out)	0.00	6.57 r	data arrival time		6.57	max_delay	6.50	6.50	output external delay	0.00	6.50	data required time	6.50				-----	data required time		6.50	data arrival time		-6.57	slack (VIOLATED)		-0.07	<p>Startpoint: c (input port) Endpoint: z3 (output port) Path Group: default Scenario: scen1 Path Type: max</p> <table> <thead> <tr> <th>Point</th> <th>Incr</th> <th>Path</th> </tr> </thead> <tbody> <tr> <td>input external delay</td> <td>0.00</td> <td>0.00 r</td> </tr> <tr> <td>c (in)</td> <td>0.00</td> <td>0.00 r</td> </tr> <tr> <td>U6/Z (IV)</td> <td>1.34</td> <td>1.34 f</td> </tr> <tr> <td>U2/Z (NR2)</td> <td>3.35</td> <td>4.69 r</td> </tr> <tr> <td>U15/Z (AO7)</td> <td>0.87</td> <td>5.56 f</td> </tr> <tr> <td>U24/Z (AO3)</td> <td>1.02</td> <td>6.57 r</td> </tr> <tr> <td>z3 (out)</td> <td>0.00</td> <td>6.57 r</td> </tr> <tr> <td>data arrival time</td> <td></td> <td>6.57</td> </tr> <tr> <td>max_delay</td> <td>6.50</td> <td>6.50</td> </tr> <tr> <td>output external delay</td> <td>0.00</td> <td>6.50</td> </tr> <tr> <td>data required time</td> <td>6.50</td> <td></td> </tr> <tr> <td></td> <td></td> <td>-----</td> </tr> <tr> <td>data required time</td> <td></td> <td>6.50</td> </tr> <tr> <td>data arrival time</td> <td></td> <td>-6.57</td> </tr> <tr> <td>slack (VIOLATED)</td> <td></td> <td>-0.07</td> </tr> </tbody> </table>	Point	Incr	Path	input external delay	0.00	0.00 r	c (in)	0.00	0.00 r	U6/Z (IV)	1.34	1.34 f	U2/Z (NR2)	3.35	4.69 r	U15/Z (AO7)	0.87	5.56 f	U24/Z (AO3)	1.02	6.57 r	z3 (out)	0.00	6.57 r	data arrival time		6.57	max_delay	6.50	6.50	output external delay	0.00	6.50	data required time	6.50				-----	data required time		6.50	data arrival time		-6.57	slack (VIOLATED)		-0.07
Point	Incr	Path																																																																																																
input external delay	0.00	0.00 r																																																																																																
c (in)	0.00	0.00 r																																																																																																
U6/Z (IV)	1.34	1.34 f																																																																																																
U2/Z (NR2)	3.35	4.69 r																																																																																																
U15/Z (AO7)	0.87	5.56 f																																																																																																
U24/Z (AO3)	1.02	6.57 r																																																																																																
z3 (out)	0.00	6.57 r																																																																																																
data arrival time		6.57																																																																																																
max_delay	6.50	6.50																																																																																																
output external delay	0.00	6.50																																																																																																
data required time	6.50																																																																																																	

data required time		6.50																																																																																																
data arrival time		-6.57																																																																																																
slack (VIOLATED)		-0.07																																																																																																
Point	Incr	Path																																																																																																
input external delay	0.00	0.00 r																																																																																																
c (in)	0.00	0.00 r																																																																																																
U6/Z (IV)	1.34	1.34 f																																																																																																
U2/Z (NR2)	3.35	4.69 r																																																																																																
U15/Z (AO7)	0.87	5.56 f																																																																																																
U24/Z (AO3)	1.02	6.57 r																																																																																																
z3 (out)	0.00	6.57 r																																																																																																
data arrival time		6.57																																																																																																
max_delay	6.50	6.50																																																																																																
output external delay	0.00	6.50																																																																																																
data required time	6.50																																																																																																	

data required time		6.50																																																																																																
data arrival time		-6.57																																																																																																
slack (VIOLATED)		-0.07																																																																																																
Path slack only summary	<p>min_delay/hold ('CLK' group)</p> <table> <thead> <tr> <th>Endpoint</th> <th>Slack</th> </tr> </thead> <tbody> <tr> <td>ffd/CR</td> <td>-0.40 (VIOLATED)</td> </tr> </tbody> </table>	Endpoint	Slack	ffd/CR	-0.40 (VIOLATED)	<p>min_delay/hold ('CLK' group)</p> <table> <thead> <tr> <th>Endpoint</th> <th>Scenario</th> <th>Slack</th> </tr> </thead> <tbody> <tr> <td>ffd/CR</td> <td>scen1</td> <td>-0.40 (VIOLATED)</td> </tr> </tbody> </table>	Endpoint	Scenario	Slack	ffd/CR	scen1	-0.40 (VIOLATED)																																																																																						
Endpoint	Slack																																																																																																	
ffd/CR	-0.40 (VIOLATED)																																																																																																	
Endpoint	Scenario	Slack																																																																																																
ffd/CR	scen1	-0.40 (VIOLATED)																																																																																																
Object verbose	<p>Net: a</p> <table> <thead> <tr> <th>max_fanout</th> <th>5.00</th> </tr> </thead> <tbody> <tr> <td>- Fanout</td> <td>7.00</td> </tr> <tr> <td>Slack</td> <td>-2.00 (VIOLATED)</td> </tr> </tbody> </table>	max_fanout	5.00	- Fanout	7.00	Slack	-2.00 (VIOLATED)	<p>Net: a Scenario: scen1</p> <table> <thead> <tr> <th>max_fanout</th> <th>5.00</th> </tr> </thead> <tbody> <tr> <td>- Fanout</td> <td>7.00</td> </tr> <tr> <td>Slack</td> <td>-2.00 (VIOLATED)</td> </tr> </tbody> </table>	max_fanout	5.00	- Fanout	7.00	Slack	-2.00 (VIOLATED)																																																																																				
max_fanout	5.00																																																																																																	
- Fanout	7.00																																																																																																	
Slack	-2.00 (VIOLATED)																																																																																																	
max_fanout	5.00																																																																																																	
- Fanout	7.00																																																																																																	
Slack	-2.00 (VIOLATED)																																																																																																	

Reporting Modes

[Table 3-3](#) classifies the constraints into different constraint types. Each constraint type has several reporting modes depending on the `-all_violators`, `-path_type`, and `-verbose` options that you specify. Each reporting mode is associated with a particular merging operation (see [Table 3-6](#)) and printing format (see [Table 3-2](#)).

Table 3-3 Reporting Modes

Constraint type	Constraints	Report mode
Connection class	<code>connection_class</code>	<p>Summary</p> <ul style="list-style-type: none"> • Constraint summation • Constraint summary <p>Verbose</p> <ul style="list-style-type: none"> • Worst constraint • Object verbose <p>All violators</p> <ul style="list-style-type: none"> • Sort constraints • Object summary <p>All violators, verbose</p> <ul style="list-style-type: none"> • Sort constraints • Object verbose
Design rule checking	<ul style="list-style-type: none"> • <code>max_capacitance</code> • <code>min_capacitance</code> • <code>max_transition</code> • <code>min_transition</code> • <code>max_fanout</code> • <code>min_fanout</code> • <code>max_area</code> 	<p>Summary</p> <ul style="list-style-type: none"> • Constraint summation • Constraint summary <p>Verbose</p> <ul style="list-style-type: none"> • Worst constraint • Object verbose <p>All violators</p> <ul style="list-style-type: none"> • Sort constraints • Object summary <p>All violators, verbose</p> <ul style="list-style-type: none"> • Sort constraints • Object verbose

Table 3-3 Reporting Modes (Continued)

Constraint type	Constraints	Report mode
Miscellaneous timing constraints	<ul style="list-style-type: none"> • <code>min_pulse_width</code> • <code>min_period</code> • <code>max_skew</code> • <code>clock_separation</code> 	<p>Summary</p> <ul style="list-style-type: none"> • Constraint summation • Constraint summary <p>Verbose</p> <ul style="list-style-type: none"> • Worst constraint • Object verbose <p>All violators</p> <ul style="list-style-type: none"> • Sort constraints • Object summary <p>All violators, verbose</p> <ul style="list-style-type: none"> • Sort constraints • Path verbose
Timing path	<ul style="list-style-type: none"> • <code>setup</code> • <code>hold</code> • <code>clock_gating_setup</code> • <code>clock_gating_hold</code> • <code>recovery</code> • <code>removal</code> 	<p>Summary</p> <ul style="list-style-type: none"> • Worst constraint per group (<code>clock_gating_setup</code>, <code>setup</code>, and <code>recovery</code>) • Constraint summation (<code>clock_gating_hold</code>, <code>hold</code>, and <code>removal</code>) • Group summary • Constraint summary <p>Verbose</p> <ul style="list-style-type: none"> • Worst constraint per group • Path verbose <p>All violators</p> <ul style="list-style-type: none"> • Sort constraints • Path slack only summary <p>All violators, verbose</p> <ul style="list-style-type: none"> • Sort constraints • Path verbose <p>All violators, <code>path_type_end</code></p> <ul style="list-style-type: none"> • Sort constraints • Object summary

DMSA Commands, Options, and Variables

For lists of commands, options, and variables that you can use with DMSA, see

- [DMSA Command List](#)
- [DMSA report_timing Options](#)
- [DMSA Variables](#)
- [Disallowed Slave Commands](#)

Note:

All regular Tcl commands are supported.

DMSA Command List

[Table 3-4](#) lists the commands that you use throughout the DMSA flow. The DMSA commands are separated into the following categories:

- Configuration and setup commands – Use these commands to create and remove your multi-scenario configuration and scenario.
- Process setup commands – Use these commands to add and remove distributed processes.
- License setup command – Use to set the number of licenses the master can use to control the slave sessions.
- Analysis focus commands – Use to set and remove sessions and scenarios that interpret commands from the master process.
- Slave context commands – Use to create a command buffer from the master to the slave processes. You can run these commands in batch or interactive mode.

Table 3-4 DMSA Commands

Command	Description
Configuration and setup commands	
create_scenario	Creates a single scenario. This command is required.
remove_scenario {scenario_list}	Removes one or more scenarios which are given as a list to the command.
remove_multi_scenario_design	Removes all multi-scenario data specified, including the session, all the scenarios, and the configuration.
report_multi_scenario_design	Reports on the current user-defined multi-scenario analysis.
set_host_options	Define the compute resources. This command is required.
Process setup commands	
remove_host_options	Removes all distributed hosts added to the farm's pool.
report_host_usage	Reports host and resource usage.
start_hosts	Starts the slave processes on the hosts specified by the <code>set_host_options</code> command. This command is required.
License setup command	
set_multi_scenario_license_limit	Limits the number of licenses the master process acquires for slave usage.
Analysis focus commands	
current_scenario	Changes the command focus to the selected scenarios within the current session. Returns a list of all scenarios in command focus.

Table 3-4 DMSA Commands (Continued)

	Command	Description
	current_session	Puts a particular group of scenarios in focus for analysis. The scenarios in focus receive and interpret commands from the master. Returns a list of all scenarios in the current session. This command is required.
	remove_current_session	Removes the previously set session. Defocuses all scenarios. Scenarios stay in memory until removed by the remove_scenario or quit command.
Slave context command		
	remote_execute	Batch or interactive mode: From the master, creates a command buffer to be executed in the slave context.

DMSA report_timing Options

[Table 3-5](#) lists report_timing options for multi-scenario reporting.

Table 3-5 report_timing Options

Option	Description
-dont_merge_duplicates	Turns off the merging of duplicate paths in the timing report. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. By using this option, PrimeTime does not merge duplicate instances of the same path into a single instance, but instead shows all critical instances of the path from all scenarios. Since the number of paths reported is limited by the -nworst, -max_paths, and other options of this command, the resulting merged report, when this option is used, might not be evenly spread out across the design, but instead can be focused on the portion of the design that is critical in each scenario.
-pre_commands	Specifies a string of semicolon-delimited commands to be executed in slave context before the execution of merged reporting. The maximum length of a command is 1000 characters.
-post_commands	Specifies string of semicolon-delimited commands to be executed in slave context before after the execution of merged reporting. The maximum length of a command is 1000 characters.

DMSA Variables

Table 3-6 lists variables for multi-scenario reporting.

Note:

You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before creating the distributed farm.

Table 3-6 Reporting Variables

Variable	Description
<code>multi_scenario_merged_error_limit</code>	Default is 100. Specifies the maximum number of errors of a particular type to be written to the command log on a per-task basis.
<code>multi_scenario_merged_error_log</code>	Default is " ". Note: Specify this variable before issuing the <code>start_hosts</code> command.
<code>multi_scenario_working_directory</code>	Default is <code>cwd</code> . Set the working directory to a directory which can be seen by all slaves and the master and to which you have write permissions. Note: Must be specified before issuing the <code>start_hosts</code> command.
<code>pt_shell_mode</code>	Specifies the working mode: <code>primetime</code> , <code>primetime_master</code> , or <code>primetime_slave</code> . This variable is read-only. You can use this variable in your scripts to determine if you are operating in a normal PrimeTime session or on the master or slave. For example, to set the working directory from only the master, you could use the following: <pre>if {\$pt_shell_mode == "primetime_master"} { set_multi_scenario_working_directory "/home/wd" }</pre>

Disallowed Slave Commands

Some commands are not allowed at certain stages of the flow. For example, if during the analysis, you execute the `remove_design` command either in a script or at the command line, an error condition results. [Table 3-7](#) lists the commands that you cannot run on the slave processes.

Table 3-7 Disallowed Commands for Slave Processes

Command	Description
<code>exit</code>	Exits <code>pt_shell</code>
<code>quit</code>	Quits <code>pt_shell</code>
<code>remove_design</code>	Removes one or more designs from memory
<code>rename_design</code>	Renames a design
<code>restore_session</code>	Restores a <code>pt_shell</code> session created with the <code>save_session</code> command for the same version of PrimeTime

Memory and CPU Resource Usage Reports

The amount of memory and CPU resources that PrimeTime uses to perform static timing analysis depends on several factors such as the size of your design, which analysis mode you are using, and your reporting requirements. PrimeTime provides performance and capacity information for distributed processes. Regardless of whether you are using a single-core analysis, multicore analysis, or distributed multi-scenario analysis flow, you can monitor memory and CPU usage in PrimeTime.

To report the overall processor time (in seconds) associated with the current `pt_shell` process, use the `cputime` command. To report the total memory (in KB) allocated by the current `pt_shell` process, use the `mem` command. When specifying the memory requirements for subsequent PrimeTime sessions, use this reported memory value with an additional 20 percent allowance for efficient operation of the operating system.

To report all the host options set in a distributed multi-scenario analysis (DMSA) flow, use the `report_host_usage` command. This command also reports peak memory, CPU usage, and elapsed time for the local process. In DMSA, it also reports peak memory, CPU usage, and elapsed time for all distributed processes. The report displays the host options specified, status of the distributed processes, number of CPU cores each process uses, and licenses used by the distributed hosts.

For processes that are configured to using multiple CPU cores on the same host machine, such as a single-core or threaded multicore analysis, the `report_host_usage` command

provides the resource usage for the current PrimeTime session. Here is an example of the report output for a single-core or threaded multicore analysis flow.

```
pt_shell> report_host_usage
*****
Report : host_usage
...
*****
Usage limits (cores)

Options Name          #      Effective
-----
(local process)       -        1
-----
Total                 1

Memory usage

Options Name      #      Peak Memory (MB)
-----
(local process)   -        35.88

Performance

Options Name      #      CPU Time (s)      Elapsed Time (s)
-----
(local process)   -        4                  33
```

In DMSA flows, resources can be spread over several hosts. Therefore, the resource usage data is provided for each distributed process. Here is an example of the `report_host_usage` output for a DMSA flow.

```
pt_shell> report_host_usage
*****
Report : host_usage
...
*****
Options Name      Host Name      32Bit      Num Processes
-----
my_opts1          >>farm<<      N          2

Options Name      #      Host Name      Job ID      Process ID      Status
-----
my_opts1          1      ptopt018      136604      1394      ONLINE
                    2      ptopt018      136605      1393      ONLINE
```

Usage limits (cores)

Options Name	#	Effective
local process)	-	1
my_opts1	1	4
	2	4
Total		9

Memory usage

Options Name	#	Peak Memory (MB)
(local process)	-	25.56
my_opts1	1	32.45
	2	32.21

Performance

Options Name	#	CPU Time (s)	Elapsed time (s)
(local process)	-	3	25
my_opts1	1	1	14
	2	2	15

When exiting PrimeTime, the peak memory and CPU usage are reported as well as the elapsed time for the session. For example,

```
Maximum memory usage for this session: 31.08 MB
CPU usage for this session: 23 seconds
Elapsed time for this session: 211 seconds
```

This information remains the same when exiting a single-core or threaded multicore analysis session. When exiting a DMSA session, information about the master and slave or distributed processes are reported. The following example is for a DMSA session:

```
Maximum memory usage for distributed processes:
my_opts1      1  ptopt018          2021.88 MB
my_opts2      2  ptopt018          2022.21 MB
                  3  ptopt018          3056.27 MB
my_opts3      4  >>farm<<        2034.92 MB
                  5  >>farm<<        2120.90 MB

CPU time usage for distributed processes:
my_opts1      1  ptopt018          1562 seconds
my_opts2      2  ptopt018          1578 seconds
                  3  ptopt018          1711 seconds
my_opts3      4  >>farm<<        1592 seconds
                  5  >>farm<<        1621 seconds
```

```
Elapsed time for distributed processes:  
my_opts1      1  ptopt018          1834 seconds  
my_opts2      2  ptopt018          1833 seconds  
                  3  ptopt018          1830 seconds  
my_opts3      4  >>farm<<        1750 seconds  
                  5  >>farm<<        1765 seconds  
  
Maximum memory usage for this session: 4378.52 MB  
CPU usage for this session:    1800 seconds  
Elapsed time for this session: 1980 seconds
```

Performance Profiling of Tcl Scripts

A Tcl profiling capability is available in PrimeTime to help increase productivity. The Tcl profiling capability uses existing commands to identify runtime and memory bottlenecks in your Tcl scripts. Use the information generated by Tcl profiling to fix performance and capacity issues during your timing runs.

To use the Tcl profiling capability,

1. Start Tcl profiling with the `start_profile` command.

PrimeTime starts to collect data about the PrimeTime commands. It generates and stores files in the `/profile` directory in the current working directory. Use the `-output_directory_name` option of the `start_profile` command to specify a different directory location for the reports. If you specify the `-verbose` option, all commands are tracked in the profiling reports, regardless of the runtime or memory used.

2. Stop Tcl profiling with the `stop_profile` command, which stops the collection of information for profiling.

Note:

If the `stop_profile` command is not issued, the profiling output is written out when you use the `exit` or `quit` command in `pt_shell`.

3. Evaluate the automatically generated Tcl profile and stopwatch reports.

After you have issued either the `stop_profile` command or exited the `pt_shell`, PrimeTime writes out several files into the `/profile` subdirectory of the current working

directory. [Table 3-8](#) provides a description of the reports that are automatically generated.

Table 3-8 Tcl Profile and Stopwatch Reports

File name	Description
tcl_profiling_summary_latest.html	Tcl profile report summary page in HTML format. Shows the start and stop times of the profiling and provides links to the report sorted by CPU time, calls, elapsed time, or memory usage.
tcl_profiling_summary_latest.txt	Summary page in plain text that shows the start and stop times of the profiling and paths to the report files.
tcl_stopwatch_6998.txt	Stopwatch report in plain text.
tcl_profile_sorted_by_cpu_6998.txt	Tcl profile report sorted by CPU time.
tcl_profile_sorted_by_calls_6998.txt	Tcl profile report sorted by number of calls.
tcl_profile_sorted_by_elapsed_6998.txt	Tcl profile report sorted by elapsed time.
tcl_profile_sorted_by_memory_6998.txt	Tcl profile report sorted by memory usage.

The Tcl script profile reports show which script called a particular command, how often it was called, how much elapsed time and CPU time was used in total for the command in the same script, and how much memory was used.

[Example 3-11](#) shows an example of the Tcl script profile report that is automatically displayed. The example shows that the `get_timing_paths` command was called one time

from the `gtp_only.tcl` script. It took 8 seconds to execute in total, which is shown in the Elapsed column.

Example 3-11 Tcl Script Profile Report

Procedure/Command	Calls	Elapsed	CPU	Delta Memory
<hr/>				
<total>	1	39	19	268
link_design	2	14	0	0
read_parasitics	1	0	0	0
update_timing	1	9	9	32
source gtp_only.tcl	1	8	7	143
get_timing_paths	1	8	7	143
source gtp_only.tcl				
save_session	1	4	0	18
link_design	1	2	1	30
read_db	1	0	0	44
source write_constraints.pt	1	0	0	0

The Stopwatch report is also automatically displayed, and you can use it to evaluate the commands. The report shows the current time and current measurements of selected metrics and their delta from the last stopwatch event. The stopwatch report is a plain-text file that you can read in a Web browser or any program that displays ASCII text.

For each command executed in the PrimeTime shell between the `start_profile` command and either the `stop_profile` command or exiting the shell, the CPU time, elapsed time, and peak memory usage are provided for the current session. The difference between the metrics for the current and previous stopwatch events are also recorded. If a command takes more than 0.1 second to run or uses more than 100 KB of memory, the start and finish stopwatch details for this command are printed out in the report. If you specify the `-verbose` option of the `start_profile` report, all commands are tracked in the profiling reports.

The first line of each section of the Stopwatch report shows the arguments for each command. The sequential stopwatch event number is also listed. The next line for each command shows when the record was created.

```
###EVNT-12645:args= -nworst 2000 -max 500
STOPWATCH[start_get_timing_paths]:
  CUR_datetime = Wed Apr 13 03:11:54 2014
```

The command name is prepended with `start_` to indicate that the stopwatch entry was saved before the command started. The report also shows the current values of the

performance and memory metrics and their differences from the previous stopwatch event. For example,

```
STOPWATCH[start_get_timing_paths]: CUR_elapsed      = 29.28 s
STOPWATCH[start_get_timing_paths]: DELTA_elapsed    = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_cputime     = 14.15 s
STOPWATCH[start_get_timing_paths]: DELTA_cputime    = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_peakmem     = 132.60 MB
STOPWATCH[start_get_timing_paths]: DELTA_peakmem   = 0.00 MB
```

After the command finishes executing, the next stopwatch event is recorded. The command name is prepended with `after_` to indicate the performance and memory values after the command finished executing. For example,

```
###EVNT-12646:args= -nworst 2000 -max 500
STOPWATCH[after_get_timing_paths]: CUR_datetime =
  Wed Apr 13 03:11:54 2014
STOPWATCH[after_get_timing_paths]: CUR_elapsed      = 36.91 s
STOPWATCH[after_get_timing_paths]: DELTA_elapsed    = 7.63 s
STOPWATCH[after_get_timing_paths]: CUR_cputime     = 21.63 s
STOPWATCH[after_get_timing_paths]: DELTA_cputime    = 7.48 s
STOPWATCH[after_get_timing_paths]: CUR_peakmem     = 276.54 MB
STOPWATCH[after_get_timing_paths]: DELTA_peakmem   = 143.94 MB
```

[Example 3-12](#) shows portions of the Stopwatch report.

Example 3-12 Stopwatch Report

```
###EVNT-1:args=
STOPWATCH[START]: CUR_datetime      = Wed Apr 13 03:11:21 2014
STOPWATCH[START]: CUR_elapsed       = 2.81 s
STOPWATCH[START]: DELTA_elapsed     = 0.00 s
STOPWATCH[START]: CUR_cputime      = 2.54 s
STOPWATCH[START]: DELTA_cputime    = 0.00 s
STOPWATCH[START]: CUR_peakmem     = 26.57 MB
STOPWATCH[START]: DELTA_peakmem   = 0.00 MB
...
###
###EVNT-12642:args=
STOPWATCH[start_update_timing]: CUR_datetime      = Wed Apr 13 03:11:46 2014
STOPWATCH[start_update_timing]: CUR_elapsed       = 20.40 s
STOPWATCH[start_update_timing]: DELTA_elapsed     = 0.00 s
STOPWATCH[start_update_timing]: CUR_cputime      = 5.54 s
STOPWATCH[start_update_timing]: DELTA_cputime    = 0.00 s
STOPWATCH[start_update_timing]: CUR_peakmem     = 100.60 MB
STOPWATCH[start_update_timing]: DELTA_peakmem   = 0.00 MB
###EVNT-12643:args=
STOPWATCH[after_update_timing]: CUR_datetime      = Wed Apr 13 03:11:46 2014
STOPWATCH[after_update_timing]: CUR_elapsed       = 29.28 s
STOPWATCH[after_update_timing]: DELTA_elapsed     = 8.88 s
STOPWATCH[after_update_timing]: CUR_cputime      = 14.15 s
STOPWATCH[after_update_timing]: DELTA_cputime    = 8.61 s
STOPWATCH[after_update_timing]: CUR_peakmem     = 132.60 MB
STOPWATCH[after_update_timing]: DELTA_peakmem   = 32.00 MB
###EVNT-12644:args= /u/tests/tcl_prof/gtp_only.tcl
STOPWATCH[start_source]: CUR_datetime      = Wed Apr 13 03:11:54 2014
STOPWATCH[start_source]: CUR_elapsed       = 29.28 s
```

```

STOPWATCH[start_source]: DELTA_elapsed      = 0.00 s
STOPWATCH[start_source]: CUR_cputime        = 14.15 s
STOPWATCH[start_source]: DELTA_cputime      = 0.00 s
STOPWATCH[start_source]: CUR_peakmem        = 132.60 MB
STOPWATCH[start_source]: DELTA_peakmem      = 0.00 MB
##EVNT-12645:args= -nworst 2000 -max 500
STOPWATCH[start_get_timing_paths]: CUR_datetime = Wed Apr 13 03:11:54 2014
STOPWATCH[start_get_timing_paths]: CUR_elapsed   = 29.28 s
STOPWATCH[start_get_timing_paths]: DELTA_elapsed   = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_cputime    = 14.15 s
STOPWATCH[start_get_timing_paths]: DELTA_cputime   = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_peakmem    = 132.60 MB
STOPWATCH[start_get_timing_paths]: DELTA_peakmem   = 0.00 MB
##EVNT-12646:args= -nworst 2000 -max 500
STOPWATCH[after_get_timing_paths]: CUR_datetime = Wed Apr 13 03:11:54 2014
STOPWATCH[after_get_timing_paths]: CUR_elapsed   = 36.91 s
STOPWATCH[after_get_timing_paths]: DELTA_elapsed   = 7.63 s
STOPWATCH[after_get_timing_paths]: CUR_cputime    = 21.63 s
STOPWATCH[after_get_timing_paths]: DELTA_cputime   = 7.48 s
STOPWATCH[after_get_timing_paths]: CUR_peakmem    = 276.54 MB
STOPWATCH[after_get_timing_paths]: DELTA_peakmem   = 143.94 MB
##EVNT-12651:args= /u/tests/tcl_prof/gtp_only.tcl
STOPWATCH[after_source]: CUR_datetime       = Wed Apr 13 03:11:54 2014
STOPWATCH[after_source]: CUR_elapsed         = 36.91 s
STOPWATCH[after_source]: DELTA_elapsed        = 0.00 s
STOPWATCH[after_source]: CUR_cputime         = 21.63 s
STOPWATCH[after_source]: DELTA_cputime        = 0.00 s
STOPWATCH[after_source]: CUR_peakmem         = 276.54 MB
STOPWATCH[after_source]: DELTA_peakmem        = 0.00 MB
...
##EVNT-12664:args=
STOPWATCH[STOP]: CUR_datetime      = Wed Apr 13 03:11:59 2014
STOPWATCH[STOP]: CUR_elapsed        = 41.89 s
STOPWATCH[STOP]: DELTA_elapsed      = 0.00 s
STOPWATCH[STOP]: CUR_cputime        = 21.68 s
STOPWATCH[STOP]: DELTA_cputime      = 0.00 s
STOPWATCH[STOP]: CUR_peakmem        = 294.82 MB
STOPWATCH[STOP]: DELTA_peakmem      = 0.00 MB

```


4

Design Data

The first step in any timing analysis is to load the design description and associated logic libraries. PrimeTime accepts design data in .ddc, .db, Verilog, and VHDL formats. It accepts logic libraries in .db format, which are typically provided by the ASIC vendor. You can load libraries that have different units (time, capacitance, and voltage). The main library is the first library in your link and target path.

To learn more about reading, reporting, and modifying design data, see

- [Search Path and Link Path](#)
- [Reading Design and Library Data](#)
- [Reading Verilog and VHDL Design Data](#)
- [Using a Milkyway Database](#)
- [Removing Designs and Libraries](#)
- [Setting the Current Design and Current Instance](#)
- [Listing Design and Library Information](#)
- [Linking the Design](#)
- [Checking the Consistency of Units With the set_units Command](#)
- [Design Objects](#)

Search Path and Link Path

The `search_path` and `link_path` variables control the directory paths in which PrimeTime searches for design data. The `search_path` variable specifies a list of directory paths that PrimeTime uses to find the designs, libraries, and other files. If you include the path to a file in the `search_path` variable, PrimeTime can find the file even if you do not specify an absolute path when you read the file. The `search_path` variable typically includes paths to design database files, logic libraries, and timing models.

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

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

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

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

1. Specify the search path as a list of paths to the directories containing the design files, timing models files, and library files needed for the analysis. For example,

```
pt_shell> set_app_var search_path ". /u/project/design \
           /u/project/library"
           . /u/project/design /u/project/library
```

Delimit each path with a space and enclose the list in double quotation marks (" ").

2. Specify the link path as a list of the library names. For example,

```
pt_shell> set_app_var link_path "* STDLIB.db"
           * STDLIB.db
```

Use an asterisk (*) to search for designs in memory. Delimit each library with a space and enclose the list in double quotation marks (" ").

Reading Design and Library Data

To analyze a design, you need to read the design data and associated logic libraries into PrimeTime. You can use the `read_ddc` command to read design information, including netlists and constraints for a design, from .ddc databases produced by other tools, including Design Compiler and IC Compiler. Otherwise, you can use the `read_db` command to read the logic libraries and design data in .db format. For information about reading design data in other formats, see [Reading Verilog and VHDL Design Data](#).

Reading Design Data in .ddc Format

To load the design information in .ddc format, follow these steps:

1. Set the search path. For example,

```
pt_shell> set_app_var search_path ". /u/proj/des1 /u/proj/lib1"
```

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

2. Optional: set the link path. For example,

```
pt_shell> set_app_var link_path "* STDLIB.db"
```

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

3. Read the .db format logic library into memory. For example,

```
pt_shell> read_db STDLIB.db
```

Note:

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

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

```
pt_shell> read_ddc TOP.ddc
pt_shell> read_ddc module1.ddc
pt_shell> read_ddc module2.ddc
pt_shell> read_ddc module3.ddc
pt_shell> read_ddc module4.ddc
```

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

```
pt_shell> link_design TOP
```

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

Reading Design Data in .db Format

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

1. Set the search path. For example, enter

```
pt_shell> set_app_var search_path ". /u/proj/des1 /u/proj/lib1"
```

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

2. Set the link path. For example, enter

```
pt_shell> set_app_var link_path "* STDLIB.db"
```

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

3. Read the logic library into memory. For example,

```
pt_shell> read_db STDLIB.db
```

Note:

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

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

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

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

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

```
pt_shell> link_design TOP
```

After the files are loaded, you can view a list of the loaded designs or libraries by using the `list_designs` or `list_libs` command. By default, the case sensitivity of the linking process is determined by the input format that created the reference. To explicitly define the

case sensitivity of the linking process, set the `link_force_case` variable. Changing the case sensitivity when reading in source files from case-sensitive formats might cause inconsistent and unexpected results.

Reading Verilog and VHDL Design Data

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

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

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

Using the PrimeTime Verilog Reader

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

- `module`, `endmodule`
- `input`, `output`, `inout`
- `wire`
- `assign`, `tran`
- `supply0`, `supply1`
- `wand`, `wor`, `tri`

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

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

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

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

You turn translation back on using one of these directives:

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

Optional Preprocessor

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

- `define
- `undef
- `ifdef
- `include
- `else
- `endif

Without the preprocessor, the Verilog reader cannot recognize these directives. The preprocessor writes a set of intermediate files into the directory specified by the `pt_tmp_dir` variable. When the preprocessor is done, the Verilog reader reads the resulting output. Note that the Verilog reader does not operate as fast with the preprocessor enabled. When you use the ``include` directive in the Verilog file, PrimeTime looks for the referenced files in the directories defined by the `search_path` variable.

Assign Statements and Synonyms

The Verilog reader creates synonyms for discarded names in an assign statement. One net name is chosen, and the others assigned to it are synonyms. During back-annotation, an explicitly named net can be found through one of its synonyms. Although the `get_nets` command finds nets using their synonyms, it cannot find them when mixed with any wildcards. For example, given the assign statement of `assign n1 = n2;` where `n1` wins, using the `get_nets` command finds the real net, as in:

```
pt_shell> get_nets n1  
{ "n1" }
```

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

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

You cannot use wildcards with synonyms:

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

Physical Verilog Power Rail Connections

The `set_lib_rail_connection` command allows PrimeTime to read a design from a physical Verilog description and link it to a corresponding logical library that does not contain power and ground pins.

When reading Verilog or VHDL with power and ground pin specifications in the netlist, the power and ground pins listed with the `-lib_pin_names` option are ignored during linking.

PrimeTime Verilog Reader Limitations

The following limitations apply to the native Verilog reader:

- The `wand`, `wor`, and `tri` statements are essentially synonyms for wire. Each creates a wire with no special attributes.
- There is very limited support for global naming, that is, referencing a wire from a different module. For example, `global.gnd` means wire `gnd` in module `global`. Global references are allowed only in instance connections. They cannot be in any other context. In addition, you must ensure the following:
 - Reference is a logic constant.
 - Global reference is not bussed.
 - Referenced module is defined in the same file as the module that is referencing it.
 - Referenced module is defined first.
 - The global name is used over the default name, but a local name is used over the global name. For example, if `global.gnd` and `1'b0` are used, `gnd` is the wire name; however, if `ZERO` is assigned to `1'b0`, and `global.gnd` is also used, `ZERO` is used.

Using the HDL Compiler Verilog Reader

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

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

When you are using ptxr to read multiple designs from a single file and there is an error reading a design, PrimeTime stops reading the file when it hits an error.

Note:

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

Setup Files for the HDL Compiler Verilog Reader

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

You can create a user-defined `ptxr_setup_file` variable in PrimeTime to reference a ptxr-specific setup file. The ptxr setup file is written in Tcl. This file can contain a limited set of commands:

- Comments
- Blank lines
- Variable assignments

Here is an example of the ptxr setup file:

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

To use the ptxr setup file:

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

To discontinue using the ptxr setup file:

```
pt_shell> unset ptxr_setup_file
```

Note:

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

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

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

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

Reading VHDL Design Files

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

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

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

Setup Files for the VHDL Compiler

When you use the VHDL Compiler, the ptxr program reads your `.synopsys_dc.setup` files (not `.synopsys_pt.setup`) to access the VHDL variables. These include the system, home, and local setup files. The system setup file must always be read. However, it is possible for `read_vhdl` to skip the home and local setup files. You can define the `ptxr_setup_file`

variable in PrimeTime to reference a ptxr-specific setup file. By default, this variable does not exist. The ptxr setup file is written in Tcl. This file can contain a very limited set of commands: comments, blank lines, and variable assignments. For example, you could create `my_ptxr.setup`:

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

To use the ptxr setup file:

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

To discontinue using the ptxr setup file:

```
pt_shell> unset ptxr_setup_file
```

Using a Milkyway Database

PrimeTime can read Milkyway databases directly. Milkyway is the standard database format used to store data for all Synopsys tools in the Galaxy Design Platform. The Milkyway database provides improved operational consistency between PrimeTime related tools such as Physical Compiler, and IC Compiler. PrimeTime reads various types of information in the Milkyway database, including netlists, libraries, constraints, and parasitic data.

Reading a Milkyway Database

The `read_milkyway` command reads in the Milkyway database. You can also use it to load the timing constraints from the Milkyway database.

Before using the command, set the `search_path` variable to specify the location of the logic libraries and the `link_path` variable to specify the libraries used for linking the design. If you set the `link_path_per_instance` variable so certain instances have different link paths, the setting is honored in the `read_milkyway` flow.

By default, PrimeTime reads the linked netlist and constraints from the Milkyway database. PrimeTime pulls in constraints from one or multiple constraint files corresponding to different options you have set. For example, the following reads in the latest version of the design `ms_des` into PrimeTime from the `mw_db` Milkyway database. This example reads in only the netlist of the `mw_des` CEL stored in the `mw_db` design library; it ignores any constraints associated with the netlist.

```
pt_shell> read_milkyway -netlist_only -library mw_db mw_des
```

The `read_milkyway` command reads in a fully resolved, linked design. Do not use the `link_design` command afterward because it could invalidate certain types of cell hierarchy. This method of reading is consistent with reading Milkyway designs into IC Compiler. However, it is different from reading designs in Verilog or VHDL format, which require `link_design` after reading.

You can instruct PrimeTime to read parasitics from the parasitics file in the database using the `read_parasitics` command. When a `read_milkyway` command precedes `read_parasitics`, the `read_parasitics` command does not need to specify any file names; it uses the same library and CEL_name as the `read_milkyway` command. The following example reads in a netlist, constraints, and parasitics from a Milkyway database (extending the previous example):

```
pt_shell> read_milkyway -library mw_db mw_des  
pt_shell> read_parasitics -format PARA
```

The Milkyway database is capable of storing multiple constraints that can correspond to various scenarios of running the design. If your Milkyway database was written specifying a particular scenario for the constraints, you need to specify the scenario when using `read_milkyway` in PrimeTime by using the `-scenario` option.

Writing a Milkyway Database

You write out a Milkyway database in IC Compiler, Physical Compiler, or Design Compiler. PrimeTime cannot write out a Milkyway database. For more information about generating a Milkyway database, see the documentation for these products.

Limitations When Reading Milkyway Format

If a Milkyway design library was created by IC Compiler, Physical Compiler, or Design Compiler, the design constraints are loaded into memory when you use the `read_milkyway` command to read the Milkyway design. However, if the Milkyway design library was created by Jupiter, the design constraints are not loaded into memory by the `read_milkyway` command.

Note:

This limitation also applies to a Milkyway design library that was updated in Jupiter, even if it was initially created by IC Compiler, Physical Compiler, or Design Compiler.

To ensure that all design constraints are loaded into memory, use one of the following methods to read a Milkyway design library that was not created or last updated in IC Compiler, Physical Compiler, or Design Compiler.

For the Milkyway and SDC files, use the `read_milkyway` command in PrimeTime to read the Milkyway design library, then reapply the design constraints by reading the golden SDC file:

```
pt_shell> read_milkyway -netlist_only my_design  
pt_shell> read_sdc my_constraints.sdc
```

For IC Compiler, Physical Compiler, or Design Compiler, read the Milkyway design library into the appropriate tool in default XG mode, then reapply the golden SDC file and write out the Milkyway design library:

```
psyn_shell-xg-t> read_milkyway my_design  
psyn_shell-xg-t> source my_constraints.sdc  
psyn_shell-xg-t> write_milkyway -output my_design
```

Removing Designs and Libraries

The `remove_design -all` command removes all designs from PrimeTime. To selectively remove designs by name or just the current design and its hierarchy, use the `remove_design design_names` or `remove_design -hierarchy` commands.

The `remove_lib -all` command removes all libraries. To selectively remove libraries by name, use the `remove_lib library_names` command.

Setting the Current Design and Current Instance

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

If you specify a design, PrimeTime sets the current design to that design. This argument can be the name of a design, or a collection containing one design. If you do not specify a design, PrimeTime returns a collection containing the current design. You can use the `current_design` command as a parameter to other PrimeTime commands. To display designs in PrimeTime memory, use the `list_designs` command.

To see the current context and to change the context from one design to another, use a command sequence like the following:

```
pt_shell> current_design
{ "TOP" }

pt_shell> list_designs

Design Registry:
    ADDER          /designs/dbs/my_design.ddc:ADDER
    FULL_ADDER    /designs/dbs/my_design.ddc:FULL_ADDER
    FULL_SUBTRACTOR /designs/dbs/
my_design.ddc:FULL_SUBTRACTOR
    HALF_ADDER    /designs/dbs/my_design.ddc:HALF_ADDER
    HALF_SUBTRACTOR /designs/dbs/
my_design.ddc:HALF_SUBTRACTOR
    SUBTRACTOR    /designs/dbs/my_design.ddc:SUBTRACTOR
*      TOP          /designs/dbs/my_design.ddc:TOP

pt_shell> current_design ADDER
{ "ADDER" }
```

The `current_instance` command sets the working instance object (cell in the design hierarchy) and enables other commands to be used relative to the instance. This command differs from the `current_design` command, which changes the working design, then sets the current instance to the top level of the new current design. Combining the `current design` and `current instance` defines the context for many PrimeTime commands.

The `current_instance` command traverses the design hierarchy similarly to the way the UNIX `cd` command traverses the file hierarchy. This command operates with a variety of `instance_name` arguments:

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

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

Examples

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

```
pt_shell> current_design TOP
{ "TOP" }

pt_shell> current_instance U1
U1

pt_shell> current_instance "."
U1

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

pt_shell> current_instance U3
U1/U3

pt_shell> current_instance ".../U4"
U1/U4

pt_shell> current_instance
Current instance is the top-level of design 'TOP'.
```

To use `current_design` to reset the current instance to the top level of the new design hierarchy, use a command sequence like the following:

```
pt_shell> current_design ADDER
{ "ADDER" }

pt_shell> current_instance .
Current instance is the top-level of design 'ADDER'.
```

To use the `current_instance` command to go to an instance of another design, use a command sequence like the following. The new design whose name is the name after the first slash of the given instance name sets the current design.

```
pt_shell> current_instance "/ALARM_BLOCK/U6"
U6
pt_shell> current_design
{ "ALARM_BLOCK" }
```

Listing Design and Library Information

PrimeTime provides commands for listing designs and libraries that are loaded. The `list_designs` command lists the designs that are in PrimeTime memory, including the current design, linked or partially linked designs, and designs instantiated in a linked design.

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

- * – Design is the current design.
- L – Design is linked.
- N – Design is not in memory.
- 1 – Design is partially linked.

To list the designs in memory, enter

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

To list the designs in use and in memory, enter

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

After removing the `MULTI` design, you can display it using the `-all` option. For example, enter

```
pt_shell> remove_design MULTI
Removing design 'MULTI'...
1
pt_shell> list_designs -all -only_used
Design Registry:
*L AD4FULA      /u/designs/top.db:AD4FULA
    ADD5A        /u/designs/add2.db:ADD5A
N MULT1         /u/designs/top.db:MULT1
```

The `list_libs` command lists the libraries in memory.

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

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

- “*” (asterisk) indicates the main library of the current design; the first library in the link path that has a time unit.
- “M” indicates the maximum library of a max-min pair.
- “m” indicates the minimum library of a max-min pair.

To list all the libraries in memory, enter

```
pt_shell> list_libs
Library Registry:
my_lib          /u/lib/my_lib.db:my_lib
tech1           /u/lib/tech1.db:tech1
```

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

```
pt_shell> link_design top_flat
Linking design top_flat...
Design 'top_flat' was successfully linked.
1

pt_shell> list_libs -only_used
Library Registry:
* tech1          /u/lib/tech1.db:tech1
```

Linking the Design

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

To link a design called `example`, enter

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

```
Design 'example' was successfully linked.
1
```

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

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

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

Per-Instance Link Paths

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

```
set link_path {* lib1.db}
set link_path_per_instance [list
    [list {ucore} {* lib2.db}]
    [list {ucore/usubblk} {* lib3.db}]]
```

Specified entries are used to link the specified level and below. If a given block matches multiple entries in the per-instance list, the more specific entry overrides the more general entry. In the earlier example:

- `lib3.db` is used to link blocks “`ucore/usubblk`” and below.
- `lib2.db` is used to link “`ucore`” and below (except within “`ucore/subblk`”).
- `lib1.db` is used for the remainder of the design (everything except within “`ucore`”).

The default of `link_path_per_instance` is an empty list, meaning that the feature is disabled. To determine the current value of this variable, use `set link_path` or `echo $link_path`.

Handling Incomplete Data

During early design development, design data needs to be updated regularly. For example, a block recently updated by a block-level designer might have fewer pins than the same block that was used by a top-level designer.

You can continue working on the design and gathering useful information even in the presence of incomplete or “dirty” data. The `link_allow_design_mismatch` variable controls whether a design is successfully linked in the presence of dirty data. By default, the variable is set to `false`, and the design fails to link when mismatches exist. When you set this variable to `true`, mismatches between an instance and reference are ignored, the design links successfully, and you can continue working on the design.

The `is_design_mismatch` attribute is available on each cell, pin, and net. If an object is dirty, that is, the object is directly affected by a mismatch between the block and top-level design, the `is_design_mismatch` attribute of the object is `true`. Use the `list_attributes` command to list the currently defined attributes.

The following information explains how PrimeTime handles dirty data and also explains what actions are required, if any:

- Top-level design has pins that are not available in subdesigns
 - If the higher-level hierarchies have pins that do not exist at the lower level, these extra pins are ignored and do not exist in the linked netlist. Nets connected to missing pins are disconnected and left dangling. Annotations, such as constraints and parasitics, on these pins are not applied. The LNK-038 warning message is issued for each pin that is ignored.
- Library and netlist cell contains different power and ground (PG) information
 - If the library and netlist cell have different PG information, the information in the library cell is assumed to be accurate. If the library cell does not have PG information, neither does the linked netlist.
- Direction of pins differ depending on the hierarchical level
 - If pins at different hierarchy levels have different directions, the direction of the pins of the lower-level cells in the design is used.
- Pin names are case-sensitive
 - The case of pin names at different hierarchical levels might not match. For example, a pin might be named UBLK/CLK at the top level and UBLK/clk at the block level. If the existing `link_force_case` variable is set to `case_insensitive`, the case of the pin is ignored and is not considered dirty. The default of the `link_force_case` variable is `check_reference`, which means the case sensitivity of the link is determined by the input format that was used to create the reference.

- Bus width causes mismatch

If the width of a bus varies for different hierarchical blocks, the bits of the bus are connected beginning with the least significant bits. If the lower-level block has additional bus bits not present in the next higher level, the bits are unconnected. If the higher-level block has additional bits that are not present in the lower-level block, the pins are ignored, and the LNK-038 warning message is issued for each ignored pin.

To view the mismatches located while linking a design, use the `report_design_mismatch` command. For example,

```
pt_shell> report_design_mismatch
report_design_mismatch

...
pin           mismatch type
-----
u1/Z          Pin direction mismatch
u2/Z          Pin direction mismatch
-----
cell          mismatch type
-----
u1            Pin direction mismatch
u2            Pin direction mismatch
-----
net           mismatch type
-----
b             Pin direction mismatch
```

Link Errors

Certain link errors can cause the linking of the design to fail. When the `link_design` command fails, the output is a design with unresolved references.

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

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

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

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

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

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

- The instance has scalar pins like `A[0]` and `A[1]`, but the reference has a bus A. Sometimes this indicates a library or subdesign that is not synchronized with the top design, which might have been flattened.

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

Checking the Consistency of Units With the `set_units` Command

To check that the specified units are consistent with the main library units, use the `set_units` command. If the specified units differ from the main library units, PrimeTime issues a warning message. You can use the `set_units` command in Synopsys Design Constraints (SDC) files, on the command line, or in script files. Although this command is optional, you should use it to identify the inadvertent use of inconsistent units from different constraint files.

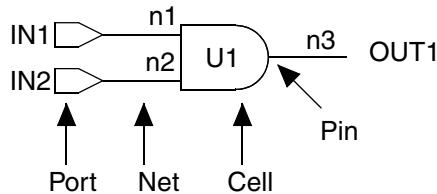
The `write_sdc` command writes out the `set_units` command as the first command. The output of reporting commands is unaffected by the units specified with the `set_units` command. Reports continue to use the units defined in the main library.

The following example shows how to use the `set_units` command:

```
set_units -time 1ns -capacitance 1nF -current 1mA \
           -voltage 1V -resistance 1kOhm -power 1W
```

Design Objects

ASIC designs are hierarchical entities composed of objects such as cells, ports, and nets (see [Figure 4-1](#)). To perform detailed timing analysis and locate the source of timing problems, you need access to these objects. PrimeTime lets you create a collection of one or more objects for querying, reporting, or creating timing assertions.

Figure 4-1 Typical ASIC Design Objects

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

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

State Objects

PrimeTime uses the state objects in [Table 4-1](#).

Table 4-1 State Objects

Netlist object	Description
Current design	The design that is the current top of the hierarchy. Most objects are referenced relative to the current design. The <code>current_design</code> command sets the working design for many PrimeTime commands.
Current instance	The instance (or hierarchical cell) that is the current scope within the current design. Traverse the hierarchy by changing the current instance. The <code>current_instance</code> command sets the working instance object (cell) and enables other commands to be used relative to a specific instance in the design hierarchy.

Netlist Objects

PrimeTime commands, attributes, and assertions are directed toward a netlist object. You can create collections of netlist objects. PrimeTime uses the netlist objects in [Table 4-2](#).

Table 4-2 Netlist Objects

Netlist object	Description
cell	The instances in the design, including those that reference hierarchical blocks and primitive library cells.
clock	Clock objects do not exist in a design by default. They are created when you define the clocks for PrimeTime. Clocks are also imported from .ddc or .db format files.
design	A design.
lib	A library.
lib_cell	The cells in a logic library.
lib_pin	The pins on library cells.
net	The nets in the current design.
path group	Collections of paths considered as a group in the cost-function calculations that drive logic synthesis in Design Compiler. A group has information, such as a set of paths, a name, and weight. The weight allows you to control the way the maximum delay cost is computed when you optimize designs in Design Compiler. Timing reports are also organized by path group.
pin	The pins of lower-level cells in the design. Pins can be input, output, or inout (bidirectional).
port	The ports of the current design. Ports can be input, output, or inout (bidirectional).
register	The primitive instances of a design that have clocks defined on them. These instances include flip-flops and latches.

5

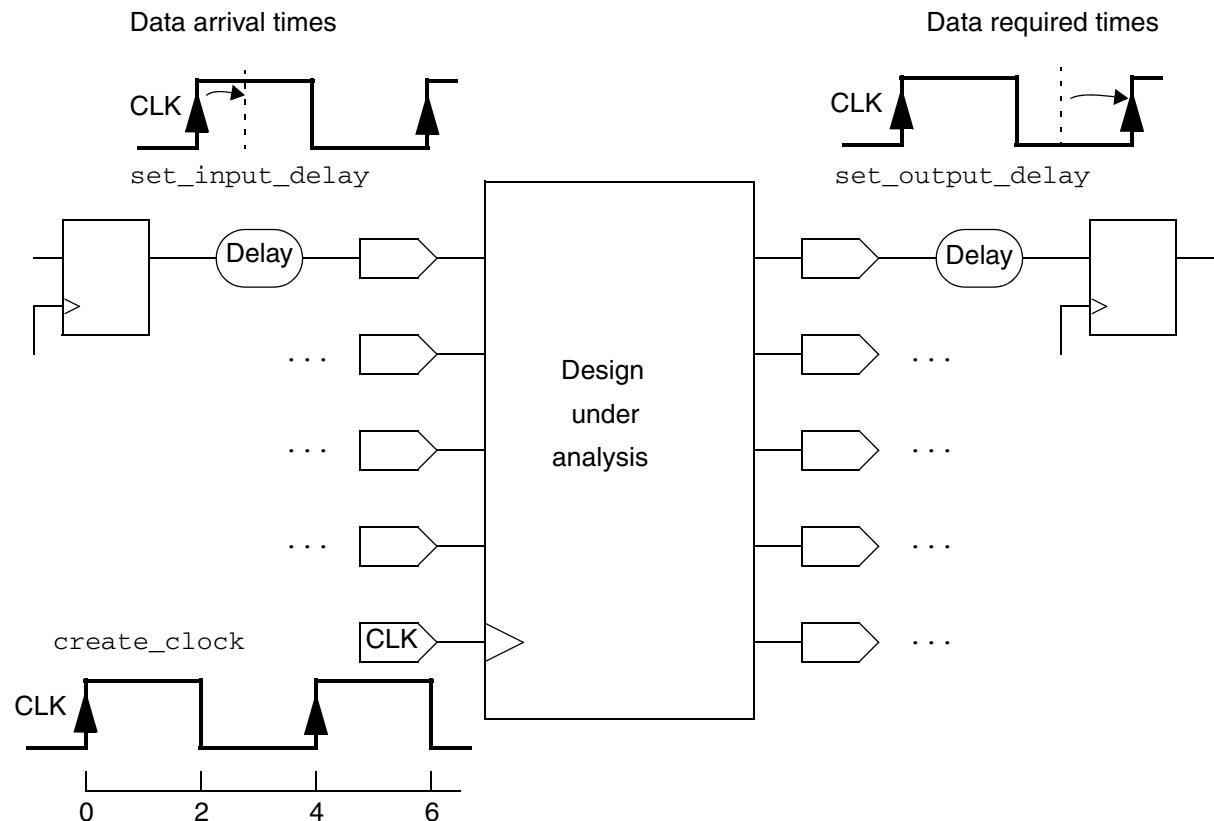
Design Constraints

To perform timing analysis, you must specify the following information about the inputs and outputs of the design:

- [Input Delays](#)
- [Output Delays](#)
- [Drive Characteristics at Input Ports](#)
- [Port Capacitance](#)
- [Wire Load Models](#)
- [Slew Propagation](#)
- [Design Rule Constraints](#)
- [Ideal Networks](#)

[Figure 5-1](#) shows these design-level timing constraints (also called *timing assertions*).

Figure 5-1 Design-Level Timing Constraints



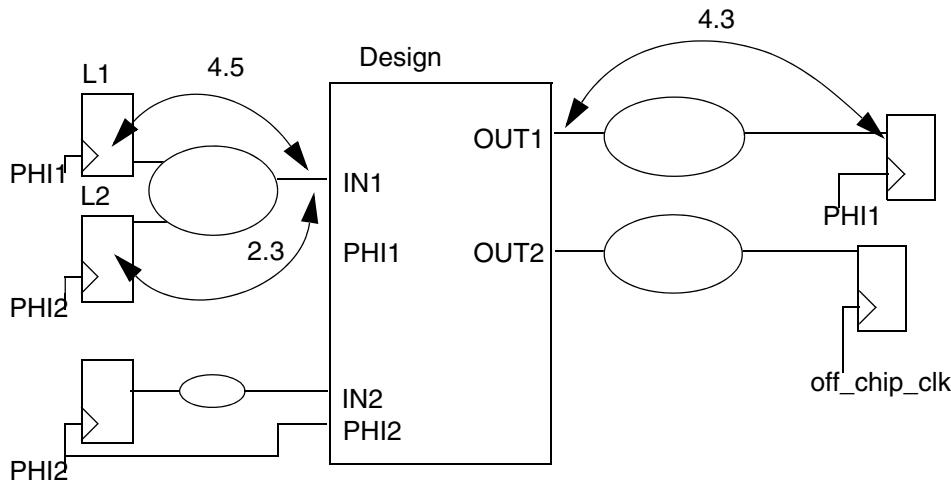
Input Delays

To do constraint checking at the inputs of the design, the tool needs information about the arrival times of signals at the inputs. To specify the timing of external paths leading to an input port, use the `set_input_delay` command. The tool uses this information to check for timing violations at the input port and in the transitive fanout from that input port. With this command, you specify the minimum and maximum amount of delay from a clock edge to the arrival of a signal at a specified input port.

Applying the `set_drive` or `set_driving_cell` commands to the port causes the port to have a cell delay that is the load-dependent value of the external driving-cell delay. To prevent this delay from being counted twice, estimate the load-dependent delay of the driving cell, then subtract that amount from the input delays on the port.

The input delay should equal the path length from the clock pin of the source flip-flop to the output pin of the driving cell, minus the load-dependent portion of the driving cell's delay. For example, see the external path for the L1 clock port to port IN1 in [Figure 5-2](#).

Figure 5-2 Two-Phase Clocking Example for Setting Port Delays



When you use the `set_input_delay` command, you can specify whether the delay value includes the network latency or source latency.

Example 1

If the delay from L1 clock port to IN1 (minus the load-dependent delay of the driving cell) is 4.5, this `set_input_delay` command applies:

```
pt_shell> set_input_delay 4.5 -clock PHI1 {IN1}
```

Example 2

If paths from multiple clocks or edges reach the same port, specify each one using the `-add_delay` option. If you omit the `-add_delay` option, existing data is removed. For example,

```
pt_shell> set_input_delay 2.3 -clock PHI2 -add_delay {IN1}
```

If the source of the delay is a level-sensitive latch, use the `-level_sensitive` option. This allows PrimeTime to determine the correct single-cycle timing constraint for paths from this port. Use the `-clock_fall` option to denote a negative level-sensitive latch; otherwise, the `-level_sensitive` option implies a positive level-sensitive latch.

To see input delays on ports, use the `report_port -input_delay` command.

To remove input delay information from ports or pins in the current design set using the `set_input_delay` command, use the `remove_input_delay` command. The default is to remove all input delay information in the `port_pin_list` option.

Using Input Ports Simultaneously for Clock and Data

PrimeTime allows an input port to behave simultaneously as a clock and data port. You can use the `timing_simultaneous_clock_data_port_compatibility` variable to enable or disable the simultaneous behavior of the input port as a clock and data port. When this variable is `false`, the default, simultaneous behavior is enabled and you can use the `set_input_delay` command to define the timing requirements for input ports relative to a clock. In this situation, the following applies:

- If you specify the `set_input_delay` command relative to a clock defined at the same port and the port has data sinks, the command is ignored and an error message is issued. There is only one signal coming to port, and it cannot be at the same time data relative to a clock and the clock signal itself.
- If you specify the `set_input_delay` command relative to a clock defined at a different port and the port has data sinks, the input delay is set and controls data edges launched from the port relative to the clock.
- Regardless of the location of the data port, if the clock port does not fanout to data sinks, the input delay on the clock port is ignored and you receive an error message.

When you set the `timing_simultaneous_clock_data_port_compatibility` variable to `true`, the simultaneous behavior is disabled and the `set_input_delay` command defines the arrival time relative to a clock. In this situation, when an input port has a clock defined on it, PrimeTime considers the port exclusively as a clock port and imposes restriction on the data edges that are launched. PrimeTime also prevents setting input delays relative to another clock.

To control the clock source latency for any clocks defined on an input port, you must use the `set_clock_latency` command.

Output Delays

To do constraint checking at the outputs of the design, the tool needs information about the timing requirements at the outputs. To specify the delay of an output port to a register, use the `set_output_delay` command.

With this command, you specify the minimum and maximum amount of delay between the output port and the external sequential device that captures data from that output port. This setting establishes the times at which signals must be available at the output port to meet the setup and hold requirements of the external sequential element:

- $\text{Maximum_output_delay} = \text{length_of_longest_path_to_register_data_pin} + \text{setup_time_of_the_register}$
- $\text{Minimum_output_delay} = \text{length_of_shortest_path_to_register_data_pin} - \text{hold_time}$

To show output delays associated with ports, use the `report_port -output_delay` command.

To remove output delay from output ports or pins set through the `set_output_delay` command, use the `remove_output_delay` command. By default, all output delays on each object in the port or pin list are removed. To restrict the removed output delay values, use the `-clock`, `-clock_fall`, `-min`, `-max`, `-rise`, or `-fall` option.

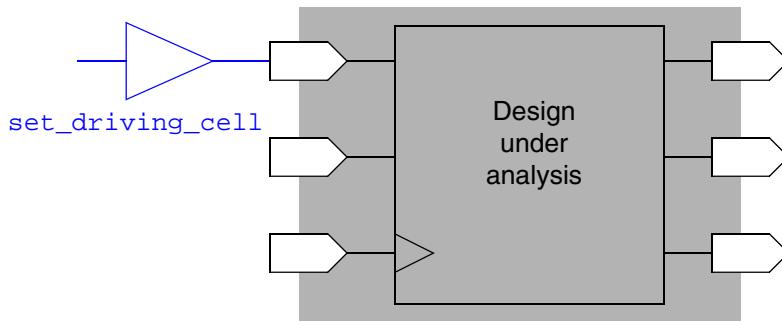
Example

To set an output delay of 4.3 relative to the rising edge of clock PHI1 on port OUT1 ([Figure 5-2](#)):

```
pt_shell> set_output_delay 4.3 -clock PHI1 {OUT1}
```

Drive Characteristics at Input Ports

To accurately time a design, you need to define the drive capability of the external cell driving each input port, as shown in [Figure 5-3](#). PrimeTime uses this information to calculate the load-dependent cell delay for the port and to produce an accurate transition time for calculating cell delays and transition times for the following logic stages.

Figure 5-3 Driving Cells

The `set_driving_cell` command can specify a library cell arc for the driving cell so that timing calculations are accurate even if the capacitance changes. This command causes the port to have the transition time calculated as if the given library cell were driving the port.

For less precise calculations, you can use the `set_drive` or `set_input_transition` command. The most recent drive command has precedence. If you issue the `set_drive` command on a port and then use the `set_driving_cell` command on the same port, information from the `set_drive` command is removed.

Setting the Port Driving Cell

The `set_driving_cell` command directs PrimeTime to calculate delays as though the port were an instance of a specified library cell. The port delay is calculated as a cell delay that consists of only the load-dependent portion of the port.

The transition time for the port is also calculated as though an instance of that library cell were driving the net. The delay calculated for a port with information from the `set_driving_cell` command takes advantage of the actual delay model for that library cell, whether it is nonlinear or linear. The input delay specified for a port with a driving cell or drive resistance should not include the load-dependent delay of the port.

To display port transition or drive capability information, use the `report_port` command with the `-drive` option.

With the `set_driving_cell` command you can specify the input rise and fall transition times for the input of a driving cell using the `-input_transition_rise` or the `-input_transition_fall` option. If no input transition is specified, the default is 0.

Setting the Port Drive Resistance

The `set_drive` command defines the external drive strength or resistance for input and inout ports in the current design. In the presence of wire load models, the transition time and delay reported for the port are equal to $R_{driver} * C_{total}$. PrimeTime uses this transition time in calculating the delays of subsequent logic stages.

You can use the `set_drive` command to set the drive resistance on the top-level ports of the design when the input port drive capability cannot be characterized with a cell in the logic library. However, this command is not as accurate for nonlinear delay models compared to the `set_driving_cell` command. The `set_drive` command is useful when you cannot specify a library cell (for example, when the driver is a custom block not modeled as a Synopsys library cell).

Setting a Fixed Port Transition Time

The `set_input_transition` command defines a fixed transition time for input ports. The port has zero cell delay. PrimeTime uses the specified transition time only in calculating the delays of logic driven by the port.

A fixed transition time setting is useful for

- Comparing the timing of a design to another tool that supports only input transition time.
- Defining transition for ports at the top level of a chip, where a large external driver and a large external capacitance exist. In this case, the transition time is relatively independent of capacitance in the current design.

Displaying Drive Information

To display drive information, you can use the `report_port` command with the `-drive` option.

Removing Drive Information From Ports

The commands in [Table 5-1](#) remove drive information from ports.

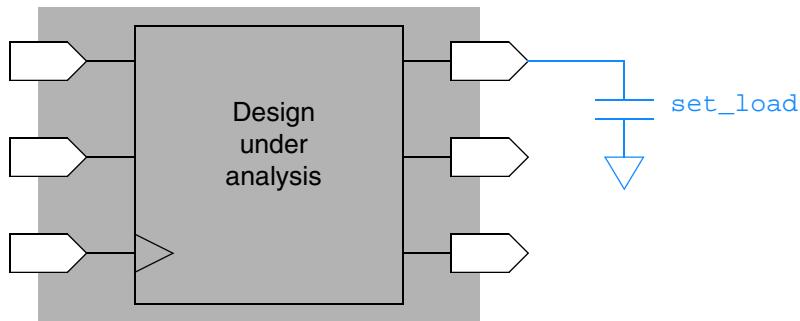
Table 5-1 Commands to Remove Drive Information

To remove this	Use this command
Driving cell information from a list of ports	<code>remove_driving_cell</code>
Drive resistance	<code>set_drive 0.0</code>
Input transition	<code>set_input_transition 0.0</code>
Drive data and all user-specified data, such as clocks, input and output delays	<code>reset_design</code>

Port Capacitance

To accurately time a design, you need to describe the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance, as shown in [Figure 5-4](#). To do this, use the `set_load` command.

Figure 5-4 Output Load



Example 1

To specify the external pin capacitance of ports, enter

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

You also need to account for wire capacitance outside the port. For prelayout, specify the external wire load model and the number of external fanout points. This process is described in [Setting Wire Load Models Manually](#).

Example 2

For postlayout, specify the external annotated wire capacitance as wire capacitance on the port. For example, enter

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

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

Wire Load Models

To accurately calculate net delays, PrimeTime needs information about the parasitic loads of the wire interconnections. Before placement and routing have been completed, PrimeTime estimates these loads by using wire load models provided in the logic library.

Logic library vendors supply statistical wire load models to support estimating wire loads based on the number of fanout pins on a net. You can set wire load models manually or automatically.

Setting Wire Load Models Manually

To manually set a named wire load model on a design, instance, list of cells, or list of ports, use the `set_wire_load_model` command.

For example, consider this design hierarchy:

```
TOP
  MID (instance u1)
    BOTTOM (instance u5)
  MID (instance u2)
    BOTTOM (instance u5)
```

To set a model called 10x10 on instances of BOTTOM, a model called 20x20 on instances of MID, and a model called 30x30 on nets at the top level, use these commands:

```
pt_shell> set_wire_load_mode enclosed
pt_shell> set_wire_load_model -name 10x10 \
           [all_instances BOTTOM]
pt_shell> set_wire_load_model -name 20x20 \
           [all_instances MID]
pt_shell> set_wire_load_model -name 30x30
```

To capture the external part of a net that is connected to a port, you can set an external wire load model and a number of fanout points. For example, to do this for port Z of the current design:

```
pt_shell> set_wire_load_model -name 70x70 [get_ports Z]
pt_shell> set_port_fanout_number 3 Z
```

To calculate delays, PrimeTime assumes that port Z has a fanout of 3 and uses the 70x70 wire load model.

To see wire load model settings for the current design or instance, use the `report_wire_load` command. To see wire load information for ports, use the `report_port` command with `-wire_load` option. To remove user-specified wire load model information, use the `remove_wire_load_model` command.

Automatic Wire Load Model Selection

PrimeTime can set wire loads automatically when you update the timing information for your design. If you do not specify a wire load model for a design or block, PrimeTime automatically selects the models based on the wire load selection group, if specified.

If you do not apply a wire load model or selection group but the library defines a `default_wire_load` model, PrimeTime applies the library-defined model to the design. Otherwise, the values for wire resistance, capacitance, length, and area are all 0.

Automatic wire load selection is controlled by selection groups, which map the block sizes of the cells to wire load models. If you specify the `set_wire_load_selection_group` command on the top design, or if the main logic library defines a `default_wire_load_selection_group`, PrimeTime automatically enables wire load selection.

When wire load selection is enabled, the wire load is automatically selected for hierarchical blocks larger than the minimum cell area, based on the cell area of the block.

To set the minimum block size for automatic wire load selection, enter

```
pt_shell> set_wire_load_min_block_size size
```

In this command, `size` is the minimum block size for automatic wire load selection, in library cell area units. The specified size must be greater than or equal to 0.

The `auto_wire_load_selection` environment variable specifies automatic wire load selection. The default setting is `true`, enabling automatic wire load selection if a selection group is associated with the design. To disable automatic wire load selection, enter

```
pt_shell> set auto_wire_load_selection false
```

To remove the wire load selection group setting, use the `remove_wire_load_selection_group` command.

Setting the Wire Load Mode

The current wire load mode setting, which can be set with the `set_wire_load_mode` command, determines the wire load models used at different levels of the design hierarchy. There are three possible mode settings: `top`, `enclosed`, or `segmented`.

If the mode for the top-level design is `top`, the top-level wire load model is used to compute wire capacitance for all nets within the design, at all levels of hierarchy. If the mode for the top-level design is either `enclosed` or `segmented`, wire load models on hierarchical cells are used to calculate wire capacitance, resistance, and area for nets inside these blocks.

If the `enclosed` mode is set, PrimeTime determines net values using the wire load model of the hierarchical cell that fully encloses the net. If the `segmented` mode is set, PrimeTime separately determines net values for segments of the net in each level of hierarchy, and then obtains the total net value from the sum of all segments of the net.

If you do not specify a mode, the `default_wire_load_mode` setting of the main logic library is used. The `enclosed` mode is often the most accurate. However, your ASIC vendor might recommend a specific mode.

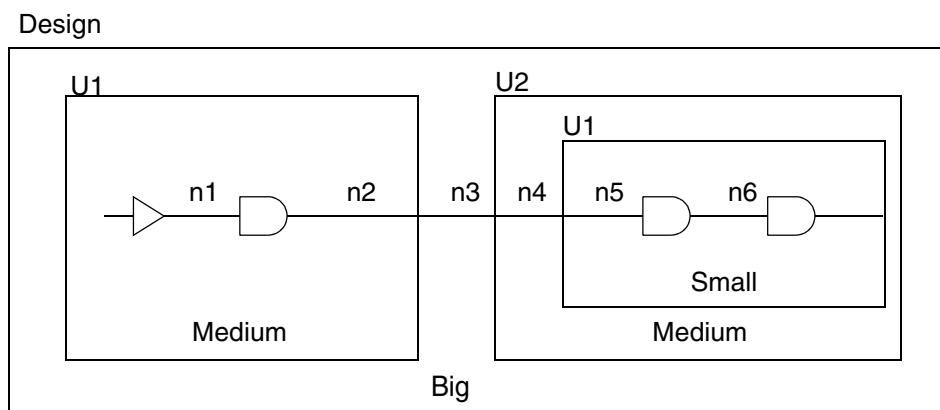
To set the wire load mode to `enclosed` on the current design, enter

```
pt_shell> set_wire_load_mode enclosed
```

The design in [Figure 5-5](#) has the following wire load models set:

```
set_wire_load_model -name Big (the current design)
set_wire_load_model -
  name Medium (instances U1 and U2)
set_wire_load_model -name Small
  (instance U2/U1)
```

Figure 5-5 Design Example for Wire Load Mode Settings



[Table 5-2](#) lists the resulting wire load modes and models that apply to the nets in [Figure 5-5](#).

Table 5-2 Wire Load Models

Wire load setting	Wire load model	Applies to these nets
top	Big	All nets
enclosed	Big	n3, U1/n2, U2/n4, U2/U1/n5
	Medium	U1/n1
	Small	U2/U1/n6
segmented	Big	n3
	Medium	U1/n1, U1/n2, U2/n4
	Small	U2/U1/n5, U2/U1/n6

Reporting Wire Load Models

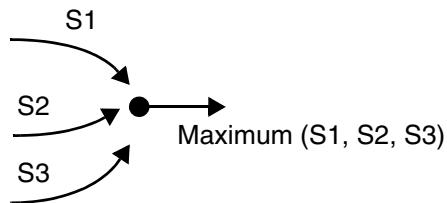
To obtain wire load reports from PrimeTime, use these commands:

- `report_wire_load`
- `report_port -wire_load`

Slew Propagation

At a pin where multiple timing arcs meet or merge, PrimeTime computes the slew per driving arc at the pin, then selects the worst slew value at the pin to propagate along ([Figure 5-6](#)). Note that the slew selected might not be from the input that contributes to the worst path, so the calculated delay from the merge pin could be pessimistic.

Figure 5-6 Maximum Slew Propagation



Minimum slew propagation is similar to maximum slew propagation. In other words, PrimeTime selects minimum slew based on the input with the best delay at the merge point.

Design Rule Constraints

PrimeTime checks for violations of design rule constraints that are defined in the library or by PrimeTime commands. These rules include

- Maximum limit for transition time
- Maximum and minimum limits for capacitance
- Maximum limit for fanout

To report design rule constraint violations in a design, use the `report_constraint` command.

Maximum Transition Time

The maximum transition time in PrimeTime is treated in a similar fashion as the slew. In this topic, slew is first discussed in the context of thresholds and derating, then the discussion is extended to maximum transition time.

You can represent a SPICE waveform as a floating-point number in Liberty tables.

[Figure 5-7](#) shows an example where:

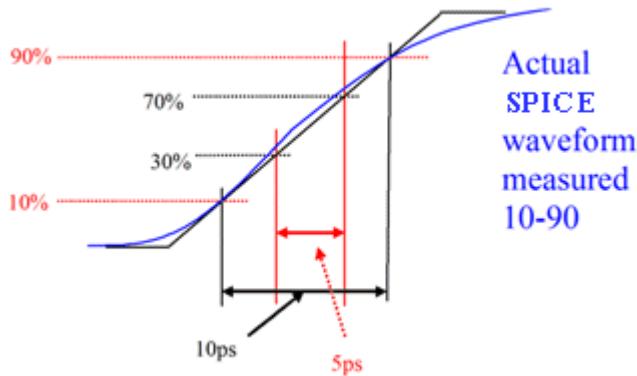
- SPICE waveform is measured 10-90
- SPICE waveform is showing in blue line
- SPICE measured transition time with slew threshold 10-90 is 10 ps
- Linearized waveform measuring transition time with slew threshold 0-100 is 12.5 ps = 10 * (100-0) / (90-10)
- Representation as single float with slew threshold 30-70 is 5 ps = 12.5 * (70-30) / (100-0)

The SPICE waveform can thus be represented as a floating-point number in Liberty tables or in PrimeTime report as follows:

- A: 10-ps slew threshold as 10-90 with slew derating 1.0
- B: 12.5-ps slew threshold as 10-90 with slew derating 0.8
- C: 5-ps slew threshold as 10-90 with derating 2.0

Note that slew threshold in library is always the threshold used for SPICE measurement. Case A is the native representation, measured in Liberty. Cases B and C are rescaled to the slew threshold 0-100 and 30-70, respectively.

Figure 5-7 SPICE Waveform



Handling Slew in Multiple Threshold and Derating Environment

Liberty allows an arbitrary slew threshold to minimize the error due to slew linearization for various process technologies. Therefore whenever slew information from library interacts with another library (or even a pin of the same library with different slew threshold), a conversion to a common base is required.

Converting Single-Float SLEWS Between Thresholds and Deratings

Assume that you have a library L1 with thresholds TL1-TH1, derate SD1, and L2 with TL2-TH2, derate SD2. Note that L1, L2 are just entities that have their own local thresholds, such as library, design, and library pin.

Assume S1 - slew in local thresholds and a derate of L1, and S2 - slew in local thresholds and a derate of L2. The same conversion rule applies if maximum transition is considered instead of slews.

You can then obtain slews expressed in the local thresholds and derate of the other object as follows:

Equivalent slew S2_1, which is slew S2 expressed in the local derate /threshold of L1:

$$S2_1 = S2 * (SD2 / (TH2 - TL2)) * ((TH1 - TL1) / SD1)$$

The meaning of S2_1 is a float number that represents a waveform of slew S2 but measured in the context of L1. The S1 and S2_1 can be directly compared; the S1 and S2 cannot.

Note that in the presence of detailed RC, slew is computed appropriately in the context of threshold and derate.

To learn how maximum transition constraint is handled in the context of thresholds and derate, see [Maximum Transition Constraint Storage](#).

Maximum Transition Constraint Storage

Maximum transition constraints can come from a user input, library, and library pin. User-specified maximum transition constraints are expressed with the main library derate and slew threshold of PrimeTime.

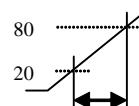
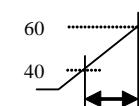
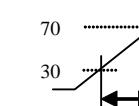
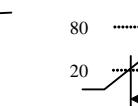
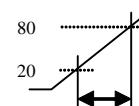
The `set_max_transition` command sets a maximum limit on the transition time for all specified pins, ports, designs, or clocks. When specified on clocks, pins in the clock domain are constrained. Within a clock domain, you can optionally restrict the constraint further to only clock paths or data paths, and to only rising or falling transitions. During constraint checking on a pin or port, the most restrictive constraint specified on a design, pin, port, clock (if the pin or port is in that clock domain), or library is considered. This is also true where multiple clocks launch the same path. The `set_max_transition` command places the `max_transition` attribute, which is a design rule constraint, on the specified objects. In PrimeTime, the slews and maximum transition constraint attributes are reported in the local threshold and derate of each pin or library. For example, the maximum transition time set in the context of the design threshold and derate is scaled to that of the design pin's threshold and derate. The scaling of transition time for slew threshold is on by default.

[Table 5-3](#) describes converting slews between different slew thresholds and derating factors.

*Table 5-3 Converting Slew*s

Object	Main library (L1)	Local library	Library pin	Design	Cell instance pin
Threshold and derate source	Yes	Yes	Yes for slew threshold. Inherits derate of the library	Uses main library derate and slew threshold	The more local trip points have higher precedence than the more general ones
Limit (maximum transition)	Yes	Yes	Yes	Yes - in main library threshold and derate	The most restrictive limit applies
Thresholds	20/80 (rise/fall)	40/60 (rise/fall)	30/70 (rise/fall)	Uses main library threshold	30/70 (rise/fall)

Table 5-3 Converting SLEws (Continued)

Object	Main library (L1)	Local library	Library pin	Design	Cell instance pin
Derate	0.5 (defined)	0.6 (defined)	0.6 (inherited)	0.5 (inherited)	0.6 (inherited)
PrimeTime or Liberty value (slew or limit)	10 ps	10 ps	10 ps	10 ps	10 ps
Linearized SPICE waveform: (rise) Similar for fall					
					
Expressed in derate and threshold of the main library	10 ps	$10 * (0.6 / (0.6 - 0.4)) * ((0.8 - 0.2) / 0.5)$	$10 * (0.6 / (0.7 - 0.3)) * ((0.8 - 0.2) / (0.8 - 0.2) / 0.5)$	$10 * 0.5 / (0.8 - 0.2) * ((0.8 - 0.2) / 0.5)$	$10 * 0.6 / (0.7 - 0.3) * ((0.8 - 0.2) / 0.5)$
PrimeTime or Liberty value (slew or limit)	10 ps	-	-	-	-
Main library limit expressed in local threshold and derate	10 ps	$10 * (0.5 / (0.8 - .02)) * ((.6 -.4) /.6)$	$10 * (0.5 / (0.8 - 0.2)) * ((0.7 - 0.3) / 0.6)$	$10 * (0.5 / ((0.8 - 0.2)) * ((0.8 - 0.2) / 0.5))$	$10 * (0.5 / (0.8 - 0.2)) * ((0.7 - 0.2) / 0.6)$

Evaluating Maximum Transition Constraint

To view the maximum transition constraint evaluations, use the `report_constraint -max_transition` command. PrimeTime reports all constraints and slews in the threshold and derate of the pin of the cell instance, and the violations are sorted based on the absolute values (that is, they are not expressed in that of design threshold and derate). You can also use the `report_constraint` command to report constraint calculations only for maximum

capacitance and maximum transition for a specified port or pin list. Use the `object_list` option to specify a list of pins or ports in the current design that you want to display constraint related information.

To see the port maximum transition limit, use the `report_port -design_rule` command. To see the default maximum transition setting for the current design, use the `report_design` command. To undo maximum transition limits previously set on ports, pins, designs, or clocks, use `remove_max_transition`.

Example 1 - Setting a Maximum Transition Limit

To set a maximum transition limit of 2.0 units on the ports of OUT*, enter

```
pt_shell> set_max_transition 2.0 [get_ports "OUT*"]
```

To set the default maximum transition limit of 5.0 units on the current design, enter

```
pt_shell> set_max_transition 5.0 [current_design]
```

To set the maximum transition limit of 4.0 units on all pins in the CLK1 clock domain, for rising transitions in data paths only, enter

```
pt_shell> set_max_transition 4.0 [get_clocks CLK1] -data_path -rise
```

Example 2 - Scaling the Maximum Transition by Slew Threshold

Consider library lib1 having a slew threshold of 10/90. The design has a maximum transition limit set by you at 0.3 ns. The main library slew threshold for rise and fall is 30/70. By using the `report_constraint -max_transition -all_violators -significant_digits 4` command, you get:

Pin	Required Transition	Actual Transition	Slack
<hr/>			
FF1/D	0.6000	0.4000	0.2000

Example 3 - Scaling the Maximum Transition by Derating

Consider library lib1 having a slew derating factor of 0.5 and a slew threshold for rise and fall of 30/70. You set a maximum transition on the design at 0.3 ns. The main library slew threshold for rise and fall is 30/70 and slew derate is 1.0. By using the `report_constraint -max_transition -all_violators -significant_digits 4` command, you see:

Pin	Required Transition	Actual Transition	Slack
<hr/>			
FF1/D	0.6000	0.4000	0.2000

Minimum and Maximum Net Capacitance

The `set_min_capacitance` command sets a minimum capacitance limit for the specified ports or for the whole design. Similarly, the `set_max_capacitance` command sets a maximum limit on total capacitance for ports or the design. Setting a capacitance limit on a port applies to the net connected to that port. Setting a capacitance limit on a design sets the default capacitance limit for all nets in the design. You have the option to additionally specify max capacitance limit on pins or clocks. When specified on clocks, the pins in the clock domain are constrained. Within a clock domain, you can optionally restrict the constraint further to only clock paths or data paths and only to rising or falling capacitance. The `set_min_capacitance` and `set_max_capacitance` commands place the `min_capacitance` or `max_capacitance` attribute (a design rule constraint) on the specified objects. Capacitance constraint checks are only applicable for output pins. During constraint check, the most restrictive constraint is considered.

To see the capacitance constraint evaluations, use the `report_constraint -min_capacitance` or `-max_capacitance` command. To see port capacitance limits, use the `report_port -design_rule` command. To see the default capacitance settings for the current design, use the `report_design` command. To undo capacitance limits that you set from the UI, use the `remove_min_capacitance` and `remove_max_capacitance` commands.

To set a minimum capacitance limit of 0.2 units on the ports of OUT*, enter

```
pt_shell> set_min_capacitance 0.2 [get_ports "OUT*"]
```

To set the default minimum capacitance limit of 0.1 units on the current design, enter

```
pt_shell> set_min_capacitance 0.1 [current_design]
```

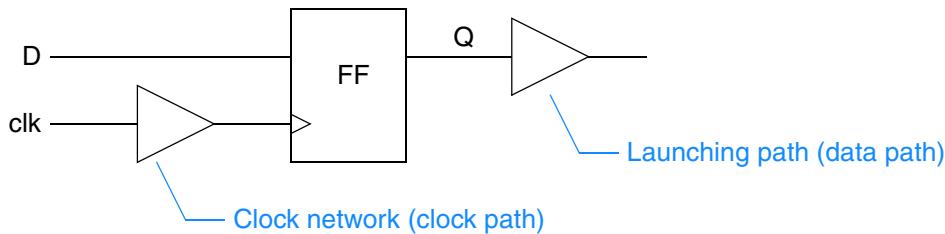
For more information, see the *Synopsys Timing Constraints and Optimization User Guide*.

Constraining Rise and Fall Maximum Capacitance

To constrain rise and fall maximum capacitance at the output of the driver pins in the clock path and data path for clocks in the design, use the `set_max_capacitance` command.

The tool applies maximum capacitance constraints on a clock to all pins in the clock network (clock path) and the pins in the launching path of the clock (data path of the clock).

Optionally, you can restrict constraints on a clock to a clock path, data path, rise, and fall capacitance as shown in [Figure 5-8](#).

Figure 5-8 Restrict Constraints

If you specify a list of clocks without specifying the `-clock_path`, `-data_path`, `-rise`, or `-fall` options, both clock path and data path and both rise and fall are considered by default. You can use the `-clock_path`, `-data_path`, `-rise`, or `-fall` options only if the list of objects is a clock list, not a port list or design. If a pin (port) has multiple constraints from the design, library, clock (and port), the most restrictive constraint is considered. PrimeTime computes the slack at a pin as the difference between the most restrictive constraint and the effective capacitance at a pin (port).

This feature is applicable to generated clocks. To report the constraint violations, use the `report_constraint` command.

Frequency-Based Maximum Capacitance Checks

The PrimeTime SI tool can consider frequency-based `max_capacitance` constraints during design rule constraint (DRC) checking for attribute reporting, the `report_constraint` command, ETM modeling, and ECO fixing. Frequency-based `max_capacitance` values are specified by lookup tables in the library for each driver pin. The pin frequency is the maximum frequency of all clocks that launch signals arriving at the pin.

If you want the `max_capacitance` values from the lookup tables to take precedence over cell-level `max_capacitance` values defined in the library, set the `timing_library_max_cap_from_lookup_table` variable to `true`. [Table 5-4](#) shows how

this variable affects how DRC checking chooses the minimum (most restrictive) `max_capacitance` value.

Table 5-4 Effect of the `timing_library_max_cap_from_lookup_table` Variable

If <code>timing_library_max_cap_from_lookup_table</code> is false (the default)...	If <code>timing_library_max_cap_from_lookup_table</code> is true...
<p>... DRC checking uses the minimum of</p> <ul style="list-style-type: none"> • All user-specified <code>max_capacitance</code> values defined at the pin, port, clock, and design levels • The <code>max_capacitance</code> value from the lookup table • The cell-level <code>max_capacitance</code> value defined in the library 	<p>... DRC checking uses the minimum of</p> <ul style="list-style-type: none"> • All user-specified <code>max_capacitance</code> values defined at the pin, port, clock, and design levels • The <code>max_capacitance</code> value from the lookup table

Maximum Fanout Load

The `set_max_fanout` command sets a maximum fanout load for specified output ports or designs. This command sets the `max_fanout` attribute (a design rule constraint) on the specified objects.

Setting a maximum fanout load on a port applies to the net connected to that port. Setting a maximum fanout load on a design sets the default maximum for all nets in the design. In case of conflict between these two values, the more restrictive value applies.

Library cell pins can have a `max_fanout` value specified. PrimeTime uses the more restrictive of the limit you set or the limit specified in the library.

To see maximum fanout constraint evaluations, use the `report_constraint -max_fanout` command. To see port maximum fanout limits, use the `report_port -design_rule` command. To see the default maximum fanout setting for the current design, use the `report_design` command.

To undo maximum fanout limits you set on ports or designs, use the `remove_max_fanout` command.

To set a maximum fanout limit of 2.0 units on the ports IN*, enter the following syntax:

```
pt_shell> set_max_fanout 2.0 [get_ports "IN*"]
```

To set the default maximum fanout limit of 5.0 units on the current design, enter the following syntax:

```
pt_shell> set_max_fanout 5.0 [current_design]
```

Fanout Load Values for Output Ports

The fanout load for a net is the sum of `fanout_load` attributes for the input pins and output ports connected to the net. Output pins can have maximum fanout limits, specified in the library or set through the `set_max_fanout` command. By default, ports are considered to have a fanout load of 0.0. The `set_fanout_load` command specifies the expected fanout load for output ports in the current design.

To set the fanout load on ports matching OUT* to 3.0, enter the following syntax:

```
pt_shell> set_fanout_load 3.0 "OUT*"
```

For more information, see the *Synopsys Timing Constraints and Optimization User Guide*.

Ideal Networks

PrimeTime allows you to create ideal networks, on which no design rule constraints are checked. During the pre-layout design stages, you might prefer to ignore large unoptimized networks with high fanout and capacitance, and focus instead on violations arising from other sources. Using ideal networks reduces runtime because PrimeTime uses “ideal timing” rather than internally calculated timing. In this way, ideal networks are similar to ideal clock networks, but they can also be applied to data networks.

Ideal networks — sets of connected ports, pins, nets, and cells — are exempt from timing updates and design rule constraint fixing. That is, ideal networks ignore `max_capacitance`, `max_fanout`, and `max_transition` design rule checks. When you specify the source of the ideal network, the pins, ports, nets, and cells contained therein are treated as ideal objects. You or ideal propagation must mark ideal objects.

To learn more about ideal networks, see

- [Propagating Ideal Network Properties](#)
- [Using Ideal Networks](#)
- [Using Ideal Latency](#)
- [Using Ideal Transition](#)

Propagating Ideal Network Properties

When you specify the source objects (ports, leaf-level pins) of an ideal network, the nets, cells, and pins in the transitive fanout of the source objects can be treated as ideal.

Propagation of the ideal network property is governed by the following rules:

- Pin is marked as ideal if it is one of the following:

- Pin is specified in the object list of the `set_ideal_network` command
- Driver pin and its cell are ideal
- Load pin attached to an ideal net
- Net is marked as ideal if all its driving pins are ideal.
- Combinational cell is marked as ideal if either all input pins are ideal or it is attached to a constant net and all other input pins are ideal.

Note:

Ideal network propagation can traverse combinational cells, but it stops at sequential cells.

PrimeTime propagates the ideal network during a timing update and propagates again from ideal source objects as necessary to account for changes in the design, for example, ECOs.

Using Ideal Networks

Using the `set_ideal_network` command, specify which networks you want set as ideal. Indicate the sources of the ideal network with the `object_list` argument. For example, the following command sets the ideal network property on the input port P1, which is propagated along the nets, cells, and pins in the transitive fanout of P1:

```
pt_shell> set_ideal_network P1
```

Use the `-no_propagate` option to limit the propagation of the ideal network for only the nets and pins that are electrically connected to the ideal network source. If, for example, you enter

```
pt_shell> set_ideal_network -no_propagate net1
```

only the net, net1, its driver pins, and its load pins are marked as ideal. No further propagation is performed.

If you do not use the `-no_propagate` option with this command, PrimeTime sets the ideal property on the specified list of pins, ports, or nets and propagates this property according to the propagation rules defined in [Propagating Ideal Network Properties](#).

To remove an ideal network, use the `remove_ideal_network` command. To report ideal ports, pins, nets, or cells, use the `report_ideal_network` command, as shown in the following example:

```
pt_shell> report_ideal_network -timing -load_pin -net -cell

*****
Report : ideal_network
      -timing
      -load_pin
      -net
      -cell
...
*****

Source ports      Latency                      Transition
and pins          Rise      Fall                Rise      Fall
                  min      max      min      max    min      max      min      max
-----
middle/p_in       --      --      --      --      --      --      --      --      --
Internal pins     Latency                     Transition
with ideal timing Rise      Fall                Rise      Fall
                  min      max      min      max    min      max      min      max
-----
n3_i/B            5.62    5.62    --      --      --      --      --      --
n0_i/Z            1.34    --      1.34    --      --      --      --      --
Boundary pins    Latency                     Transition
                  Rise      Fall                Rise      Fall
                  min      max      min      max    min      max      min      max
-----
middle/p_out      --      --      --      --      --      --      --      --
Nets
-----
p_in
p_out
n2
n1
n0
Cells
-----
n3_i
n2_i
n1_i
n0_i
```

Using Ideal Latency

By default, the delay of an ideal network is zero. You can specify ideal latency on pins and ports in an ideal network by using the `set_ideal_latency` command. Ideal latency is accumulated along a path in the same way as a delay value.

Note that ideal latency takes effect only if the object is ideal. If it is not in an ideal network, PrimeTime issues a warning to that effect in the `report_ideal_network` command.

Remove latency from a pin or port by using the `remove_ideal_latency` command.

Using Ideal Transition

The transition time of an ideal network is zero by default. You can, however, specify an ideal transition time value with the `set_ideal_transition` command.

If you set an ideal transition value on an object, the value is propagated from that object along the ideal network either to the network boundary pins or until another ideal transition value is encountered.

Ideal transitions you have annotated on pins or ports take effect only if the object is ideal. If the object is not ideal, PrimeTime issues a warning in the `report_ideal_network` command informing you of this. Use the `remove_ideal_transition` command to remove the transition time you set on a port or pin.

6

Clocks

An essential part of timing analysis is accurately specifying clocks and clock effects, such as latency (delay from the clock source) and uncertainty (amount of skew or variation in the arrival of clock edges).

To learn how to specify, report, and analyze clocks, see

- [Clock Overview](#)
- [Specifying Clocks](#)
- [Specifying Clock Characteristics](#)
- [Using Multiple Clocks](#)
- [Clock Sense](#)
- [Using Pulse Clocks](#)
- [Timing PLL-Based Designs](#)
- [Specifying Clock-Gating Setup and Hold Checks](#)
- [Specifying Internally Generated Clocks](#)
- [Generated Clock Edge Specific Source Latency Propagation](#)
- [Clock Mesh Analysis](#)

Clock Overview

PrimeTime supports the following types of clock information:

Multiple clocks

You can define multiple clocks that have different waveforms and frequencies. Clocks can have real sources in the design (ports and pins) or can be virtual. A virtual clock has no real source in the design itself.

Clock network delay and skew

You specify the delay of the clock network relative to the source (clock latency) and the variation of arrival times of the clock at the destination points in the clock network (clock skew). For multiclock designs, you can specify interclock skew. You can specify an ideal delay of the clock network for analysis before clock tree generation, or you can specify that the delay needs to be computed by PrimeTime for analysis after clock tree generation. PrimeTime also supports checking the minimum pulse width along a clock network.

Gated clocks

You can analyze a design that has gated clocks. A gated clock is a clock signal under the control of gating logic (other than simple buffering or inverting a clock signal). PrimeTime performs both setup and hold checks on the gating signal.

Generated clocks

You can analyze a design that has generated clocks. A generated clock is a clock signal generated from another clock signal by a circuit within the design itself, such as a clock divider.

Clock transition times

You can specify the transition times of clock signals at register clock pins. The transition time is the amount of time it takes for the signal to change from one logic state to another.

Specifying Clocks

You need to specify all clocks used in the design. The clock information includes:

- Period and waveform
- Latency (insertion delay)
- Uncertainty (skew)
- Divided and internally generated clocks

- Clock-gating checks
- Fixed transition time for incomplete clock networks

PrimeTime supports analysis of the synchronous portion of a design. PrimeTime analyzes paths between registers or ports. For a design with multiple interacting clocks, PrimeTime determines phase relationship between the startpoint and endpoint clocks. The clocks can be single phase, multiple phase, or multiple frequency clocks.

Creating Clocks

You must specify all of the clocks in the design by using the `create_clock` command. This command creates a clock at the specified source. A source can be defined at an input port of the design or an internal pin. PrimeTime traces the clock network automatically so the clock reaches all registers in the transitive fanout of its source.

A clock you create with the `create_clock` command has an ideal waveform that ignores the delay effects of the clock network. After you create the clock, you must describe the clock network to perform accurate timing analysis. See [Specifying Clock Characteristics](#).

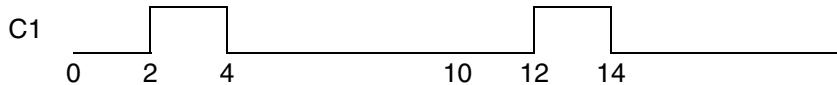
The `create_clock` command creates a path group having the same name as the clock. This group contains all paths ending at points clocked by this clock. For information about path groups, see [Path Groups](#).

To create a clock on ports C1 and CK2 with a period of 10, a rising edge at 2, and a falling edge at 4, enter

```
pt_shell> create_clock -period 10 -waveform {2 4} {C1 CK2}
```

The resulting clock is named C1 to correspond with the first source listed. C1 produces the waveform shown in [Figure 6-1](#).

Figure 6-1 C1 Clock Waveform



PrimeTime supports analyzing multiple clocks propagated to a single register. You can define multiple clocks on the same port or pin by using the `-add` option of the `create_clock` command.

Creating a Virtual Clock

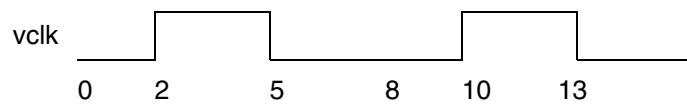
You can use the `create_clock` command to define virtual clocks for signals that interface to external (off-chip) clocked devices. A virtual clock has no actual source in the current design, but you can use it for setting input or output delays.

To create a virtual clock named `vclk`, enter

```
pt_shell> create_clock -period 8 -name vclk -waveform {2 5}
```

The `vclk` clock has the waveform shown in [Figure 6-2](#).

Figure 6-2 vclk Clock Waveform



Selecting Clock Objects

The `get_clocks` command selects clocks for a command to use, for example, to ensure that a command works on the CLK clock and not on the CLK port.

To report the attributes of clocks having names starting with `PHI1` and a period less than or equal to 5.0, enter

```
pt_shell> report_clock [get_clocks -filter "period <= 5.0" PHI1*]
```

Applying Commands to All Clocks

The `all_clocks` command is equivalent to the PrimeTime `get_clocks *` command. The `all_clocks` command returns a token representing a collection of clock objects. The actual clock names are not printed.

To disable time borrowing for all clocks, enter

```
pt_shell> set_max_time_borrow 0 [all_clocks]
```

Removing Clock Objects

To remove a list of clock objects or clocks, use the `remove_clock` command.

Note:

The `reset_design` command removes clocks as well as other information.

To remove all clocks with names that start with CLKB, enter

```
pt_shell> remove_clock [get_clocks CLKB*]
```

To remove all clocks, enter

```
pt_shell> remove_clock -all
```

Specifying Clock Characteristics

Clocks you create with the `create_clock` command have perfect waveforms that ignore the delay effects of the clock network. For accurate timing analysis, you must describe the clock network. The main characteristics of a clock network are latency and uncertainty.

Latency consists of clock source latency and clock network latency. Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design. Clock network latency is the time a clock signal (rise or fall) takes to propagate from the clock definition point in the design to a register clock pin.

Clock uncertainty is the maximum difference between the arrival of clock signals at registers in one clock domain or between domains. This is also called skew. Because clock uncertainty can have an additional margin built in to tighten setup or hold checks, you can specify different uncertainties for setup and hold checks on the same path.

Setting Clock Latency

PrimeTime provides two methods for representing clock latency. You can either

- Allow PrimeTime to compute latency by propagating the delays along the clock network. This method is very accurate, but it can be used only after clock tree synthesis has been completed.
- Estimate and specify explicitly the latency of each clock. You can specify this latency on individual ports or pins. Any register clock pins in the transitive fanout of these objects are affected and override any value set on the clock object. This method is typically used before clock tree synthesis.

Setting Propagated Latency

You can have PrimeTime automatically determine clock latency by propagating delays along the clock network. This process produces highly accurate results after clock tree synthesis and layout, when the cell and net delays along the clock network are all back-annotated or net parasitics have been calculated. The edge times of registers clocked by a propagated

clock are skewed by the path delay from the clock source to the register clock pin. Using propagated latency is appropriate when your design has actual clock trees and annotated delay or parasitics.

To propagate clock network delays and automatically determine latency at each register clock pin, enter

```
pt_shell> set_propagated_clock [get_clocks CLK]
```

You can set the propagated clock attribute on clocks, ports, or pins. When set on a port or pin, it affects all register clock pins in the transitive fanout of the object.

To remove a propagated clock specification on clocks, ports, pins, or cells in the current design, use the `remove_propagated_clock` command.

Specifying Clock Source Latency

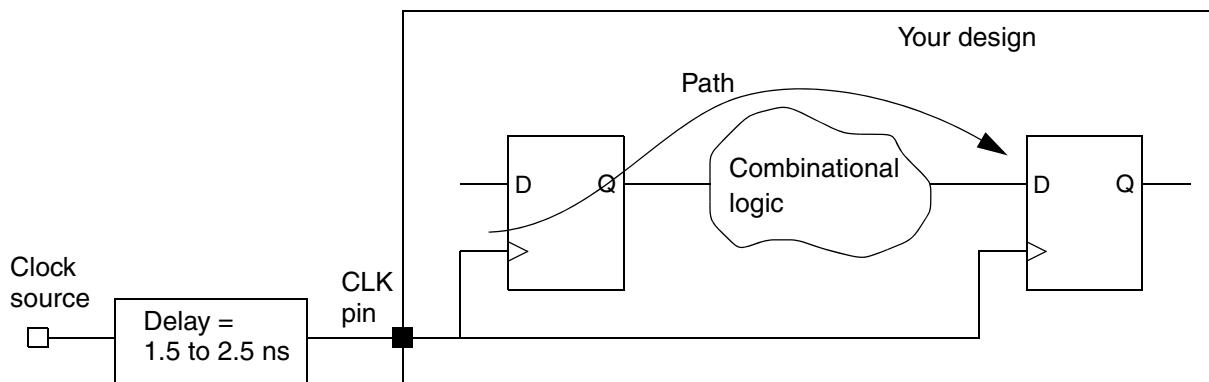
You can specify clock source latency for ideal or propagated clocks. Source latency is the latency from the ideal waveform to the source pin or port. The latency at a register clock pin is the sum of clock source latency and clock network latency.

For internally generated clocks, PrimeTime can automatically compute the clock source latency if the master clock of the generated clock has propagated latency and there is no user-specified value for generated clock source latency.

Propagated latency calculation is usually inaccurate for prelayout design because the parasitics are unknown. For prelayout designs, you can estimate the latency of each clock and directly set that estimation with the `set_clock_latency` command. This method, known as **ideal clocking**, is the default method for representing clock latency in PrimeTime. The `set_clock_latency` command sets the latency for one or more clocks, ports, or pins.

To specify an external uncertainty for source latency, use the `-early` and `-late` options of the `set_clock_latency` command. For example, consider a source latency that can vary from 1.5 to 2.5 ns, as shown in [Figure 6-3](#).

Figure 6-3 External Source Latency

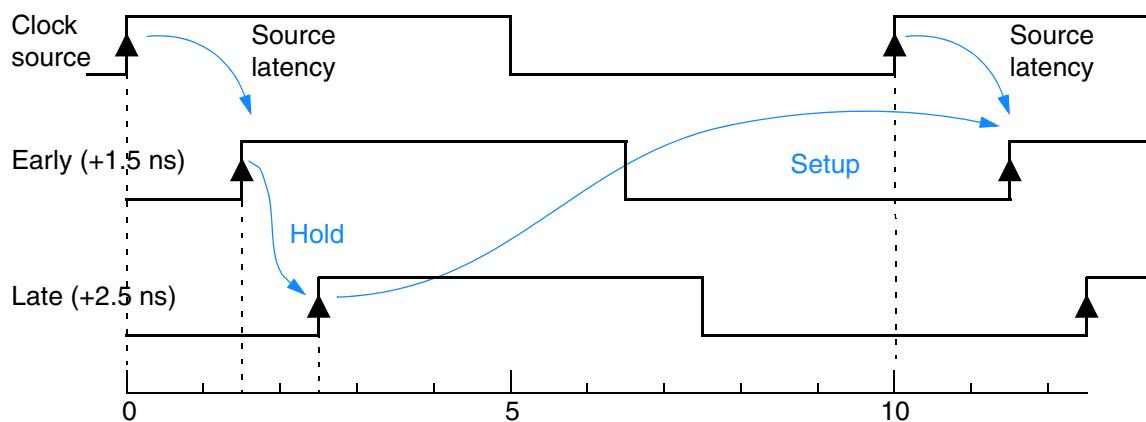


To specify this type of source latency, you can use commands such as the following:

```
pt_shell> create_clock -period 10 [get_ports CLK]
pt_shell> set_clock_latency 1.5 -source -early [get_clocks CLK]
pt_shell> set_clock_latency 2.5 -source -late [get_clocks CLK]
```

PrimeTime uses the more conservative source latency value (either early or late) for each startpoint and endpoint clocked by that clock. For setup analysis, it uses the late value for each startpoint and the early value for each endpoint. For hold analysis, it uses the early value for each startpoint and the late value for each endpoint. [Figure 6-4](#) shows the early and late timing waveforms and the clock edges used for setup and hold analysis in the case where the startpoint and endpoint are clocked by the same clock.

Figure 6-4 Early/Late Source Latency Waveforms



The following examples demonstrate how to set different source latency values for rising and falling edges.

To set the expected rise latency to 1.2 and the fall latency to 0.9 for CLK, enter

```
pt_shell> set_clock_latency -rise 1.2 [get_clocks CLK]
pt_shell> set_clock_latency -fall 0.9 [get_clocks CLK]
```

To specify an early rise and fall source latency of 0.8 and a late rise and fall source latency of 0.9 for CLK1, enter

```
pt_shell> set_clock_latency 0.8 -source -early [get_clocks CLK1]
pt_shell> set_clock_latency 0.9 -source -late [get_clocks CLK1]
```

The `remove_clock_latency` command removes user-specified clock network or source clock latency information from specified objects.

Setting Clock Uncertainty

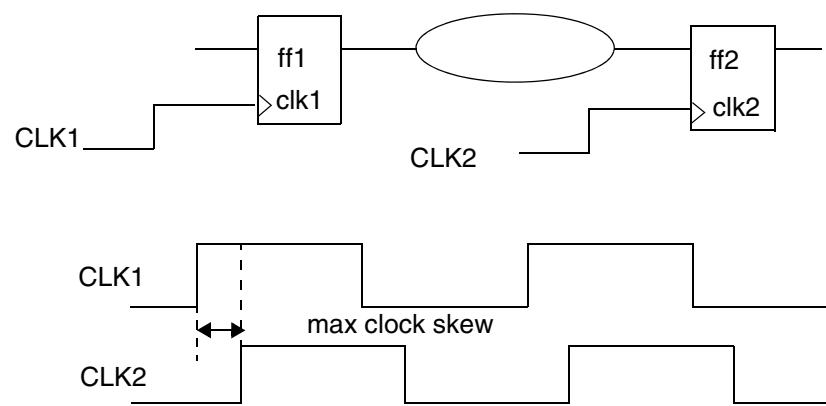
You can model the expected uncertainty (skew) for a prelayout design with setup or hold and rise or fall uncertainty values. PrimeTime subtracts a setup uncertainty value from the data required time when it checks setup time (maximum paths). PrimeTime adds a hold uncertainty value to the data required time when it checks the hold time (minimum paths). If you specify a single uncertainty value, PrimeTime uses it for both setup checks and hold checks.

You can specify the uncertainty or skew characteristics of clocks by using the `set_clock_uncertainty` command. The command specifies the amount of time variation in successive edges of a clock or between edges of different clocks. It captures the actual or predicted clock uncertainty.

You can specify simple clock uncertainty or interclock uncertainty. Simple uncertainty is the variation in the generation of successive edges of a clock with respect to the exact, nominal times. You specify one or more objects, which can be clocks, ports, or pins. The uncertainty value applies to all capturing latches clocked by the specified clock or whose clock pins are in the fanout of the specified ports or pins.

Interclock uncertainty is more specific and flexible, supporting different uncertainties between clock domains. It is the variation in skew between edges of different clocks. You specify a “from” clock using the `-from`, `-rise_from`, or `-fall_from` option and a “to” clock the `-to`, `-rise_to`, or `-fall_to` option. The interclock uncertainty value applies to paths that start at the “from” clock and end at the “to” clock. Interclock uncertainty is relevant when the source and destination registers are clocked by different clocks. You can define uncertainty similarly between two clock pins driven from the same clock, or you can define it as an interclock uncertainty between two registers with different clocks, as shown in [Figure 6-5](#).

Figure 6-5 Example of Interclock Uncertainty



When performing a setup or hold check, PrimeTime adjusts the timing check according to the worst possible difference in clock edge times. For example, for a setup check, it subtracts

the uncertainty value from the data required time, thus requiring the data to arrive sooner by that amount, to account for a late launch and an early capture with the worst clock skew.

When a path has both simple clock uncertainty and interclock uncertainty, the interclock uncertainty value is used, for example

```
pt_shell> set_clock_uncertainty 5 [get_clocks CLKA]
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] \
           -to [get_clocks CLKA]
```

When the path is from CLKB to CLKA, the interclock uncertainty value 2 is used.

The following commands specify interclock uncertainty for all possible interactions of clock domains. If you have paths from CLKA to CLKB, and CLKB to CLKA, you must specify the uncertainty for both directions, even if the value is the same. For example,

```
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKA] \
           -to [get_clocks CLKB]
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] \
           -to [get_clocks CLKA]
```

To set simple clock uncertainty (setup and hold) for all paths leading to endpoints clocked by U1/FF*/CP, enter

```
pt_shell> set_clock_uncertainty 0.45 [get_pins U1/FF*/CP]
```

To set a simple setup uncertainty of 0.21 and a hold uncertainty of 0.33 for all paths leading to endpoints clocked by CLK1, enter

```
pt_shell> set_clock_uncertainty -setup 0.21 [get_clocks CLK1]
pt_shell> set_clock_uncertainty -hold 0.33 [get_clocks CLK1]
```

To remove clock uncertainty information from clocks, ports, pins, or cells, or between specified clocks, use the `remove_clock_uncertainty` command removes uncertainty.

Dynamic Effects of Clock Latency

Dynamic effects on the clock source latency, such as phase-locked loop (PLL) clock jitter can be modeled using the `-dynamic` option of the `set_clock_latency` command. This option allows you to specify a dynamic component of the clock source latency. Clock reconvergence pessimism removal (CRPR) handles the dynamic component of clock latency in the same way as it handles the PrimeTime SI delta delays. For more information about CRPR, see [CRPR With Dynamic Clock Arrivals](#).

You can model clock jitter using the `set_clock_uncertainty` command. However, the clock uncertainty settings do not affect the calculation of crosstalk arrival windows and are not considered by CRPR. The `set_clock_latency` command allows you to specify clock jitter as dynamic source clock latency. The clock jitter specified in this manner properly affects the calculation of arrival windows. The static and dynamic portions are also correctly handled by CRPR.

For example, to specify an early source latency of 3.0 and a late source latency of 5.0 for clock CLK:

```
pt_shell> set_clock_latency -early -source 3.0 [get_ports CLK]
pt_shell> set_clock_latency -late -source 5.0 [get_ports CLK]
```

To model external dynamic effects, such as jitter in the clock source, you can adjust the early and late source latency values. For example, to add plus or minus 0.5 more time units of dynamic latency to the static latency:

```
pt_shell> set_clock_latency -early -source 2.5 \
           -dynamic -0.5 [get_clocks CLK]

pt_shell> set_clock_latency -source -late 5.5 \
           -dynamic 0.5 [get_clocks CLK]
```

The first of these two commands specifies a total early source latency of 2.5, consisting of static source latency of 3.0 and dynamic source latency of -0.5. The second command specifies a total late source latency of 5.5, consisting of static source latency of 5.0 and dynamic source latency of 0.5.

If dynamic latency has been specified as in the foregoing example and the clock named CLK has been set to use propagated latency, the total latency (static and dynamic) is used for delay calculations. However, for a timing path that uses different clock edges for launch and capture, CRPR uses only the static latency, not the dynamic latency, leading up to the common point in the clock path, and removes the resulting static pessimism from the timing results.

The `report_timing` command reports the static and dynamic component of clock latency separately. The `report_crpr` command reports the clock reconvergence pessimism (CRP) time values calculated for both static and dynamic conditions, and the choice from among those values actually used for pessimism removal. For more information about the `report_crpr` command, see [Reporting CRPR Calculations](#).

Estimating Clock Pin Transition Time

Transition times are typically computed for the ports, cells, and nets in the clock network. If the clock network is not complete, the result can be inaccurate transition times on register clock pins. For example, a single buffer might be driving 10,000 register clock pins because the clock tree has not been constructed. This can cause a long transition time on the register clock pins affecting clock-to-output delays, as well as setup and hold delay calculation.

You can estimate the clock transition time for an entire clock network using the `set_clock_transition` command. To specify a nonzero transition time for an ideal clock, use the `set_clock_transition` command. For example,

```
pt_shell> set_clock_transition 0.64 -fall [get_clocks CLK1]
```

The transition time value applies to all nets directly feeding sequential elements clocked by the specified clock.

Use the `-rise` or `-fall` option of the command to specify a separate transition time for only rising or only falling edges of the clock. Use the `-min` or `-max` option to specify the transition time for minimum operating conditions or maximum operating conditions.

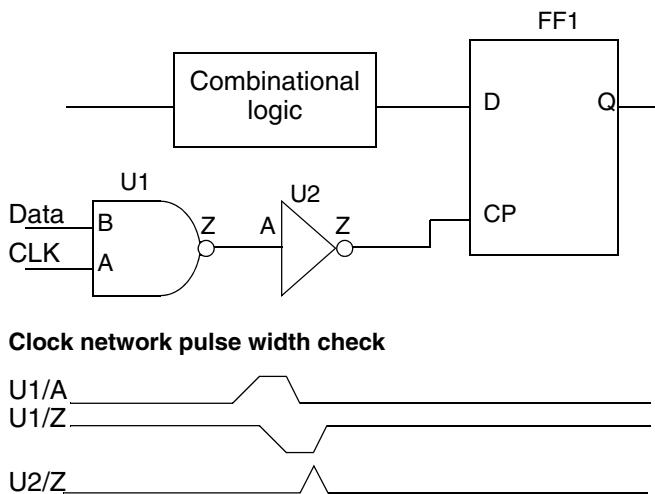
To remove the estimated clock pin transition time, use the `remove_clock_transition` command.

Checking the Minimum Pulse Width

Minimum pulse width checks are important to ensure proper operation of sequential circuits. The pulse width of the original clock might be reduced because of gating logic or delay characteristics of the clock network. Two problems can result:

- If the clock pulse is too small at a register clock pin, the device might not capture data properly. Library cell pins might have a minimum pulse width limit, which is checked by the `report_constraint` command.
- The pulse width might shrink so much at some point in the clock network that it is not propagated further, as shown in [Figure 6-6](#). PrimeTime can check the entire clock network to ensure that the pulse does not shrink below a certain threshold.

Figure 6-6 Clock Network Pulse Width Check



No default is assumed for clock tree pulse width checks. To specify the constraint for normal non-pulse generator for clocks, cells, pins, ports, and the current design, use the `set_min_pulse_width` command. To constrain the pulse generator networks, use the `set_pulse_clock_min_width` and `set_pulse_clock_max_width` commands.

Note:

The `report_constraint` command checks minimum pulse width for register clock pins and for clock networks.

Note:

You can also specify minimum pulse width on clock pins in the library cell description.

To remove minimum pulse width checks for clock signals in a clock tree or at sequential devices, use the `remove_min_pulse_width` command.

Checking the Minimum Period

To perform minimum period checks, use these commands:

- `report_constraint [-min_period]`
- `report_analysis_coverage [-check_type min_period]`

By default, minimum period checking considers the propagated clock arrival time. The following equation shows how the tool calculates the period of a given clock point by using either the rising or falling edge of the propagated clock, depending on which is larger:

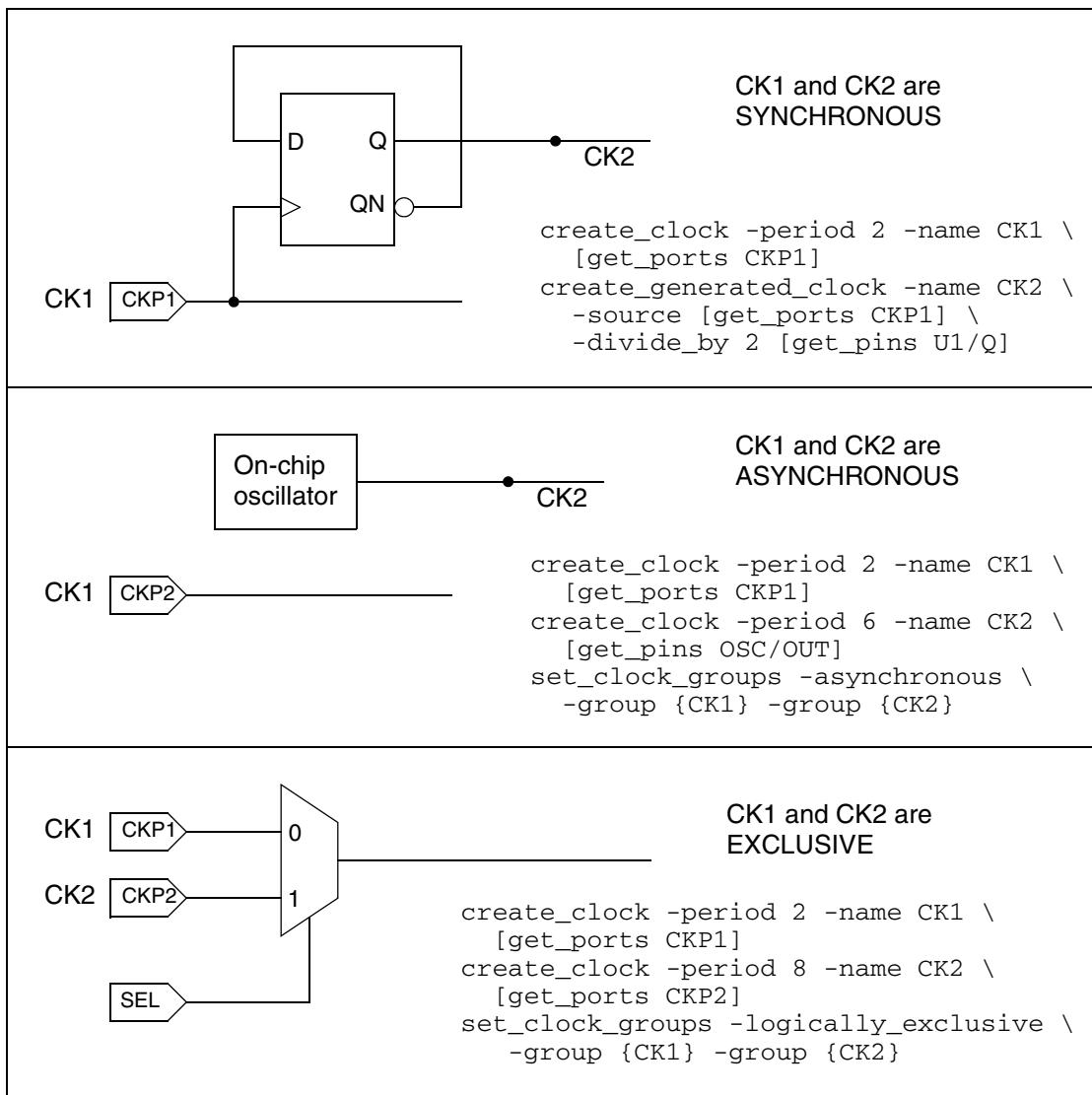
$$\text{calculated period} = \text{period} -$$

$$\max(\text{rising edge propagated clock arrival}, \text{falling edge propagated clock arrival})$$

Using Multiple Clocks

When multiple clocks are defined for a design, the relationships between the clock domains depend on how the clocks are generated and how they are used. The relationship between two clocks can be synchronous, asynchronous, or exclusive, as shown by the examples in [Figure 6-7](#).

For PrimeTime to analyze paths between different clock domains correctly, you might need to specify false paths between clocks, exclude one or more clocks from consideration during the analysis, or specify the nature of the relationships between different clocks.

Figure 6-7 Synchronous, Asynchronous, and Exclusive Clocks

Synchronous Clocks

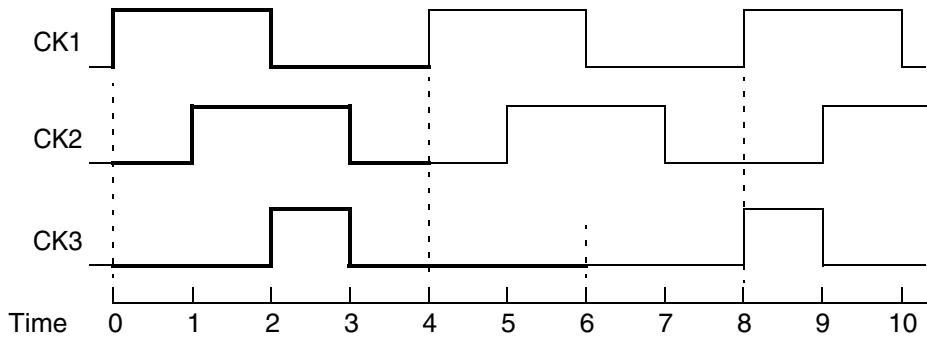
Two clocks are synchronous with respect to each other if they share a common source and have a fixed phase relationship. Unless you specify otherwise, PrimeTime assumes that two clocks are synchronous if there is any path with data launched by one clock and captured by

the other clock. The clock waveforms are synchronized at time zero, as defined by the `create_clock` command. For example, consider the following `create_clock` commands:

```
pt_shell> create_clock -period 4 -name CK1 -waveform {0 2}
pt_shell> create_clock -period 4 -name CK2 -waveform {1 3}
pt_shell> create_clock -period 6 -name CK3 -waveform {2 3}
```

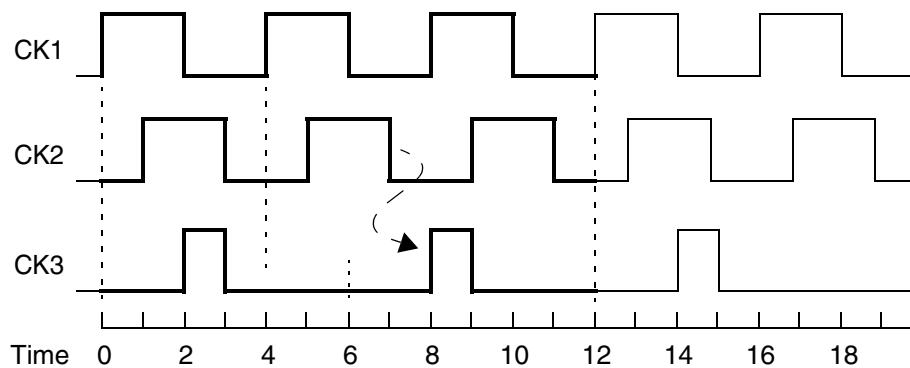
PrimeTime creates the clocks as specified in the commands, with the waveforms synchronized as shown in [Figure 6-8](#). PrimeTime adjusts the timing relationships further for any specified or calculated latency or uncertainty.

Figure 6-8 Synchronous Clock Waveforms



In a design that uses these three clocks, there might be paths launched by one clock and captured by another clock. When such paths exist, to test all possible timing relationships between different clock edges, PrimeTime internally “expands” the clocks to the least common multiple of all synchronous clock periods, thus creating longer-period clocks with multiple rising and falling edges. For example, the three clocks in the foregoing example have periods of 4, 4, and 6. The least common multiple of these periods, called the base period, is 12. To analyze the paths that cross the clock domains, PrimeTime internally expands the clocks by repeating them over the base period. The resulting clock waveforms are shown in [Figure 6-9](#). Each expanded clock has a period of 12.

Figure 6-9 Expanded Clock Waveforms



PrimeTime checks timing paths between all edges in the expanded clocks. For example, the most restrictive setup check between a falling edge of CK2 and a rising edge of CK3 is from time=7 to time=8, as shown by the dashed arrow in [Figure 6-9](#).

Define multiple clocks in a manner consistent with the way they actually operate. To declare clocks that are not synchronous, you can use case analysis or commands, such as the `set_clock_groups -logically_exclusive` or `set_false_path` command.

Asynchronous Clocks

Two clocks are asynchronous if they do not communicate with each other in the design. For example, a free-running, on-chip oscillator is asynchronous with respect to a system clock signal coming into the chip from the outside. Clock edges in the two clock domains can occur at any time with respect to each other.

You can declare a relationship between two clocks to be asynchronous. In that case, PrimeTime does not check the timing paths launched by one clock and captured by the other clock, which is like declaring a false path between the two clocks. In addition, if you are doing crosstalk analysis, PrimeTime SI assigns infinite arrival windows to the nets in aggressor-victim relationships between the two clock domains.

To declare an asynchronous relationship between two clocks, use the `set_clock_groups -asynchronous` command.

Exclusive Clocks

Two clocks are exclusive if they do not interact with each other. For example, a circuit might multiplex two different clock signals onto a clock line, one a fast clock for normal operation and the other a slow clock for low-power operation. Only one of the two clocks is enabled at any given time, so there is no interaction between the two clocks.

To prevent PrimeTime from spending time checking the interaction between exclusive clocks, you can declare a false path between the clocks or use the `set_clock_groups -logically_exclusive` command to declare the clocks to be exclusive. Otherwise, you can use case analysis to disable the clock that you do not want to include in the current analysis.

To declare clocks CK1 and CK2 to be exclusive:

```
pt_shell> set_clock_groups -logically_exclusive \
           -group {CK1} -group {CK2}
```

This causes PrimeTime to ignore any timing path that starts from the CK1 domain and ends at the CK2 domain, or from the CK2 to the CK1 domain. This is like setting a false path from CK1 to CK2 and from CK2 to CK1. However, this setting is not reported by the `report_exceptions` command. To find out about clock groups that have been set, use the

`report_clock -groups` command. If desired, you can also use the `report_clock -groups clock_list` command to restrict the clock groups report to specific clocks of interest.

Avoid setting false paths between clock domains that have been declared to be exclusive because doing so is redundant. The `set_clock_groups -logically_exclusive` command invalidates all false paths set between the exclusive clock domains. This is also true for the `set_clock_groups -asynchronous` command.

You can specify multiple clocks in each group. For example, to declare clocks CK1 and CK2 to be exclusive with respect to CK3 and CK4:

```
pt_shell> set_clock_groups -logically_exclusive \
           -group {CK1 CK2} -group {CK3 CK4}
```

This causes PrimeTime to ignore any path that starts in one group and ends in the other group.

If you specify more than two groups, each group is exclusive with respect to the other specified groups. For example,

```
pt_shell> set_clock_groups -logically_exclusive \
           -group {CK1 CK2} -group {CK3 CK4} -group {CK5}
```

If you specify just one group, that group is exclusive with respect to all other clocks. For example,

```
pt_shell> set_clock_groups -logically_exclusive \
           -group {CK1 CK2}
```

You can optionally assign a name to a clock group declaration, which makes it easier to later remove that particular declaration:

```
pt_shell> set_clock_groups -logically_exclusive -name EX1 \
           -group {CK1 CK2} -group {CK3 CK4}
```

Use the `remove_clock_groups` command to remove a clock grouping declaration:

```
pt_shell> remove_clock_groups -logically_exclusive -name EX1
```

To remove all exclusive clock grouping declarations made with the `set_clock_groups` command:

```
pt_shell> remove_clock_groups -logically_exclusive -all
```

The `set_clock_groups -asynchronous` command defines groups of clocks that are asynchronous with respect to each other. Asynchronous clock group assignments are separate from exclusive clock group assignments, even though both types of clock groups are defined with the `set_clock_groups` command. Clock groups can be physically exclusive as well as logically exclusive due to multiplexing of the clock signals or physical separation. There can be no crosstalk between physically exclusive clocks, as well as no

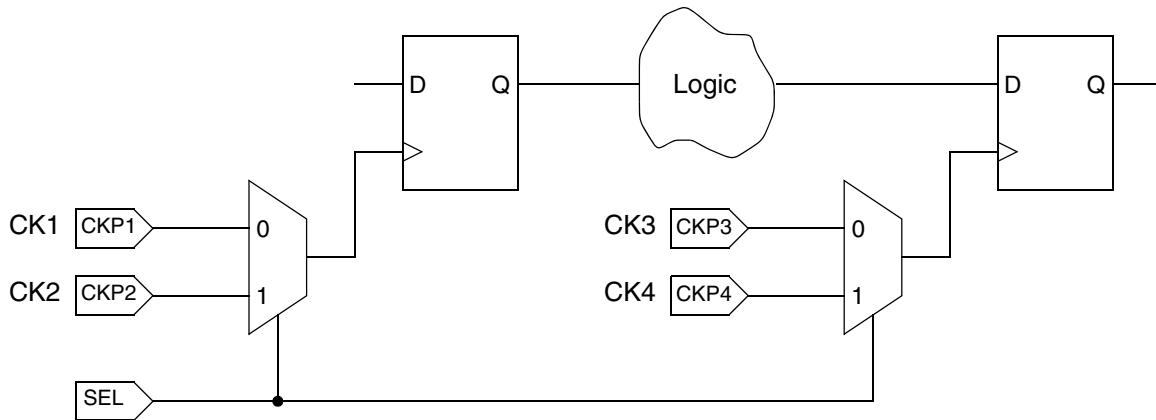
logical interaction. In that situation, use the `-physically_exclusive` option rather than the `-logically_exclusive` option. This prevents the tool from attempting to perform crosstalk analysis between the clock nets. For information about the handling of asynchronous clocks in crosstalk analysis, see [Asynchronous Clocks](#).

The following examples demonstrate some of the ways to specify exclusive clocks.

Example: Four Clocks and One Selection Signal

Consider the circuit shown in [Figure 6-10](#). There are four clocks, CK1 through CK4. By default, PrimeTime analyzes the interactions between all combinations of clocks. However, the logic enables only two clocks at a time, either CK1 and CK3 or CK2 and CK4.

Figure 6-10 Four Clocks and One Selection Signal



One way to prevent checking between unrelated clocks is to set a false path between the clocks. For example,

```

pt_shell> set_false_path -from CK1 -to CK2
pt_shell> set_false_path -from CK2 -to CK1
pt_shell> set_false_path -from CK3 -to CK4
pt_shell> set_false_path -from CK4 -to CK3
pt_shell> set_false_path -from CK1 -to CK4
pt_shell> set_false_path -from CK4 -to CK1
pt_shell> set_false_path -from CK1 -to CK1
pt_shell> set_false_path -from CK2 -to CK3
pt_shell> set_false_path -from CK3 -to CK2
  
```

In that case, PrimeTime tests all of the valid combinations of enabled clocks in a single run, while ignoring the invalid combinations.

Another way is to use case analysis and set a logic value, either 0 or 1, on the SEL input, which checks the timing for a particular case of SEL=0 or SEL=1. For example,

```

pt_shell> set_case_analysis 0 [get_ports SEL]
  
```

With SEL=0, only CK1 and CK3 are active; CK2 and CK4 are ignored. If you want to analyze both cases, two analysis runs are necessary: one with SEL=0 and another with SEL=1.

Another method to accomplish the same effect is to use the `set_disable_timing` command. For example, to disable checking of all paths leading from the CKP2 and CKP4 clock input pins of the design:

```
pt_shell> set_disable_timing [get_ports {CKP2 CKP4}]
```

Still another way is to specify which clocks can be active together at the same time and which clocks are currently active. For example,

```
pt_shell> set_clock_groups -logically_exclusive -name E1 \
           -group {CK1 CK3} -group {CK2 CK4}
pt_shell> set_active_clocks [all_clocks]
```

The `set_clock_groups` command defines groups of clocks that are exclusive with respect to each other. PrimeTime does not check paths that start from a clock in one group and end at a clock in another group. If you specify just one group, that group is considered exclusive with respect to all other clocks.

In the preceding example, the `set_active_clocks` command makes all four clocks active, so that PrimeTime analyzes all valid paths while avoiding the invalid clock combinations.

If you want to consider only the case where SEL=0, you can do it easily by using a different `set_active_clocks` command:

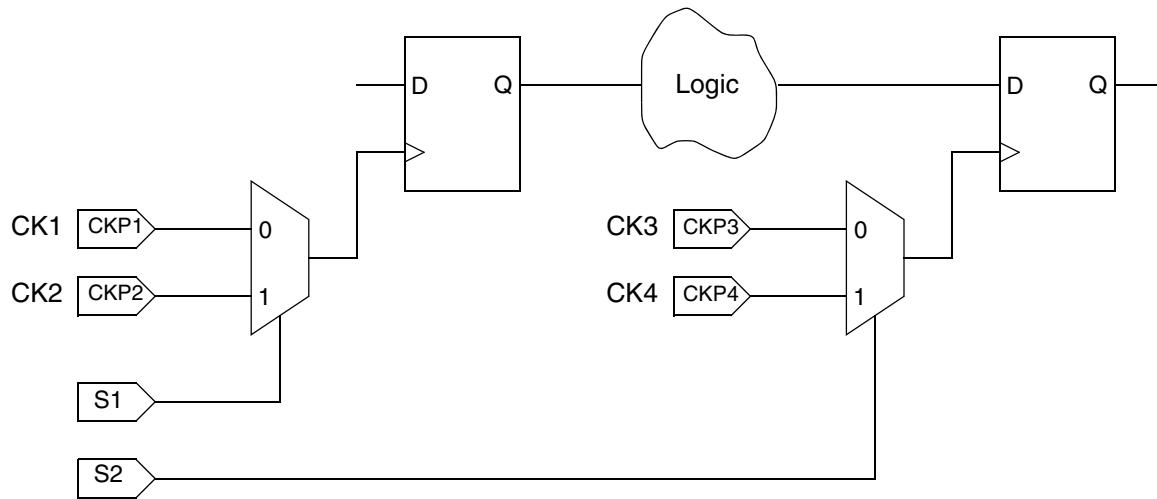
```
pt_shell> set_clock_groups -logically_exclusive -name E1 \
           -group {CK1 CK3} -group {CK2 CK4}
pt_shell> set_active_clocks {CK1,CK3}
```

Setting clocks CK1 and CK3 active means that CK2 and CK4 are inactive, which is just like using case analysis and setting SEL=0 or setting false paths between all combinations of clocks not using CK1 and CK3.

Example: Four Clocks and Two Selection Signals

Consider the circuit shown in [Figure 6-11](#), which is similar to the previous example. There are two separate clock selection inputs, S1 and S2, for the two multiplexers.

Figure 6-11 Four Clocks and Two Selection Signals



Paths between CK1 and CK2 and between CK3 and CK4 are not valid, but all other combinations are possible.

To check all valid paths while avoiding invalid ones, you can declare false paths between the clocks:

```
pt_shell> set_false_path -from CK1 -to CK2
pt_shell> set_false_path -from CK2 -to CK1
pt_shell> set_false_path -from CK3 -to CK4
pt_shell> set_false_path -from CK4 -to CK3
```

Another way is to use case analysis and set logic values on S1 and S2 to check a particular case. For example,

```
pt_shell> set_case_analysis 0 [get_ports S1]
pt_shell> set_case_analysis 0 [get_ports S2]
```

If you want to analyze all four cases using case analysis, four analysis runs are necessary, with S1-S2 = 00, 01, 10, and 11. Still another way is to use the `set_clock_groups` and `set_active_clocks` commands. For example,

```
pt_shell> set_clock_groups -logically_exclusive -name mux1 \
    -group {CK1} -group {CK2}
pt_shell> set_clock_groups -logically_exclusive -name mux2 \
    -group {CK3} -group {CK4}
pt_shell> set_active_clocks {CK1,CK2,CK3,CK4}
```

PrimeTime analyzes all valid paths from CK1 to CK3 and CK4, and from CK2 to CK3 and CK4 (and in the opposite direction if there are any such paths), but not between CK1 and CK2 or between CK3 and CK4.

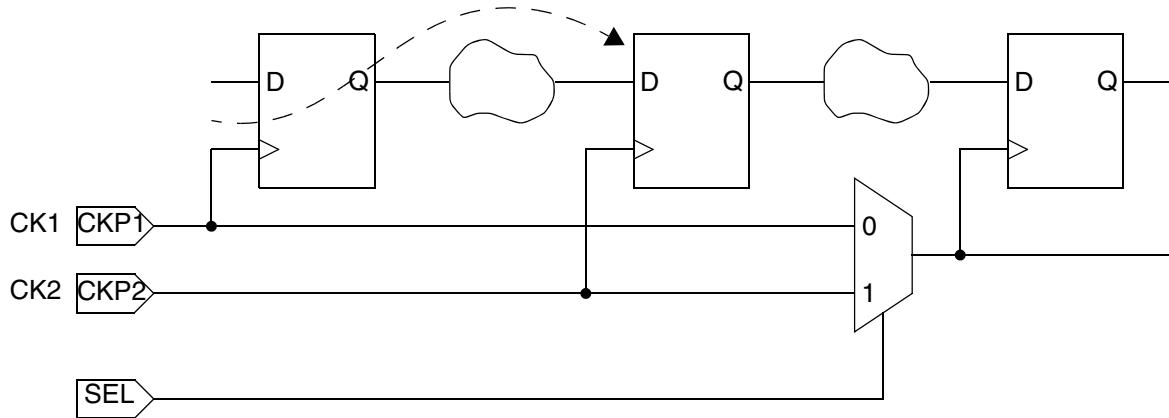
If you only want to consider the case where S1-S2=00, you can do it easily by using a different `set_active_clocks` command:

```
pt_shell> set_clock_groups -logically_exclusive -name mux1 \
    -group {CK1} -group {CK2}
pt_shell> set_clock_groups -logically_exclusive -name mux2 \
    -group {CK3} -group {CK4}
pt_shell> set_active_clocks {CK1,CK3}
```

Example: Multiplexed Clocks and Case Analysis

Consider the circuit shown in [Figure 6-12](#). The clocks CK1 and CK2 are at the startpoint and endpoint of one path. These clocks are also multiplexed onto a shared clock line.

Figure 6-12 Multiplexed Clocks Specified With Case Analysis



If you want PrimeTime to check the valid path between CK1 and CK2, but avoid checking invalid paths between the two clock domains downstream from the multiplexer, the best way to do it is with case analysis.

Removing Clocks From Analysis

By default, PrimeTime analyzes all clocks specified for a design and all interactions between different clock domains. In a design with multiple clocks, it is often desirable to do timing analysis with only certain clocks enabled. For example, a device might use a fast clock for normal operation and a slow clock in power-down mode; you might be interested in the timing behavior for just one operating mode.

One way to specify the active clocks is to use case analysis. You specify a logic value, either 0 or 1, for a port or pin that controls clock selection. For example, if a port called SEL selects the active clock in the device, you could use this command:

```
pt_shell> set_case_analysis 0 [get_ports SEL]
```

In that case, timing analysis is restricted to the case where SEL=0.

Another method is to use the `set_active_clocks` command, which specifies the list of clocks that are active for the analysis. Clocks not listed in the command are inactive and not considered during the analysis. For example,

```
pt_shell> set_active_clocks {CK1,CK2}
```

Clocks CK1 and CK2 are considered for analysis and all others are ignored. If a generated clock is based on an inactive clock, the generated clock is also made inactive. Using the `set_active_clocks` command triggers a full timing update for the design.

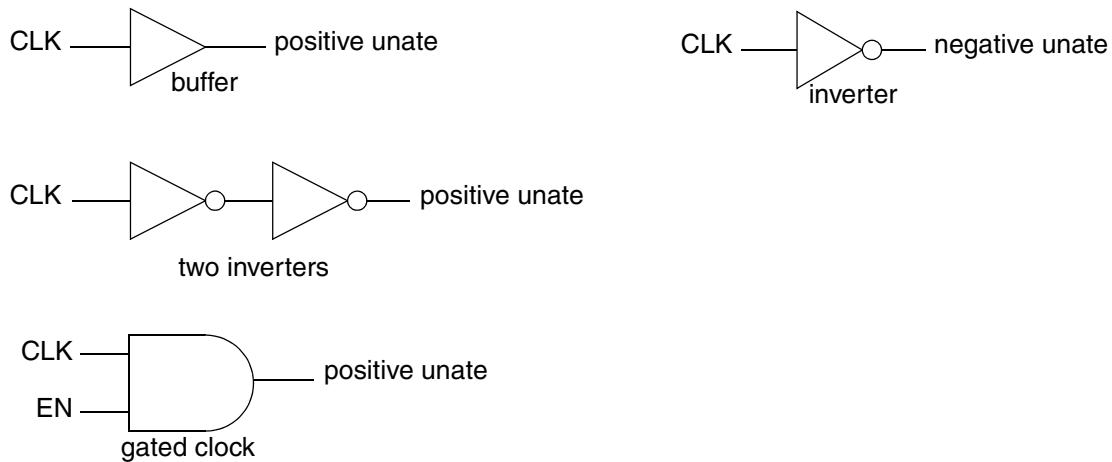
To make all clocks active (the default behavior), use the `set_active_clocks [all_clocks]` command. To choose an entirely new set of active clocks, use the `set_active_clocks` command again; each use of the command overrides the previous settings. The `set_active_clocks` command works for standard timing analysis, but it does not work for context characterization, model extraction, or the `write_sdc` command.

Clock Sense

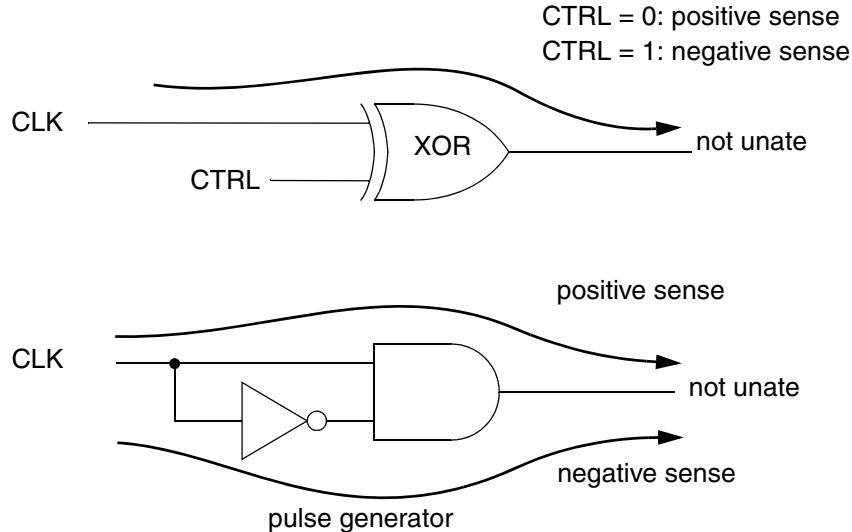
PrimeTime keeps track of inverters and buffers in clock trees. It recognizes the positive or negative sense of the clock signal arriving at each register clock pin. No specific action is necessary to tell PrimeTime the sense of a clock tree that has only buffers and inverters. In this case, the clock signal arriving at the register clock pin is said to be *unate*.

A clock signal is *positive unate* if a rising edge at the clock source can only cause a rising edge at the register clock pin, and a falling edge at the clock source can only cause a falling edge at the register clock pin.

Similarly, a clock signal is *negative unate* if a rising edge at the clock source can only cause a falling edge at the register clock pin, and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted. See [Figure 6-13](#).

Figure 6-13 Unate Clock Signals

A clock signal is not unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate because there are non-unate arcs in the gate. The clock sense could be either positive or negative, depending on the state of the other input to the XOR gate, as shown in

[Figure 6-14.](#)*Figure 6-14 Non-Unate Clock Signals*

PrimeTime considers the output of the pulse generator at the bottom of [Figure 6-14](#) to be non-unate because there are both inverting and non-inverting paths through the logic. The non-inverting path is the direct path through the AND gate and the inverting path is through the inverter.

To specify either a positive or negative clock sense to be propagated forward from a pin within a non-unate part of the clock network, use the `set_sense -type clock` command. This command can affect only the non-unate part of a clock network. If you specify the command for a pin in a unate section of the clock network, and the requested sense disagrees with the actual sense, PrimeTime issues an error message, and the setting is ignored.

To resolve this ambiguity for PrimeTime, you can specify the sense of a clock signal at a point in the clock path using the `set_sense -type clock` command. For example,

```
pt_shell> set_sense -type clock -positive [get_pins xor1.z]
```

This command tells PrimeTime to propagate only the positive unate paths through the output pin of the XOR gate, with respect to the original clock source. From that point onward, PrimeTime keeps track of the sense of the signal through any subsequent buffers or inverters.

The positive unate setting applies to any clock that passes through the specified pin. If multiple clocks can reach that pin, you can restrict the setting to certain clocks, as in the following example:

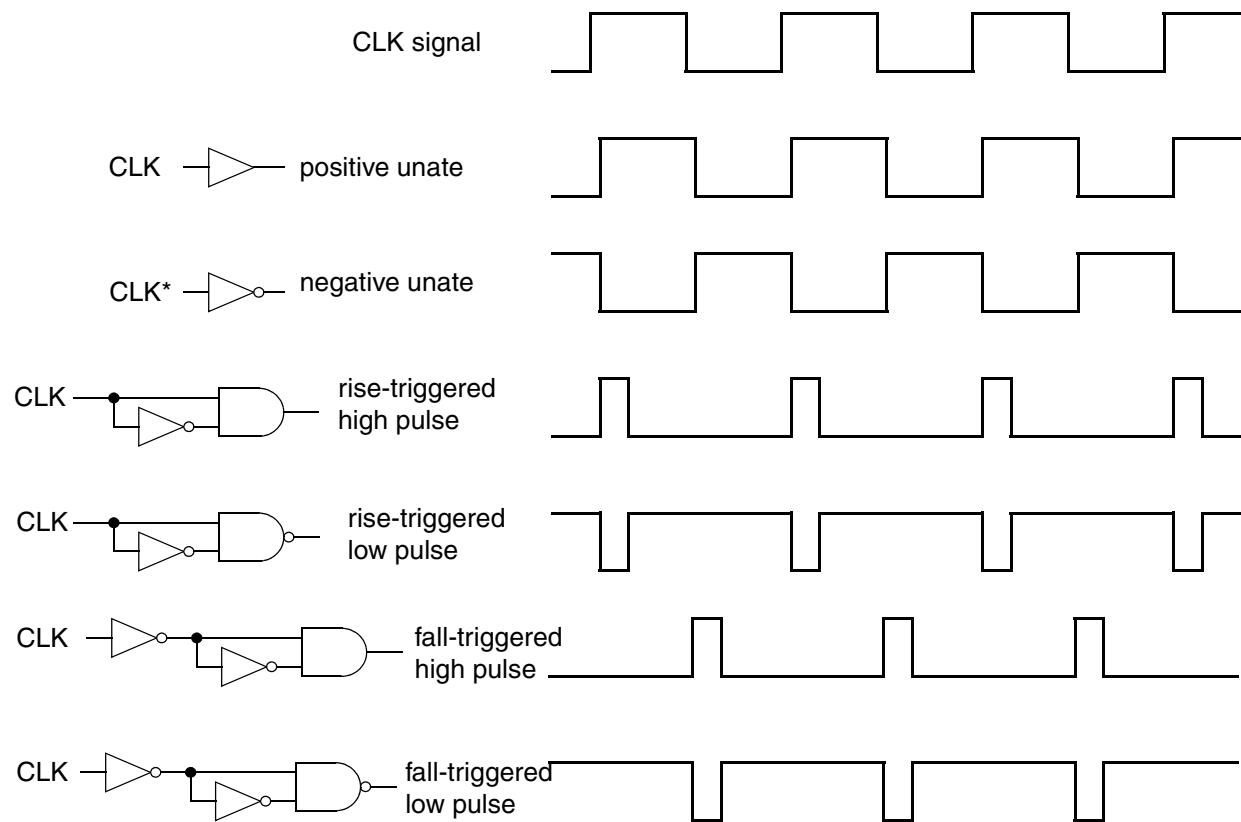
```
pt_shell> set_sense -type clock -positive \
           -clocks [get_clocks CLK] [get_pins mux1.z]
```

The `set_sense -type clock` command has some additional sense types to support pulse clocks. To specifying the clock sense at a point, use one of these commands:

```
set_sense -type clock -positive object_list
set_sense -type clock -negative object_list
```

The object list specifies the pins where the sense is being defined. If multiple clocks pass through the objects, you can specify the clocks affected by the command by using the `-clocks` option. To reverse the effects of `set_sense -type clock`, use the `remove_sense -type clock` command.

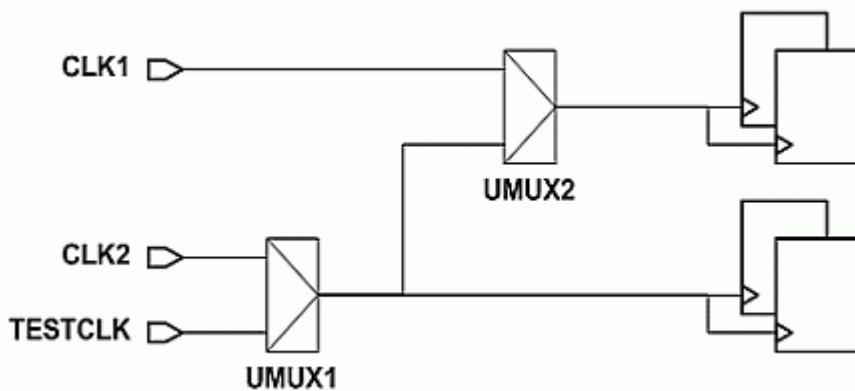
[Figure 6-15](#) shows examples of clock-modifying circuits and the resulting clock senses.

Figure 6-15 Clock Sense Examples

To stop clock propagation from a specified pin or cell timing arc, use the `set_sense -type clock -stop_propagation` command. You could use the `-stop_propagation` option to stop propagation of specified clocks in the clock list from the specified pins or cell timing arcs in the object list. This is appropriate in cases where the clock physically does not propagate past a certain pin as clock propagation is stopped at the pin.

[Figure 6-16](#) shows an example of physical clock stopping. In this example, the control logic is such that UMUX2 is allowed to select CLK2, but it never selects TESTCLK. In this case, TESTCLK never physically exists beyond UMUX2. To model this in PrimeTime, use this command:

```
set_sense -type clock -stop_propagation -clocks TESTCLK UMUX2/Z
```

Figure 6-16 Physical Clock Stopping

Using Pulse Clocks

A pulse clock consists of a sequence of short pulses whose rising and falling edges are both triggered by the same edge of another clock. Pulse clocks are often used to improve performance and reduce power consumption.

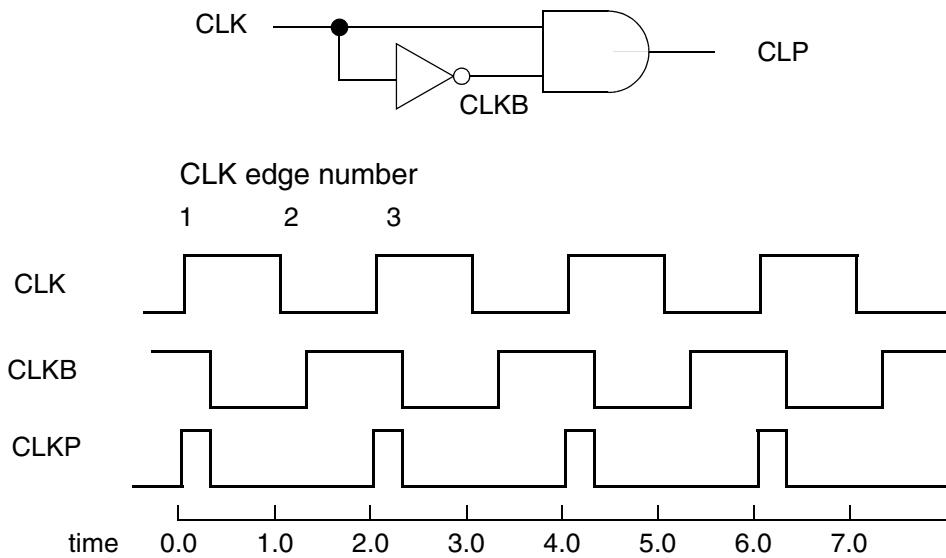
To analyze the timing of a circuit containing pulse clocks, PrimeTime needs information about the timing characteristics of the clock. There are three ways to provide this information:

- Use a pulse generator cell that has been characterized with pulse generator attributes in the .lib description.
- Use the `create_generated_clock` command to describe the pulse timing with respect to the source clock.
- Use the `set_sense -type clock` command to specify the sense of the generated pulses with respect to the source clock.

The best method is to use a pulse generator cell that has been characterized in its .lib library description. In that case, no additional action is necessary in PrimeTime to specify the pulse clock characteristics. For information about specifying the pulse generator characteristics of a library cell, see the Library Compiler documentation.

If characterized pulse generator cells are not available in the library, you must specify the pulse clock characteristics at each pulse generation point by using either the `create_generated_clock` or `set_sense -type clock` command. Using the `create_generated_clock` command creates a new clock domain at a pulse generation point. Using `set_sense -type clock` does not create a new clock domain, but merely specifies the sense for an existing clock downstream from the specified point. For example, consider the pulse clock circuit shown in [Figure 6-17](#). Each rising edge of CLK generates a pulse on CLKP.

Figure 6-17 Pulse Clock Specified as a Generated Clock



The same edge (edge number 1) of the source triggers both the rising and falling edges of the pulse clock. The pulse width is determined by the delay of the inverter.

To specify the generated pulse clock CLKP as a generated clock:

```
pt_shell> create_generated_clock -name CLKP -source CLK \
    -edges {1 1 3} [get_pins and2/z]
```

Specifying the generated clock as a pulse clock using repeated edge digits (rather than specifying the pulse clock edge times) ensures correct checking of delays between the source clock and the pulse clock.

In general, the position of the repeated digit determines whether an active-high or active-low pulse is generated, and the edge number that is repeated determines the type of edge in the master clock used to trigger the pulse:

- -edges {1 1 3} – Rising edge of source triggers high pulse.
- -edges {2 2 4} – Falling edge of source triggers high pulse.
- -edges {1 3 3} – Rising edge of source triggers low pulse.
- -edges {2 4 4} – Falling edge of source triggers low pulse.

Instead of using the `create_generated_clock` command to define a new clock, you can use the `set_sense -type clock` command to specify the sense of the existing clock:

```
pt_shell> set_sense -type clock -pulse rise_triggered_high_pulse \
    [get_pins and2/z]
```

This command tells PrimeTime that the clock at the output of the AND gate is a pulse that rises and falls on the rising edge of the source clock. The pulse clock is not defined as a separate clock domain. Instead, it is just a different sense (rise-triggered high pulse sense) of the source clock downstream from the specified point in the clock network.

In general, the clock sense of a pulse clock can be specified at a location by using one of the following commands:

```
set_sense -type clock -pulse rise_triggered_high_pulse object_list
set_sense -type clock -pulse rise_triggered_low_pulse object_list
set_sense -type clock -pulse fall_triggered_high_pulse object_list
set_sense -type clock -pulse fall_triggered_low_pulse object_list
```

The nominal width of the generated pulses is zero, whether you use a pulse generator cell defined in the library, the `create_generated_clock` command, or the `set_sense -type clock` command. To determine the actual pulse width, PrimeTime considers the different rise and fall latency values at the pulse generator output pin.

$$(\text{high pulse width}) = (\text{fall network latency}) - (\text{rise network latency})$$

$$(\text{low pulse width}) = (\text{rise network latency}) - (\text{fall network latency})$$

You can allow PrimeTime to calculate the propagated latency from the circuit, or you can use the `set_clock_latency` command to specify the latency values (and therefore the pulse width) explicitly. For example, to set an ideal pulse width to 0.5 for high pulses, for all registers downstream from pin and2/z, and with an overall latency of 0.6, the commands are:

```
pt_shell> set_clock_latency -rise 0.6 and2/z
pt_shell> set_clock_latency -fall 1.1 and2/z
```

Constraining Pulse Widths in the Fanout of Pulse Generator Cells

PrimeTime provides the ability to constrain the pulse clock network. The transitive fanout of pulse generator is referred to as pulse clock network. The clock propagating through the pulse generator in the pulse clock network is referred to as the pulse clock. However, if the pulse generator has sequential arcs, its output is not a clock signal unless a generated clock is defined at the output. Use the `set_pulse_clock_min_width` and `set_pulse_clock_max_width` commands to constrain the pulse generator networks.

Use the `set_pulse_clock_min_width` command to constrain the minimum pulse width in the fanout of pulse generator instances by pulse generator instance name or pulse generator library cell. The minimum pulse width constraint from the library also applies to the pulse clock network. You can set minimum pulse width by using the following syntax:

```
set_pulse_clock_min_width -transitive_fanout value object_list
```

Note that if the CRPR is enabled, the clock reconvergence pessimism (CRP) value is applied to the pulse width as a credit. The CRP is subtracted from the pulse width for

maximum pulse width calculation and added to the pulse width for minimum pulse width calculation.

Use the `set_pulse_clock_max_width` command to constrain the maximum pulse width in the fanout of pulse generator instances by pulse generator instance name or pulse generator library cell. You can set maximum pulse width by using the following syntax:

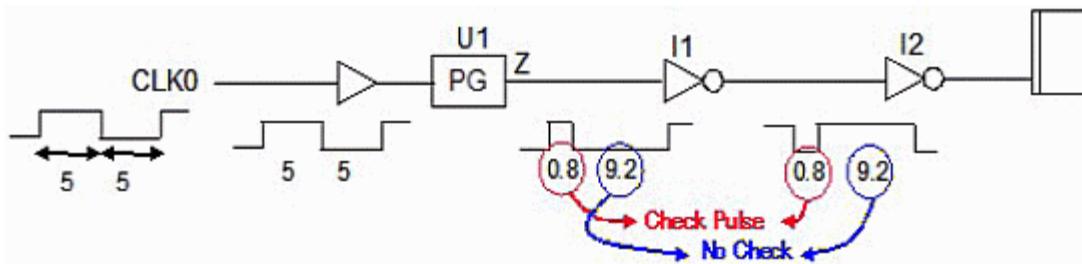
```
set_pulse_clock_max_width -transitive_fanout value object_list
```

For either the maximum or minimum version of the command, if you constrain the pulse width by a clock, the constraint applies to the fanout of all pulse generators driven by that clock. If you constrain it by design, the constraint applies to all pulse clock networks. If there are conflicting constraints, the most restrictive constraint is used.

The pulse width constraint of high or low is inferred based on sense propagation.

[Figure 6-18](#) shows the sense propagation. For example, before the inverter, the minimum pulse width of the high pulse is constrained while after the inverter the minimum pulse width of the low pulse is constrained.

Figure 6-18 Sense Propagation



When removing either the maximum or minimum pulse width constraint you must explicitly specify all of the sources of the constraint. To remove the minimum pulse width constraint from the pulse generator network, you can use the `remove_pulse_clock_min_width` command. To remove the maximum pulse width constraint from the pulse generator network, you can use the `remove_pulse_clock_max_width` command.

To report the pulse width of all pulse generator networks, use one of the following commands:

- `report_pulse_clock_min_width`
- `report_pulse_clock_max_width`
- `report_constraint -pulse_clock_min_width`
- `report_constraint -pulse_clock_max_width`

Pulse clock constraint checking is controlled by the `timing_enable_pulse_clock_constraints` variable. When this variable is set to `true`

(the default), the general minimum pulse width constraints are checked everywhere, except the pulse clock network. More specific pulse clock constraints are used in the pulse clock network. If you set the `timing_enable_pulse_clock_constraints` variable to `false`, the more specific pulse clock constraints are ignored and the general minimum pulse width constraints are used for the design, including pulse clock networks.

Note that during the `report_analysis_coverage` command, this variable is temporarily changed back to `false` to properly obtain design information. An informational message is also printed.

Constraining Transition Times for Pulse Generator Cells

PrimeTime provides support for constraining the minimum transition value at the input of a pulse generator instance. To constrain the minimum transition, use the following syntax:

```
set_pulse_clock_min_transition [-rise | -fall] value object_list
```

You can constrain the minimum transition by pulse generator instance name, pulse generator library cell, clock, or design. If the constraint is applied on a clock, the input of all the pulse generators in that clock network is constrained. If the constraint is applied to a design, all the pulse generator inputs are constrained. You can constrain minimum transition only at the input of pulse generators.

To constrain the maximum transition in the fanout of pulse generator instances, use the `set_pulse_clock_max_transition -transitive_fanout` command.

You can constrain this maximum transition by pulse generator instance name, pulse generator library cell, clock, or design. When the constraints are set on a clock, the fanout of all pulse generators driven by this clock are constrained. When the pulse clock maximum transition constraint is set on the design, all the pulse networks are constrained.

Note:

The maximum transition constraint set on design by using the `set_max_transition` command applies to the pulse network as does the maximum transition constraint specified on the library pins.

For both the maximum and minimum case, if the constraints are conflicting the most restrictive constraint are valid. You can separately constrain rise and fall transitions by using the `-rise` and `-fall` options.

The `set_pulse_clock_max_transition` command also allows you to specify the constraint that you want applied only to the input of pulse generators. The `-transitive_fanout` option specifies the constraint set at the transitive fanout of pulse generator. If this option is not set, only the input of the pulse generators are constrained.

You can remove the constraint from the input of pulse generator cells or from the pulse generator, using, respectively, the `remove_pulse_clock_min_transition` or `remove_pulse_clock_max_transition -transitive_fanout` commands.

Note:

To remove the constraint from the input of pulse generators, do not use the `-transitive_fanout` option.

To report the maximum transition computation, you can use the `report_pulse_clock_max_transition` and `report_constraint_pulse_clock_max_transition` commands. You can report the minimum transition computation at the input of all pulse generator networks by using the `report_pulse_clock_min_transition` and the `report_constraint_pulse_clock_min_transition` commands.

Note:

To report the maximum transition computation at the input of pulse generator cells, do not use the `-transitive_fanout` option.

To enable pulse clock constraint checking, set the `timing_enable_pulse_clock_constraints` variable to true.

Timing PLL-Based Designs

Phase-locked loops (PLL) are common in high-speed designs. Their ability to nearly zero out the delay of a large clock tree allows for much higher speed interchip communication. Certain effects, such as OCV and signal integrity analysis, requires a complete and accurate static timing analysis of the design, including the PLL. The PLL reduces the clock skew at launch and capture flip-flops by making the phase of the clock at the feedback pin the same as the phase at the reference clock.

PrimeTime supports the analysis of PLL cells. The PLL library model contains the information regarding the reference clock pin, output pin, and feedback pin in the form of the attributes on the PLL cell pins. This information is used to perform additional error checking during the definition of the generated clock at the outputs of the PLL. For each PLL, you provide all of the relevant information regarding the reference, feedback, and output pins so that each PLL cell is identified. During the timing update, it automatically computes the timing of the feedback path and applies it as a phase correction on the PLL cell. This approach improves runtime and also simplifies the analysis script. This approach supports:

- Multiple PLLs
- PLLs with multiple output
- PLL jitter and long-term drift
- CRPR calculations for PLL paths

- PrimeTime SI analysis
- Sequential cells in the feedback path
- PLL adjustment during path-based analysis

Usage for PLL Timing

You create each PLL-generated clock by using the `create_generated_clock` command with the `-pll_feedback` and `-pll_output` options. Both these options are required when defining a generated clock at the output of a PLL. The `-pll_feedback` option indicates which PLL feedback pin is used for the clock's phase shift correction. The `-pll_output` option specifies the PLL clock output pin that propagates to the feedback pin. For single-output PLLs, the `-pll_output` pin is the same as the generated clock source pin. For multiple-output PLLs with a single shared feedback path, the `-pll_output` pin can differ from the source pin. When using these options to define a PLL, only certain options are available with the following restrictions applying:

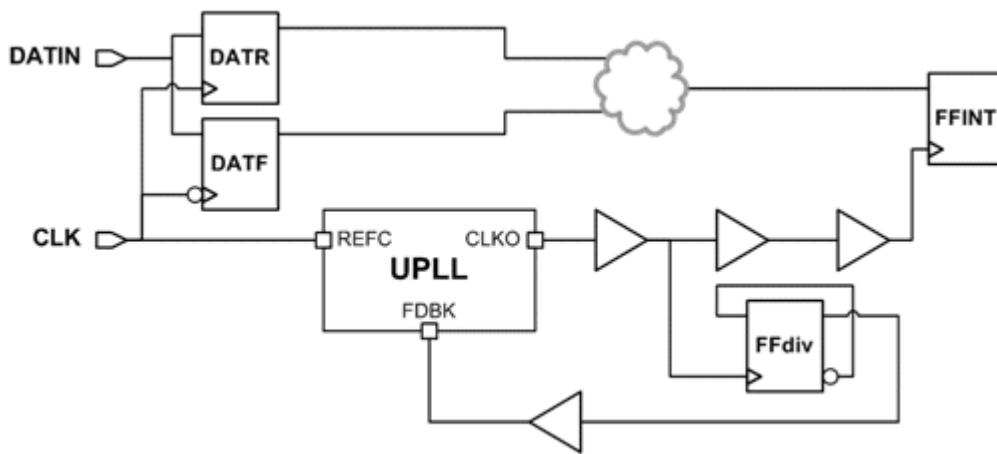
- The `source_objects` option specifies a single pin object that corresponds to the clock output pin of the PLL being described.
- The `master_pin` option corresponds to the reference clock input pin of the PLL.
- The `feedback_pin` option corresponds to the feedback input pin of the PLL.
- The `output_pin` option corresponds to the output pin from which the feedback path starts. The `output_pin` option can be different from the pin in the `source_objects` option if the source pin's phase correction is determined by another clock output pin of the PLL.
- All four pins must belong to the same cell.

For the PLL adjustment to occur correctly, set both the PLL output clock and reference clock arriving at the reference clock pin of the PLL as propagated clocks.

Sequential Cells on a Feedback Path

If the feedback path has any sequential elements on it, you need to define a generated clock on the output of the last sequential element. The master pin of this generated clock can be any pin that satisfies the following two conditions:

- The fanout of the output pin is connected to the feedback pin.
- The pin lies on the feedback path.

Figure 6-19 Example Circuit

In the circuit in [Figure 6-19](#), the PLL output clock at pin CLKO is twice the frequency of the PLL reference clock at pin REFClk. The sequential cell FFdiv acts as a clock divider and converts the frequency of the PLL output clock to that of the reference clock. To correctly define the PLL configuration, you need to define a generated clock at the output clk_out1 of the UPLL, using the following command:

```

create_generated_clock \
    -name CLK_pll \
    -source [get_pins UPLL/REFClk] \
    -pll_output [get_pins UPLL/CLKO] \
    -pll_feedback [get_pins UPLL/FDBK] \
    -multiply_by 2 \
    [get_pins UPLL/CLKO]

```

In addition to the PLL output clock, you also need to define a clock divider at the output of the FFdiv flip-flop, using the following command:

```

create_generated_clock \
    -name pll_FDBK_clk \
    -source [get_pins UPLL/CLKO] \
    -divide_by 2 \
    [get_pins FFdiv/Q]

```

This setup allows PrimeTime to perform correct PLL adjustment.

PLL Drift and Jitter

The PLL drift and PLL jitter characteristics of the phase-corrected output clock are defined using the `set_clock_latency` command for the PLL output clock with the `-pll_shift` option. The presence of the `-pll_shift` option implies that the delay that is specified is added to the base early or late latency of the PLL generated clock.

Note that this is different than the usual behavior of the `set_clock_latency` command, where PrimeTime overrides the clock latency values with the specified values. When you specify this option, the delay value corresponds to the PLL drift. PLL jitter is specified using both the `-pll_shift` and `-dynamic` options. Specify the `-pll_shift` option for the generated clock that is defined at the PLL output connected to the PLL feedback pin. PrimeTime applies the same shift to every output of the PLL.

The following example shows the command syntax for PLL jitter:

```
set drift 0.100
set jitter 0.020

create_clock -period 5 [get_ports CLK]
create_generated_clock -name CLK_pll -divide_by 1 \
    -source UPLL/CLKIN \
    -pll_feedback UPLL/FDBK \
    -pll_output UPLL/CLKOUT
set_clock_latency -source -pll_shift [get_clocks CLK_pll] \
    -early -${drift} -dynamic -${jitter}
set_clock_latency -source -pll_shift [get_clocks CLK_pll] \
    -late +${drift} -dynamic +${jitter}
set_propagated_clock {CLK CLK_pll}
```

For more information, see the `set_clock_latency` command.

CRPR Calculations for PLL Paths

For PLLs with more than one output, PrimeTime considers all the output clocks to have the same phase. Thus, PrimeTime considers the outputs to be indistinguishable from each other with respect to clock reconvergence pessimism calculations. For example, for a PLL with two outputs, OUTCLK1 and OUTCLK2, and a path launched by a clock at OUTCLK1 and captured by a clock at OUTCLK2, PrimeTime removes the clock reconvergence pessimism up to the outputs of the PLL. PrimeTime does this, even though the last physical common pin on the launch and capture clock paths is the reference pin of the PLL. Removing the clock reconvergence pessimism is essential as the output clocks of the PLL have to be in phase with each other. The `report_crpr` command demonstrates this behavior by showing one of the PLL outputs as the common pin for paths launched and captured by different outputs of the PLL.

Reporting and Timing Checks

The `check_timing` command validates that a PLL clock reaches each feedback pin. This check is also performed when using the `update_timing` command.

The `report_timing` command includes the PLL adjustment next to the output of the PLL as follows:

```
...
clock CLK (rise edge)          0.00      0.00
clock source latency           0.00      0.00
clk_in (in)                   0.00      0.00 r
inst_my_pll/REFCLK (my_pll)    0.00      0.00 r
inst_my_pll/OUTCLK (my_pll)   (gclock source) -798.03 *   -798.03 r
b1/A (buf1a1)                 0.00      -798.03 r
...
...
```

Requirements for PLL Library Cells

The PLL library cell should have a single positive unate timing arc from the reference clock pin to each of the outputs of the PLL. The reference pin, output clock pin, and feedback pin of the PLL are identified by `is_pll_reference_pin`, `is_pll_output_pin`, and `is_pll_feedback_pin` attributes, respectively. You can use the `is_pll_cell` cell attribute to identify a particular cell as a PLL cell. An example of a PLL library cell is as follows:

```
cell(my_pll) {
    is_pll_cell : true;

    pin( REFCLK ) {
        direction : input;
        is_pll_reference_pin : true;
    }

    pin( FBKCLK ) {
        direction : input;
        is_pll_feedback_pin : true;
    }

    pin (OUTCLK1) {
        direction : output;
        is_pll_output_pin : true;
        timing() { // Timing Arc
            related_pin: "REFCLK";
            timing_sense: positive_unate;
            cell_fall(scalar) {
                values("0.0")
            }
        }
    }
}
```

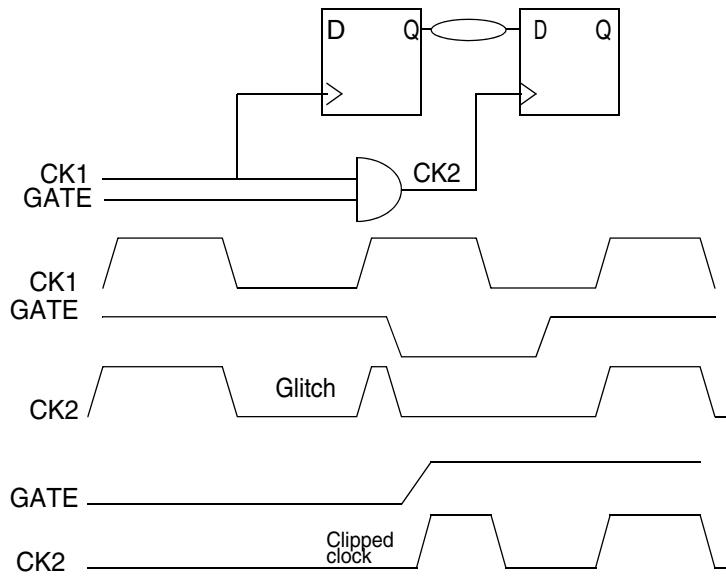
```
pin (OUTCLK2) {
    direction : output;
    is_pll_output_pin : true;
    timing() // Timing Arc
        related_pin: "REFCLK";
        timing_sense: positive_unate;
        cell_fall(scalar) {
            values("0.0")
        }
    }
}
```

Specifying Clock-Gating Setup and Hold Checks

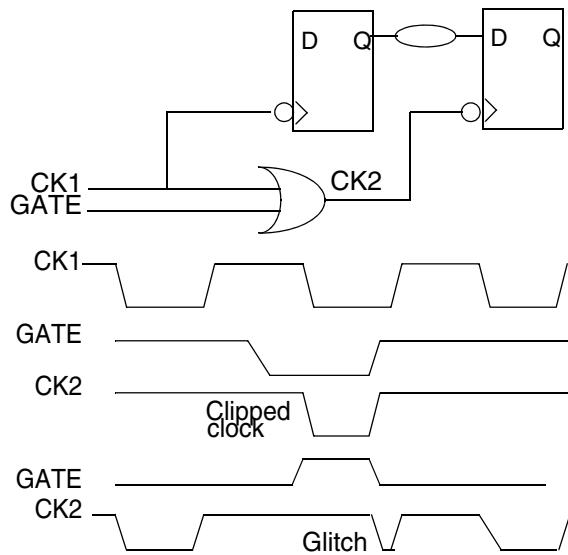
A gated clock signal occurs when the clock network contains logic other than inverters or buffers. For example, if a clock signal acts as one input to a logical AND function and a control signal acts as the other input, the output is a gated clock.

PrimeTime automatically checks for setup and hold violations on gating inputs to ensure that the clock signal is not interrupted or clipped by the gate. This check is performed only for combinational gates where one signal is a clock that can be propagated through the gate, and the gating signal is not a clock.

The clock-gating setup check ensures that the control data signal enables the gate before the clock becomes active. The arrival time of the leading edge of the clock pin is checked against both edges of any data signal feeding the data pins to prevent a glitch at the leading edge of the clock pulse or clipped clock pulse. The clock-gating setup violations are shown in [Figure 6-20](#).

Figure 6-20 Clock-Gating Setup Violations

The clock-gating hold check ensures that the control data signal remains stable while the clock is active. The arrival time of the trailing edge of the clock pin is checked against both edges of any data signal feeding the data pins using the hold equation. A clock-gating hold violation causes either a glitch at the trailing edge of the clock pulse or a clipped clock pulse. Note that for clock-gating checks, it is required that the gating check must reach a clock pin to succeed. These clock pins include reference pins of sequential timing constraints, from-pins of sequential arcs, and so on. The clock-gating hold violations are shown in [Figure 6-21](#).

Figure 6-21 Clock-Gating Hold Violations

By default, PrimeTime checks setup and hold times for gated clocks. A value of 0.0 is set as the setup and hold time (unless the library cell for the gate has gating setup or hold timing arcs). You can specify a nonzero setup or hold value with the `set_clock_gating_check` command.

The `set_clock_gating_check` command affects only clock-gating checks that exist at the specified cell or pin. To apply the clock-gating parameters for an entire clock domain's network, specify a clock object by using the `set_clock_gating_check ... [get_clocks CLK]` command. For example, to specify a setup requirement of 0.2 and a hold requirement of 0.4 on all gates in the clock network of CLK1, enter

```
pt_shell> set_clock_gating_check -setup 0.2 -hold 0.4 [get_clocks CLK1]
```

To specify a setup requirement of 0.5 on gate and1, enter

```
pt_shell> set_clock_gating_check -setup 0.5 [get_cells and1]
```

The `report_clock_gating_check` command performs clock-gating setup and hold checks. Use it to identify any clock-gating violations.

```
pt_shell> report_clock_gating_check objects
```

To remove clock-gating checks set with the `set_clock_gating_check` command, use the `remove_clock_gating_check` command.

Disabling or Restoring Clock-Gating Checks

You can specify any pin or cell in the current design or subdesigns to disable or restore clock-gating checks.

To disable clock-gating checks on cells or pins, use this command:

```
pt_shell> set_disable_clock_gating_check objects
```

To restore clock-gating checks on cells or pins, use this command:

```
pt_shell> remove_disable_clock_gating_check objects
```

If objects are not specified, all clock-gating checks are disabled. It is equivalent to setting the `timing_disable_clock_gating_checks` variable to `true`.

For example, to restore disabled clock-gating checks for the object U44, use this command:

```
pt_shell> remove_disable_clock_gating_check U44
```

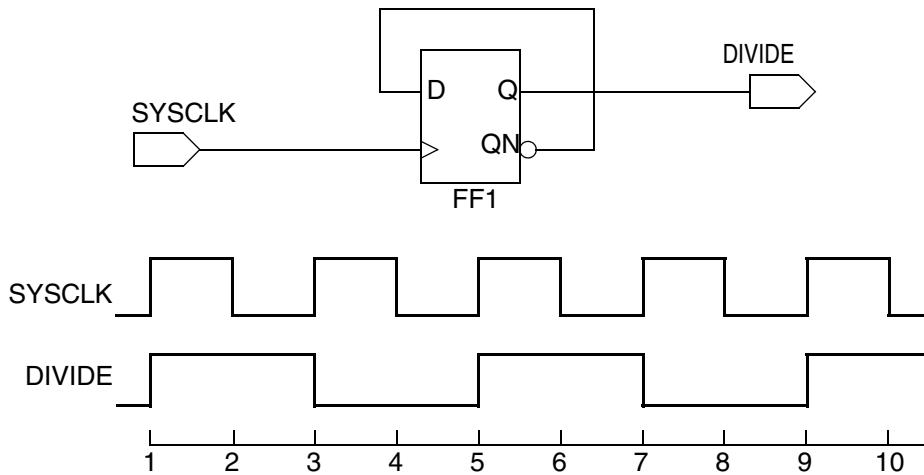
To disable clock-gating checks on the specified cell or pin, use this command:

```
pt_shell> set_disable_clock_gating_check U44/z
```

Specifying Internally Generated Clocks

A design might include clock dividers or other structures that produce a new clock from a master source clock. A clock that is generated by on-chip logic from another clock is called a generated clock.

[Figure 6-22](#) shows the waveforms of the master and generated clock for a divide-by-2 clock generator. The clock waveform is ideal, with no clock-to-Q delay. PrimeTime does not consider the circuit logic of a clock generator, so you must specify the behavior of a generated clock as a separate clock. However, you can define the relationship between the master clock and the generated clock, so that the generated clock characteristics change automatically when the master clock is changed.

Figure 6-22 Divide-by-2 Clock Generator

The `create_generated_clock` command specifies the characteristics of an internally generated clock. By default, using `create_generated_clock` on an existing generated clock object overwrites that clock. Generated clock objects are expanded to real clocks at the time of analysis. The clock expansion happens automatically within the `report_timing` command or explicitly with the `update_timing` command.

You can create the generated clock as a frequency-divided clock (`-divide_by` option), frequency-multiplied clock (`-multiply_by` option), or edge-derived clock (`-edges` option). You can modify the generated clock waveform by specifying either `-multiply_by`, `-divide_by`, or `-edges` with the `-combinational` option. When you create the generated clock using the `-combinational` option, there must be a valid path for propagating the rise and fall edges of master clock to the generated clock source pin and the source latency paths for this type of generated clock only includes the logic where the master clock propagates.

Specifying a Divide-by-2 Generated Clock

To specify a divide-by clock, use the `-divide_by` option of the `create_generated_clock` command and specify the frequency division factor. For example, to create the divide-by-2 generated clock in [Figure 6-22](#), shown previously, specify 2 as the frequency division factor:

```
pt_shell> create_generated_clock -name DIVIDE \
    -source [get_ports SYSCLK] -divide_by 2 [get_pins FF1/Q]
```

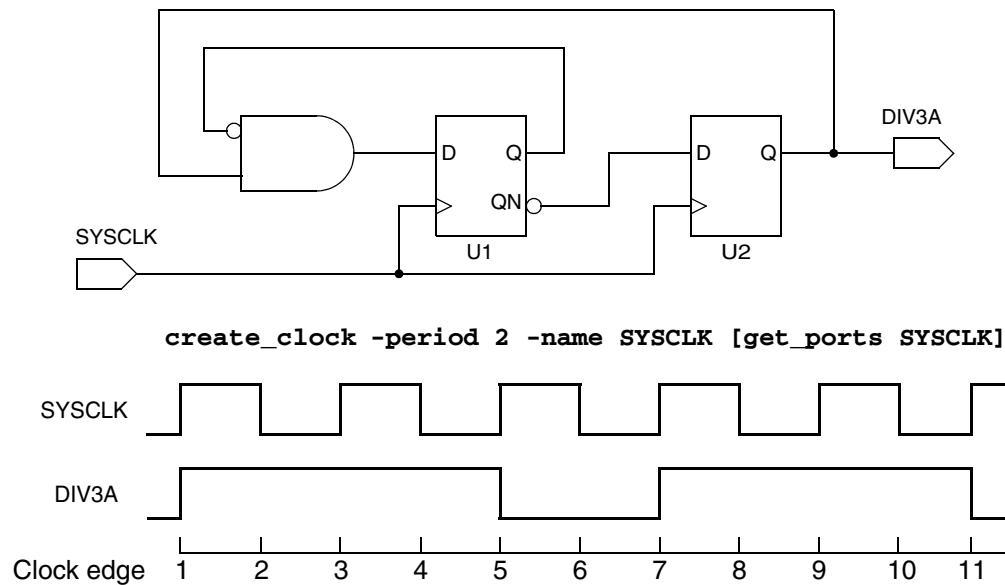
Specify a port or a pin (not a clock name) as the master source from which the new clock is generated. Specify a pin as the creation point for the new generated clock.

Note that generated clock edges are based on occurrences of rising edges at the master clock source pin (specified with the `-source` option). If you need to create a generated clock based on falling edges at the master clock source pin, use the `-edges` option rather than the `-divide_by` option (see [Creating a Divide-by Clock Based on Falling Edges](#)).

Creating a Generated Clock Based on Edges

You can use the `create_generated_clock -edges` command to specify the generated clock in terms of edges of the master clock waveform on the master pin. For example, consider the clock generator circuit shown in [Figure 6-23](#).

Figure 6-23 Divide-by-3 Clock Generator



The generated clock signal `DIV3A` has a period three times longer than the master clock, with an asymmetrical waveform. To specify this waveform, enter

```
pt_shell> create_generated_clock -edges { 1 5 7 } \
           -name DIV3A -source [get_ports SYSCLK] [get_pins U2/Q]
```

Creating Clock Senses for Pulse Generators

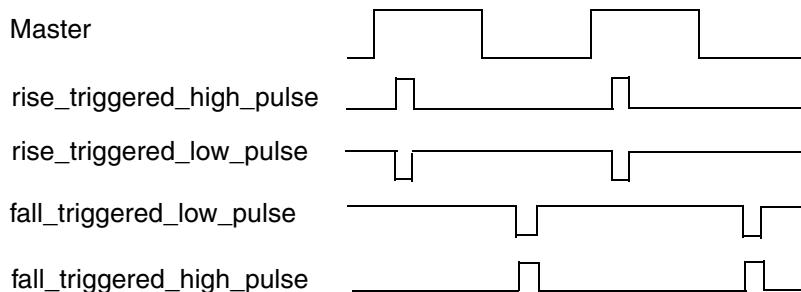
Pulse generators can improve circuit performance and save power consumption. Because of clock gating, skew management, and preserving generated pulse integrity, it might be necessary to insert thousands of pulse generators in a design to control the pulse. Although you can specify `create_generated_clock` with nondecreasing edges at the output of each pulse generator, it does not scale well. Not only do you have to find and specify each

generated clock, the proliferation of clock groups makes it hard to use. Therefore, for pulse generators that do not change the frequency of the incoming clock, you can set the `pulse_clock` attribute to the following senses to support the pulse generators:

- `rise_triggered_high_pulse`
- `rise_triggered_low_pulse`
- `fall_triggered_high_pulse`
- `fall_triggered_low_pulse`

Setting these clock senses supports the four types without the need to add a generated clock.

Figure 6-24 Pulse Clock Types That Do Not Change Frequency



This pin is allowed on internal, bidirectional, or output pins. If you set it on a bidirectional pin, it affects only those clock paths that follow through the output side of that pin. The ideal clock width for clocks with any of the pulse clock senses is 0. The pulse width is computed as:

```
high pulse = fall_network_latency - rise_network_latency
low pulse = rise_network_latency - fall_network_latency
```

Use the `set_clock_latency` command to set the ideal clock pulse width. You can add the command to any pin in the clock network that affects all registers in the fanout of the command. For example, to set an ideal pulse high width of 0.5 for all registers downstream from pin PG/Z and with an overall latency of 0.6, use these commands:

```
set_clock_latency -rise 0.6 PG/Z
set_clock_latency -fall 1.1 PG/Z
```

If your waveform is non-monotonic, yet has increasing values, use the `create_clock -waveform` command. This instructs PrimeTime to create the correct clock for those clocks driven by a pulse generator. For example,

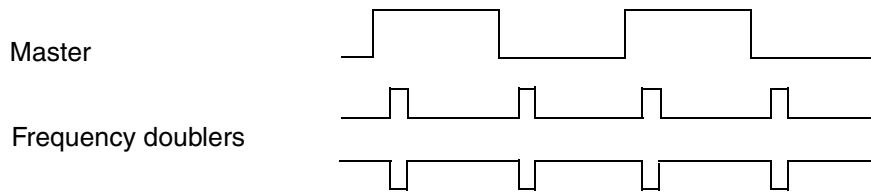
```
create_clock -waveform {0.0 0.0} -period 10 [get_ports BCK]
set_clock_latency -source -rise 0.23 -fall 0.65 [get_ports BCK]
```

PrimeTime generates an error for any clock specified as a pulse generator if it combines with another sense of the same clock. To specify the clock sense in those locations where

multiple senses of the same clock have merged together, and it is ambiguous which sense PrimeTime should use, use the `set_sense -type clock` command. Specify the type of pulse clock sense you want using the `set_sense -type clock -pulse` command, specifying the value as `rise_triggered_high_pulse`, `rise_triggered_low_pulse`, `fall_triggered_high_pulse`, or `rise_triggered_low_pulse`.

As previously mentioned, you can handle pulse generators that alter frequency by using the `create_generated_clock` command. For example, this approach works well in the case of frequency doublers, as shown in [Figure 6-25](#).

Figure 6-25 Pulse Clock Types That Change Frequency

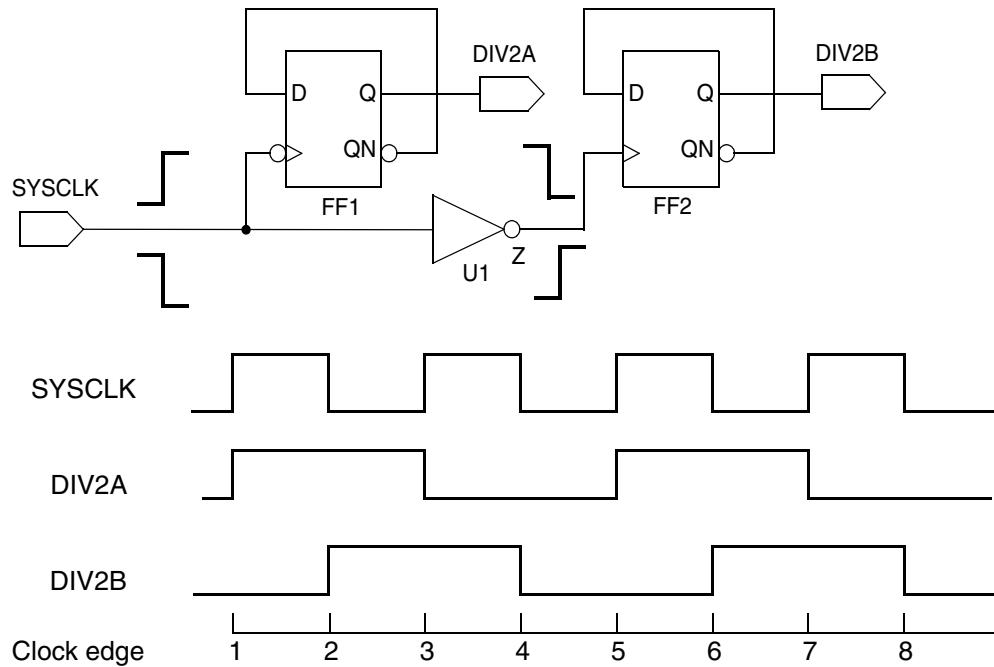


If the requested sense of clock does not exist, PrimeTime issues a warning. For example, if there was only a positive unate path from the clock source to a pin with `set_sense -type clock -pulse rise_triggered_high_pulse`, PrimeTime does not run the command as no rising at the clock source to failing at the clock sense pin path is found. However, if there are both a positive and negative unate path from the clock source to a pin, PrimeTime runs this same command with the rise latency coming from the positive unate path and the fall latency coming from the negative unate path.

Creating a Divide-by Clock Based on Falling Edges

If you need to create a generated clock based on falling edges at the master clock pin, use the `create_generated_clock` command with the `-edges` option. The generated clock examples in [Figure 6-26](#) demonstrate how to do this.

Figure 6-26 Generated Divide-by-2 Clocks Based on Different Edges



The generated clock DIV2A is based on the rising edge of the master clock at the SYSCLK port, so you can specify the generated clock using either the `-divide_by` option or the `-edges` option:

```
pt_shell> create_generated_clock -name DIV2A \
    -source [get_ports SYSCLK] -divide_by 2 [get_pins FF1/Q]

pt_shell> create_generated_clock -name DIV2A \
    -source [get_ports SYSCLK] -edges { 1 3 5 } [get_pins FF1/Q]
```

The generated clock DIV2B is based on the falling edge of the master clock at the SYSCLK port, so you cannot use the `-divide_by` option. However, you can still use the `-edges` option:

```
pt_shell> create_generated_clock -name DIV2B \
    -source [get_ports SYSCLK] -edges { 2 4 6 } [get_pins FF2/Q]
```

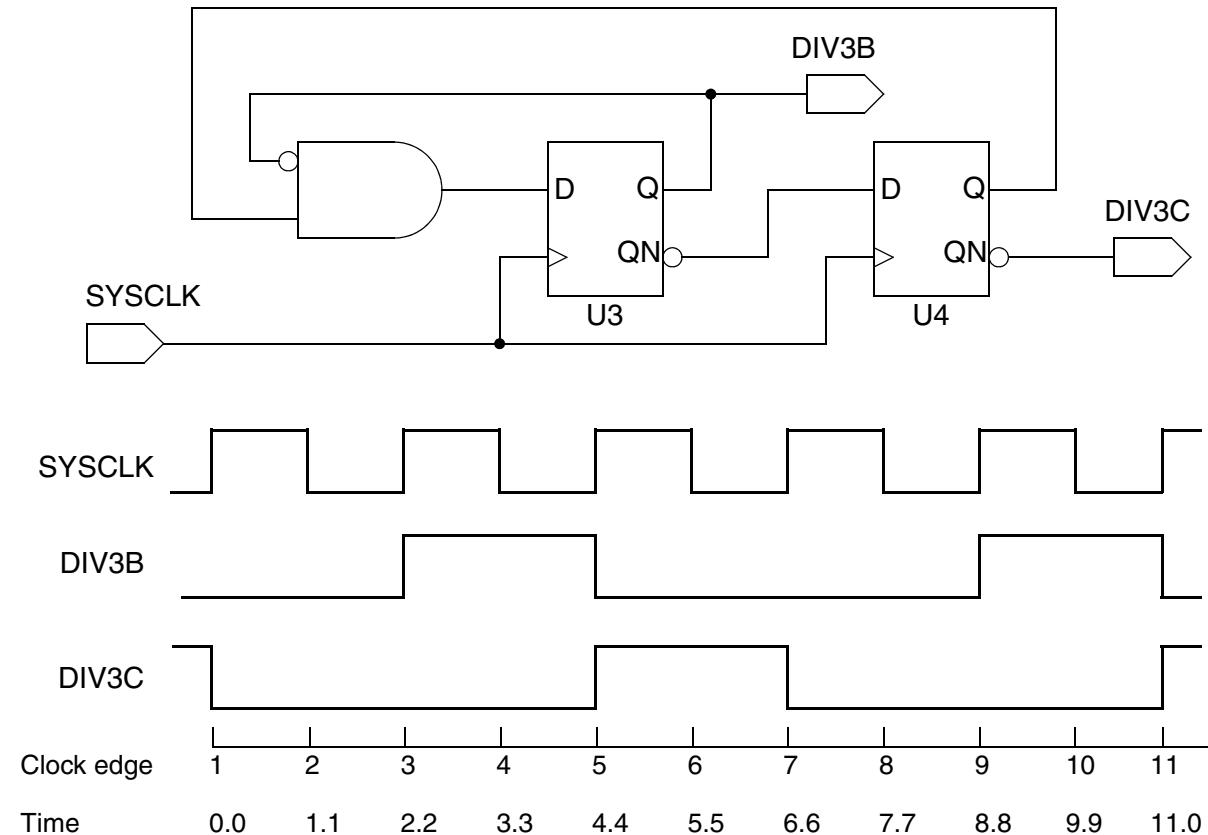
Another way to specify DIV2B is to use a different source pin:

```
pt_shell> create_generated_clock -name DIV2B \
    -source [get_pins U1/Z] -edges { 1 3 5 } [get_pins FF2/Q]
```

Shifting the Edges of a Generated Clock

You can shift the edges of a generated clock by a specified amount of time. This shift is not considered clock latency. For example, consider the clock generator circuit shown in [Figure 6-27](#).

Figure 6-27 Generated Divide-by-3 Clock With Shifted Edges



To specify the master source clock and the two generated clocks, you could use commands such as the following:

```
pt_shell> create_clock -period 2.2 -name CLK [get_ports SYSCLK]  
  
pt_shell> create_generated_clock -edges { 3 5 9 } \  
      -name DIV3B -source [get_ports SYSCLK] [get_pins U3/Q1]  
  
pt_shell> create_generated_clock -edges { 3 5 9 } \  
      -edge_shift { 2.2 2.2 2.2 } \  
      -name DIV3C -source [get_ports SYSCLK] [get_pins U4/QN]
```

Note:

You can use the `-edge_shift` option only in conjunction with the `-edges` option.

The `report_clock` command reports the specified clock as follows,

p - propagated_clock
G - Generated clock

Clock	Period	Waveform	Attrs	Sources
<hr/>				
-				
CLK	2.20	{0 1.1}		{SYSCLK}
DIV3B	6.60	{2.2 4.4}	G	{U3/Q}
DIV3C	6.60	{4.4 6.6}	G	{U4/Q}
Generated Clock	Master Source	Generated Source	Waveform Modification	
<hr/>				
-				
DIV3B	MYCLK	U3/Q	edges(3 5 9)	
DIV3C	MYCLK	U4/Q	edges(3 5 9) shifts(2.2 2.2 2.2)	

Multiple Clocks at the Source Pin

The `create_generated_clock` command defines a clock that depends on the characteristics of a clock signal reaching a pin in the design, as specified by the `-source` argument. Because it is possible for the source pin to receive multiple clocks, you might need to specify which clock is used to create the generated clock. To specify a clock, use the `-master_clock` option together with the `-source` option of the `create_generated_clock` command.

To get information about a generated clock, use the `report_clock` command. The report tells you the name of the master clock, the name of the master clock source pin, and the name of the generated clock pin.

Selecting Generated Clock Objects

The `get_clocks` command selects a clock object. You can restrict the clock selection to certain clocks according to naming conventions or by filtering. This command returns a token that represents a collection of clocks whose names match the pattern and whose attributes pass a filter expression. For example, if all generated clock names match the pattern `CLK_DIV*`, you can do the following:

```
pt_shell> set_false_path \
    -from [get_clocks CLK_DIV*] \
    -to [get_clocks CLKB]
```

If the generated clocks do not follow a naming pattern, you can use filtering based on attributes instead. For example,

```
pt_shell> set_false_path \
           -from [get_clocks CLK* \
           -filter "is_generated==TRUE"] \
           -to [get_clocks CLKB]
```

Reporting Clock Information

The `report_clock` command shows information about clocks and generated clocks in the current design. For a detailed report about a clock network, use the `report_clock_timing` command as described in [Clock Network Timing Report](#).

The `report_transitive_fanout -clock_tree` command shows the clock networks in your design.

Removing Generated Clock Objects

The `remove_generated_clock` command removes generated clocks. To remove the generated clock named `CLK_DIV2` (and its corresponding clock object), enter

```
pt_shell> remove_generated_clock CLK_DIV2
```

Generated Clock Edge Specific Source Latency Propagation

PrimeTime calculates the clock source latency for a generated clock taking into account the edge relationship between master clock and generated clock. It finds the worst-case path that propagates forward to produce the specified edge type defined by the generated clock with respect to the master clock source.

If paths exist, but they do not satisfy the needed edge relationships, PrimeTime returns an error message (UITE-461) stating that no path is reported for generated clock source latency (zero clock source latency is used). The following is an example of an unsatisfied generated clock returning a UITE-461 error message:

```
Error: Generated clock 'clk_div2' 'rise_edge' is not
      satisfiable; zero source latency will be used. (UITE-461)
Error: Generated clock 'clk_div2' 'fall_edge' is not
      satisfiable; zero source latency will be used. (UITE-461)
```

Path Type: min

Point	Incr	Path
<hr/>		
clock clk_div2 (rise edge)	0.000	0.000
clock source latency	0.000	0.000
Udiv/Q (FD1)	0.000	0.000 r
clock clk_div2 (rise edge)	0.000	0.000
clock source latency	0.000	0.000
Udiv/Q (FD1)	0.000	0.000 r
...		

Clock Mesh Analysis

PrimeTime SI provides an easy-to-use method to analyze a clock mesh network with transistor-level accuracy using SPICE simulation. If the clock network remains unchanged, you can reuse the simulation results in subsequent timing and signoff analysis in PrimeTime.

To learn how to perform clock mesh analysis, see

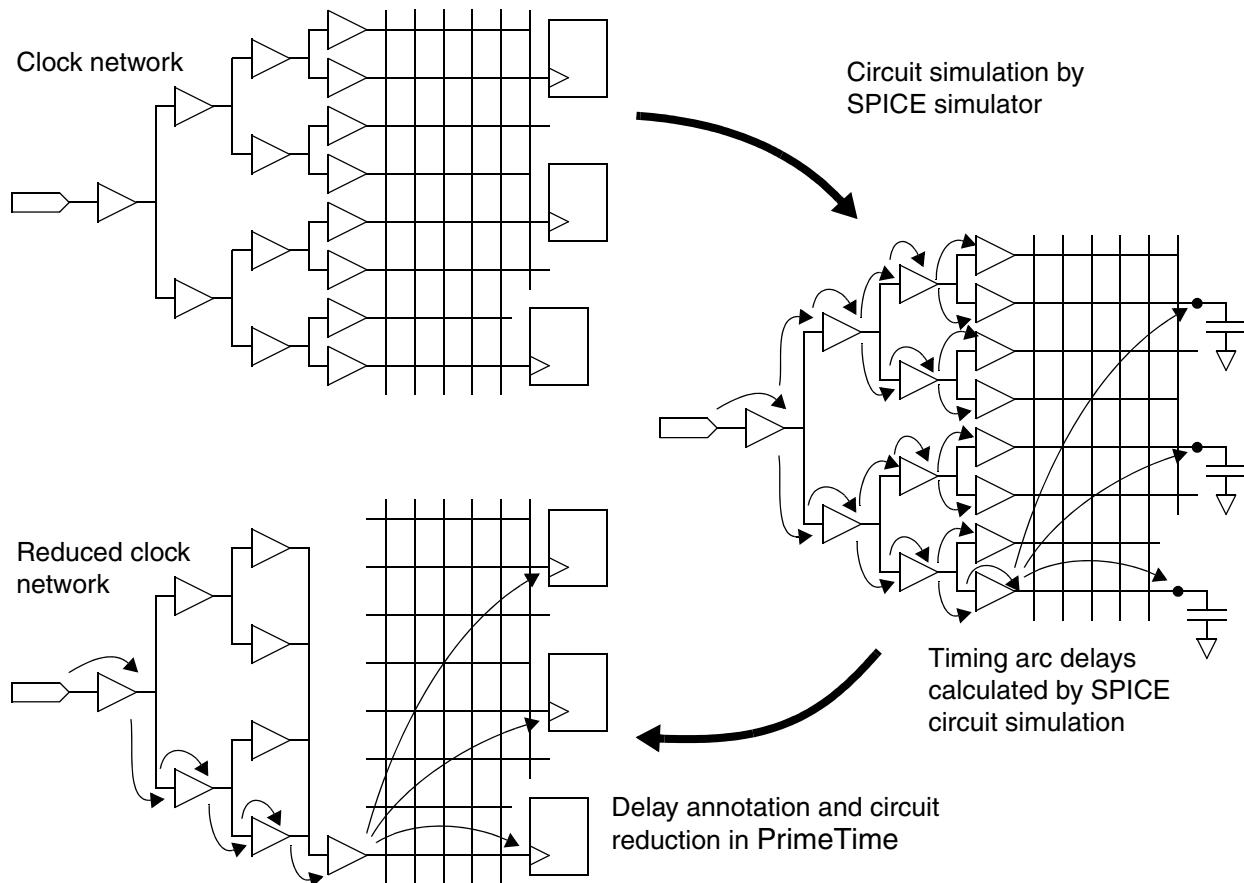
- [Overview of Clock Mesh Analysis](#)
 - [Usage Flow](#)
 - [Clock Network Simulation Commands](#)
-

Overview of Clock Mesh Analysis

To use clock mesh analysis, specify the root node of the clock network you want to analyze. Beginning at this node, PrimeTime SI traces the entire clock network, including the clock mesh and sequential device loads. Next, PrimeTime invokes an external HSPICE-compatible circuit simulator to simulate the network and determine the exact timing of each individual cell arc and net arc in the clock network.

PrimeTime SI back-annotates the measured delays on the design and then performs timing arc reduction on the clock mesh structure. This reduction process retains the single mesh driver having the least delay and disables the timing through the other drivers of the mesh. The timing arcs starting from the disabled drivers are replaced by equivalent timing arcs that start from the single retained driver, as shown in [Figure 6-28](#).

Figure 6-28 Clock Mesh Analysis and Circuit Reduction

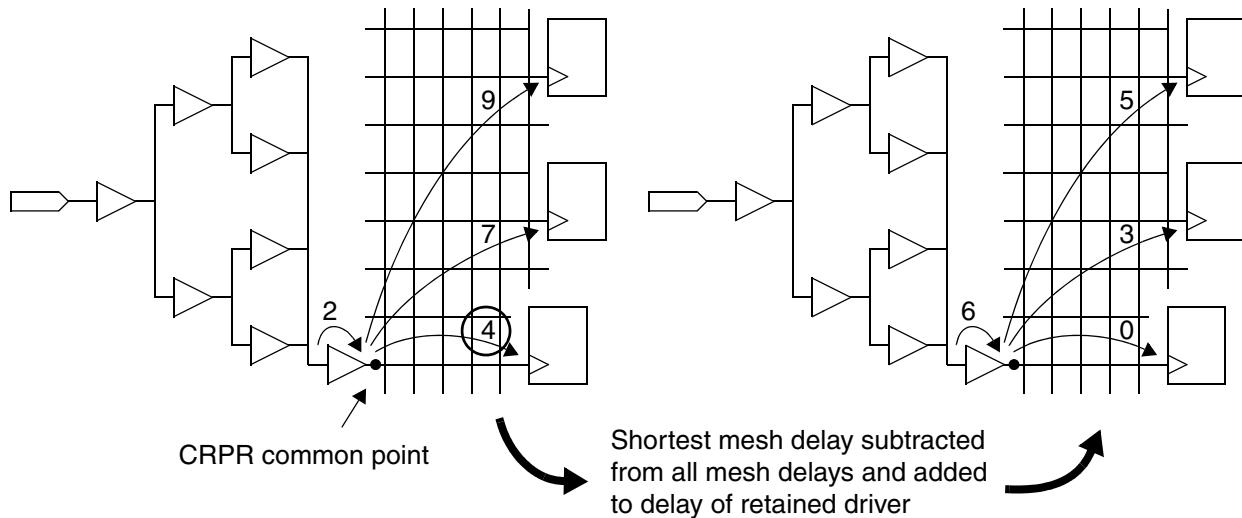


The purpose of circuit reduction is to avoid the large number of combinations of drivers and loads in a full-mesh analysis, while maintaining accurate driver-to-load timing results. If the mesh has n drivers and m loads, there are $n \times m$ timing arcs between drivers and loads in the mesh. However, by reducing the mesh circuit to a single driver, the number of driver-to-load timing arcs is reduced to just m .

The circuit has various delay values from the single mesh driver to many loads on the mesh. For clock reconvergence pessimism removal (CRPR), the shortest delay from the mesh driver to the nearest mesh load represents the amount of delay that is shared by all the timing arcs from the driver to the loads. To gain the most benefit of CRPR, this shared delay is accounted for before the CRPR common point. Accordingly, PrimeTime SI makes the adjustment shown in [Figure 6-29](#) during CRPR calculations.

PrimeTime SI uses the output of the single retained mesh driver as the CRPR common point. It finds the shortest net delay through the mesh, adds that value to the mesh driver cell delay, and subtracts that same value from all net delays through the mesh.

Figure 6-29 Clock Reconvergence Pessimism Removal From Mesh



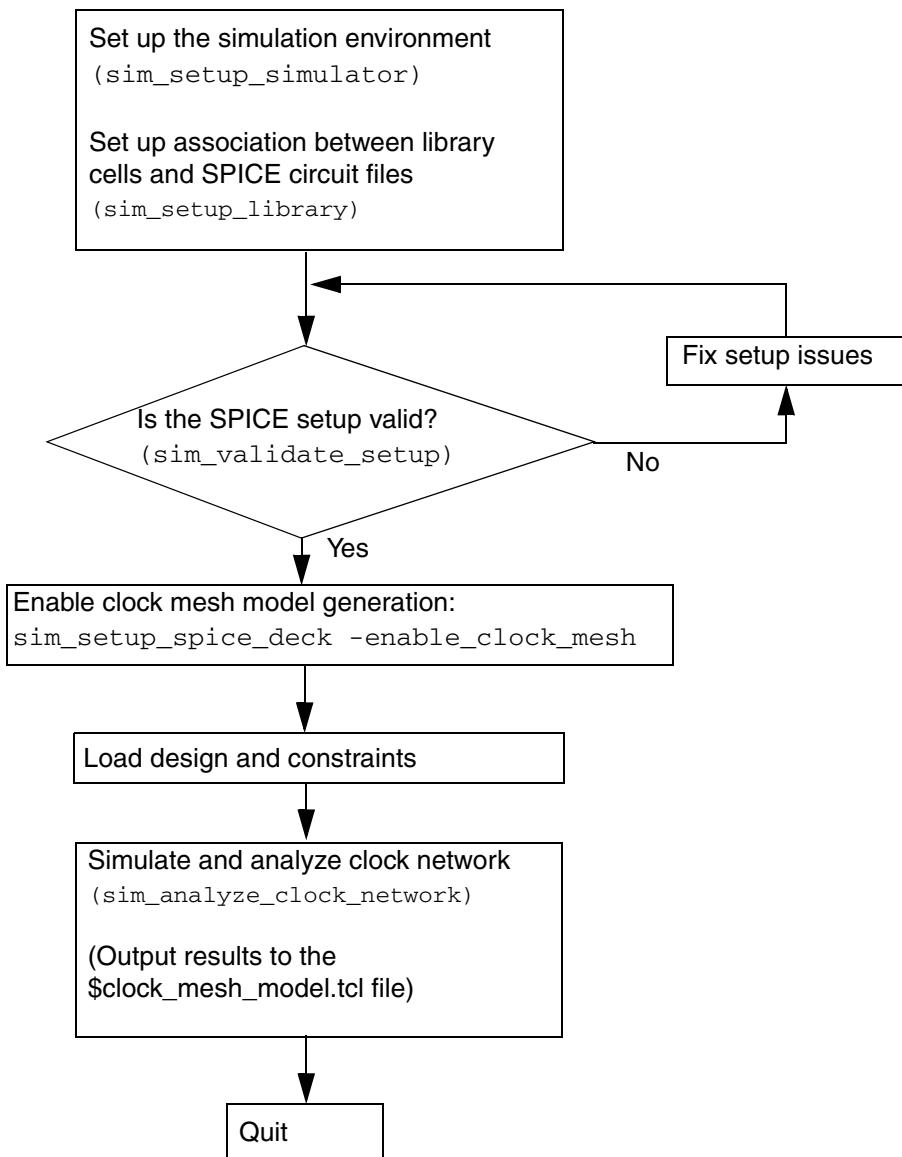
If you use the `report_timing` command to report a timing path that goes through the mesh, the report shows the timing through the single retained mesh driver and reflects the results obtained from the SPICE simulations.

Usage Flow

To perform clock mesh analysis,

1. Generate the clock mesh model using clock network simulation commands. Perform this step one time. [Figure 6-30](#) summarizes the flow.

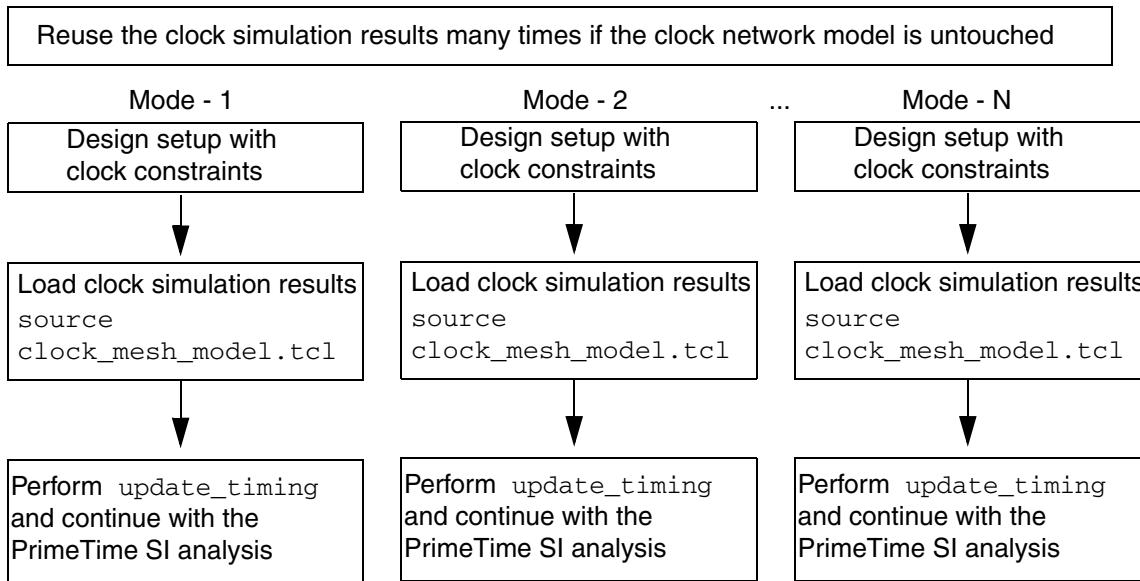
Figure 6-30 Generating the Clock Mesh Model



2. Reuse the generated clock mesh model as many times as necessary.

If the clock network model remains the same for different analysis runs such as timing analysis, engineering change order (ECO) netlist editing, and constraints clean up, you can reuse the generated clock mesh model. [Figure 6-31](#) shows the generated clock model reuse for analysis with different design modes.

Figure 6-31 Reusing the Clock Mesh Model in PrimeTime Analysis



To prepare for the clock network simulation, use the `sim_setup_simulator` command to set up the SPICE simulator. Set up the SPICE circuit files associated with the logic library cells that are used in the clock network with the `sim_setup_library` command. Optionally, check the setup with the `sim_validate_setup` command.

To perform the simulation and back-annotate the timing results on the clock network, use the `sim_analyze_clock_network` command. The simulation results are stored in a Tcl script file.

The following script demonstrates the clock network simulation and analysis flow:

```

# Setup simulation environment -- SPICE simulator, working directory,
# transistor models, ...

sim_setup_simulator -simulator /usr/bin/hspice -work_dir ./tmp_dir \
-sim_options ".option ingold=2 numdgt=6 statfl=1 ..."

foreach lib [get_libs *] {
    sim_setup_library -library $lib -sub_circuit ...
}

# Validate setup for each library if needed
sim_validate_setup -lib_cell ...

# Enable clock mesh model generation
sim_setup_spice_deck -enable_clock_mesh

```

```

# Set up design, parasitics, and clock-related constraints
read_verilog ...
link
read_parasitics ...
create_clock ...
...

# Perform clock network related timing update, simulation and
# back-annotation.
sim_analyze_clock_network -from $clock_root_pin \
    -output $clock_mesh_model.tcl ...

```

In subsequent PrimeTime analysis runs, you can reuse the same simulation results without repeating the clock mesh model generation steps if the clock network has not changed. Source the Tcl script file generated by the `sim_analyze_clock_network` command for each subsequent PrimeTime analysis run.

Clock Network Simulation Commands

The following commands support the clock network simulation and analysis feature:

- [sim_setup_simulator](#)
- [sim_setup_library](#)
- [sim_setup_spice_deck](#)
- [sim_validate_setup](#)
- [sim_analyze_clock_network](#)

sim_setup_simulator

The `sim_setup_simulator` command sets up the simulation environment by specifying the simulator location, simulator type, and working directory. Here is an example of the `sim_setup_simulator` command.

```
pt_shell> sim_setup_simulator -simulator /usr/bin/hspice \
    -simulator_type hspice -work_dir ./tmp_dir \
    -sim_options ".option ingold=2 numdgt=6 statfl=1 ..."
```

The following options are available in the `sim_setup_simulator` command:

- The `-simulator` option specifies the path to the simulator executable.
- The `-simulator_type` option specifies the type of the simulator. The default is `hspice` for HSPICE, `hsim` for HSIM, and `nanosim` for NanoSim. If the simulator type is unspecified, PrimeTime attempts to derive the simulator type using the path to the simulator executable specified with the `-simulator` option.

- The `-sim_options` specifies the options for the simulator. The simulator options are specified as a string that is appended to the SPICE decks.
- The `-work_dir` option specifies the working directory where temporary SPICE files are stored. This setting overrides any previous working directory settings. The default working directory is `$pt_tmp_dir`.
- The `-preserve` option specifies whether to retain the temporary simulation files. It can be set to `failed`, `all`, or `none` to preserve the temporary files for a failed simulation, all simulations, or none of the simulations, respectively. The default is `failed`.

sim_setup_library

The `sim_setup_library` command performs library setup tasks such as mapping the transistor models to the gate-level models. Here is an example of the `sim_setup_library` command.

```
pt_shell> sim_setup_library -library $lib_name \
    -sub_circuit /u/xtalk/si_lib_gen/unit/gem/hspice \
    -header /u/xtalk/si_lib_gen/unit/gem/model_hspice \
    -file_name_pattern {my_%s.spc}
```

The following options are available in the `sim_setup_library` command:

- The `-library` option specifies the PrimeTime gate-level library to be associated with the SPICE setup.
- The `-sub_circuit` option specifies the directory where PrimeTime locates the subcircuit files, one file per cell, to be included in the simulation decks to model the behavior of the cells. Alternatively, you can specify a single file containing all the subcircuits; however, using a single file can result in reduced simulation performance.
- The `-header` option specifies the name of a header file to be included at the beginning of the SPICE file. This file must contain the model file, corner instantiation, and any other simulator options. The file must not contain any measure statement or other circuit specific statements, and it must not contain temperature or voltage statements. Generally, the SPICE options must be the same in all header files for each library.
- The `-file_name_pattern` option provides a restricted space for the name of subcircuit files selected from the directory specified by the `-sub_circuit` option. This allows the directory specified with the `-sub_circuit` option to contain subcircuits for multiple libraries.

sim_setup_spice_deck

The `sim_setup_spice_deck` command specifies the setup options to write out the SPICE deck. Here is an example of using this command to enable clock model generation flow.

```
pt_shell> sim_setup_spice_deck -enable_clock_mesh
```

The following options are available in the `sim_setup_spice_deck` command:

- The `-enable_clock_mesh` option enables the model generation flow for performing clock network analysis. Set this option before reading the design parasitics. If this option is not set, the `sim_analyze_clock_network` command fails.
- The `-use_pin_load` option improves the SPICE simulation runtime in certain situations. It also enables SPICE simulations for load cells that do not have corresponding SPICE models. Using this option can result in a loss of accuracy for SPICE simulation because the physical load cells are replaced by the equivalent pin capacitance model.

sim_validate_setup

Setting the SPICE simulation environment incorrectly can cause problems in the clock mesh analysis flow. To help ensure that the setup is correct, you can use the `sim_validate_setup` command. This command invokes the simulator specified by the `sim_setup_simulator` command on the timing arcs of a given library cell and verifies that the basic characterization setting in the library matches the SPICE setup specified by the `sim_setup_simulator` and `sim_setup_library` commands. The `sim_validate_setup` command supports only combinational cells.

The `sim_validate_setup` command checks for the presence of a SPICE executable, SPICE netlists, SPICE model files, and applicable licenses. It checks for necessary SPICE options and data such as I/O rail settings and correct unit sizes. It also checks the model license and compatibility of the SPICE version with the model. If no errors are found, the delay and slew values computed by SPICE and PrimeTime are displayed.

Before you use the `sim_validate_setup` command, the library must be fully validated for accuracy. This command assumes that the library is correct and accurate, and any mismatch between PrimeTime and SPICE is caused by an incorrect simulator setting, not library errors. You can only use the command when no designs are loaded into PrimeTime. Here is an example of the `sim_validate_setup` command.

```
pt_shell> sim_validate_setup -from A -to Y \
    -lib_cell [get_lib_cells $lib_name/IV170BQ] \
    -capacitance 48 -transition_time 0.2
```

The following options are available in the `sim_validate_setup` command:

- The `-lib_cell` option specifies which library cell object to verify.
- The `-from` and `-to` options specify the set of library arcs between the input and output pins, respectively, of the library cell being validated.
- The `-capacitance` option specifies the load to be driven at the output of the library cell. The value is in capacitance units of the PrimeTime main library.
- The `-transition_time` option specifies the input transition time (slew) for the timing arc. A ramp or a composite current source (CCS) predriver waveform with this slew is

generated at the corresponding input pin of the library cell. Both rising and falling transitions are simulated. The value is in time units of the PrimeTime main library.

- The `-verbose` option produces more details in the echoed output, such as the location of the temporary files.

sim_analyze_clock_network

The `sim_analyze_clock_network` command extracts the clock network from the specified clock root and invokes the simulator specified by the `sim_setup_simulator` command. It creates a SPICE deck, links to the simulation environment, runs the simulation, and back-annotates the results onto the design in PrimeTime. Here is an example of the `sim_analyze_clock_network` command.

```
pt_shell> sim_analyze_clock_network -from [get_ports CLK1] \
-output ./clock_mesh_model.tcl
```

The following options are available in the `sim_analyze_clock_network` command:

- The `-from` option specifies the port or pin at the root of the clock network. The clock network is extracted by tracing from the root until sequential loads are reached. Clock-gating cells are included in the clock network.
- The `-from_rise_slew` and `-from_fall_slew` options specify the rise and fall slew, respectively, at the clock root node in library time units. If these slew values are specified in the command, they are used to create the input stimulus. Otherwise, the default rise and fall slew values from PrimeTime are used.
- The `-output` option specifies the name of the file to write the simulation results. The results are written in Synopsys Design Constraints (SDC) format using the `set_annotated_delay`, `set_annotated_transition`, and `set_disable_timing` commands. You can use this file as a replacement for simulating the clock network for multiple runs on the same design in PrimeTime if the clock network remains the same for each run.
- The `-to` option specifies a list of pins where the clock network tracing is stopped. This can help you control the extent of the clock network logic that is simulated and back-annotated. The clock network tracing continues until it reaches any of the specified pins or reaches the clock pin of a sequential device.
- The `-use_probe` option adds probe points to the SPICE simulation for all pins in the clock network.

7

Timing Paths and Exceptions

A timing path is a point-to-point sequence through a design that starts at a register clock pin or an input port, passes through combinational logic elements, and ends at a register data input pin or an output port.

For basic information about path startpoints and endpoints, delay calculation, setup and hold constraints, time borrowing, and timing exceptions, see [Overview of Static Timing Analysis](#). To learn more about timing paths, see

- [Path Groups](#)
- [Path Timing Reports](#)
- [Path Timing Calculation](#)
- [Path Specification Methods](#)
- [Timing Exceptions](#)

Path Groups

PrimeTime organizes paths into groups. This path grouping can affect the generation of timing analysis reports. For example, the `report_timing` command, when used with the `-group` option, reports the worst path in each of the listed path groups.

In Design Compiler, path grouping also affects design optimization. Each path group can be assigned a weight (also called cost function). The higher the weight, the more effort Design Compiler uses to optimize the paths in that group. You can assign weights to path groups in PrimeTime, but this weight information is not used in PrimeTime.

PrimeTime implicitly creates a path group each time you use the `create_clock` command to create a new clock. The name of the path group is the same as the clock name.

PrimeTime assigns a path to that path group if the endpoint of the path is a flip-flop clocked by that clock. PrimeTime also creates the following path groups implicitly:

- ****clock_gating_default**** – The group of paths that end on combinational elements used for clock gating.
- ****async_default**** – The group of paths that end on asynchronous preset/clear inputs of flip-flops.
- ****default**** – The group of constrained paths that do not fall into any of the other implicit categories; for example, a path that ends on an output port.
- **none** – Unconstrained paths.

In addition to these implicit path groups, you can create your own user-defined path groups by using the `group_path` command. This command also lets you assign any particular path to a specific path group. To get information about the current set of path groups, use the `report_path_group` command.

To remove a path group, use the `remove_path_group` command. Paths in that group are implicitly assigned to the default path group. For example, to place all paths to ports with names matching OUT_1* into their own group called out1bus, enter

```
pt_shell> group_path -name out1bus -to [get_ports OUT_1*]
```

Path Timing Reports

You can use path timing reports to focus on particular timing violations and to determine the cause of a violation. By default, the `report_timing` command reports the path with the worst setup slack among all path groups.

A path timing report provides detailed timing information about any number of requested paths. The level of detail you want to see in the output can vary. For example, you can view this information:

- Gate levels in the logic path
- Incremental delay value of each gate level
- Sum of the total path delays
- Amount of slack in the path
- Source and destination clock name, latency, and uncertainty
- OCV with clock reconvergence pessimism removed

The `report_timing` command provides options to control reporting of the following types of information:

- Number of paths
- Types of paths
- Amount of detail
- Startpoints, endpoints, and intermediate points along the path

The `report_timing` command by itself, without any options, is the same as the following command:

```
pt_shell> report_timing -path_type full -delay_type max -max_paths 1
```

PrimeTime generates a maximum-delay (setup constraint) report on the full path, showing only the single worst path. For example,

```
pt_shell> report_timing
...
Startpoint: a (input port)
Endpoint: c_d (output port)
Path Group: default
Path Type: max
Point           Incr      Path
-----
input external delay    10.00   10.00 r
a (in)             0.00    10.00 r
m1/Z (MUX21H)       1.00    11.00 r
u1/S (FA1)          1.00    12.00 r
c_d/Z (AN2)         1.00    13.00 r
c_d (out)           0.00    13.00 r
data arrival time                13.00
max_delay          15.00   15.00
```

output external delay	-10.00	5.00
data required time		5.00

data required time		5.00
data arrival time		-13.00

slack (VIOLATED)		-8.00

To report the four worst setup paths among all path groups, enter

```
pt_shell> report_timing -max_paths 4
```

When the `-max_paths` option is set to any value larger than 1, the command only reports paths that have negative slack. To include positive-slack paths in multiple-paths reports, use the `-slack_lesser_than` option, as in the following example:

```
pt_shell> report_timing -slack_lesser_than 100 -max_paths 40
```

To report the worst setup path in each path group, enter

```
pt_shell> report_timing -group [get_path_groups *]
```

To show the transition time and capacitance, enter

```
pt_shell> report_timing -transition_time -capacitance
```

PrimeTime displays a report similar to the following:

```
*****
Report : timing
    -path full
    -delay max
    -max_paths 1
    -transition_time
    -capacitance
Design : counter
...
*****
Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point                Cap      Trans     Incr      Path
-----
clock CLK (rise edge)          0.00      0.00      0.00
clock network delay (ideal)   0.00      0.00      0.00
ffa/CLK (DTC10)               0.00      0.00      0.00 r
ffa/Q (DTC10)                 3.85      0.57      1.70      1.70 f
U7/Y (IV110)                  6.59      1.32      0.84      2.55 r
U12/Y (NA310)                 8.87      2.47      2.04      4.58 f
U17/Y (NA211)                 4.87      1.01      1.35      5.94 f
U23/Y (IV120)                 2.59      0.51      0.37      6.30 r
U15/Y (BF003)                 2.61      0.88      0.82      7.12 f
U16/Y (BF003)                 2.61      1.46      0.99      8.11 r
U10/Y (AN220)                 2.63      0.46      1.04      9.15 r
ffd/D (DTN10)                  0.46      0.00      0.00      9.15 r
data arrival time              0.00      0.00      0.00      9.15

clock CLK (rise edge)          10.00     10.00     10.00
clock network delay (ideal)   0.00      0.00      0.00
ffd/CLK (DTN10)               10.00     10.00     10.00 r
library setup time             -1.33     8.67      8.67
data required time             8.67      8.67      8.67
-----
data required time             8.67
data arrival time              -9.15
-----
slack (VIOLATED)              -0.48
```

Path Timing Calculation

The `report_delay_calculation` command reports the detailed calculation of delay from the input to the output of a cell or from the driver to a load on a net. The type of information you see depends on the delay model you are using.

Many ASIC vendors consider detailed information about cell delay calculation to be proprietary. To protect this information, the `report_delay_calculation` command shows cell delay details only if the library vendor has enabled this feature with the `library_features` attribute in Library Compiler.

Path Specification Methods

The `report_timing` command, the timing exception commands (such as `set_false_path`), and several other commands allow a variety of methods to specify a single path or multiple paths for a timing report or for applying timing exceptions. One way is to explicitly specify the `-from $startpoint` and `-to $endpoint` options in the `report_timing` command for the path, as in the following examples:

```
pt_shell> report_timing -from PORTA -to PORTZ
pt_shell> set_false_path -from FFB1/CP -to FFB2/D
pt_shell> set_max_delay -from REGA -to REGB 12
pt_shell> set_multicycle_path -setup 2 -from FF4 -to FF5
```

Each specified startpoint can be a clock, a primary input port, a sequential cell, a clock input pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has an input delay specified. If you specify a clock, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, the command applies to one path startpoint on that cell.

Each specified endpoint can be a clock, a primary output port, a sequential cell, a data input pin of a sequential cell, or a pin that has an output delay specified. If you specify a clock, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, the command applies to one path endpoint on that cell.

PrimeTime also supports a special form where the startpoint or endpoint becomes a `-through` pin, and a clock object becomes the `-from` or `-to` object. You can use this method with all valid startpoint and endpoint types, such as input ports, output ports, clock flip-flop pins, data flip-flop pins, or clock-gating check pins. For example,

```
pt_shell> report_timing -from [get_clocks ...] -through $startpoint
pt_shell> report_timing -through $endpoint -to [get_clocks ...]
```

Note:

This method is more computationally intensive, especially in larger designs. Whenever possible, use `-from` and `-to` to specify the paths.

You can restrict the `report_timing` command to only rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_from`, `-rise_to`, `-fall_through`, and so on.

Multiple Through Arguments

You can use multiple `-through`, `-rise_through`, and `-fall_through` arguments in a single command to specify a group of paths. For example,

```
pt_shell> report_timing -from A1 -through B1 -through C1 -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1. For example,

```
pt_shell> report_timing -from A1 -through {B1 B2} -through {C1 C2} -to D1
```

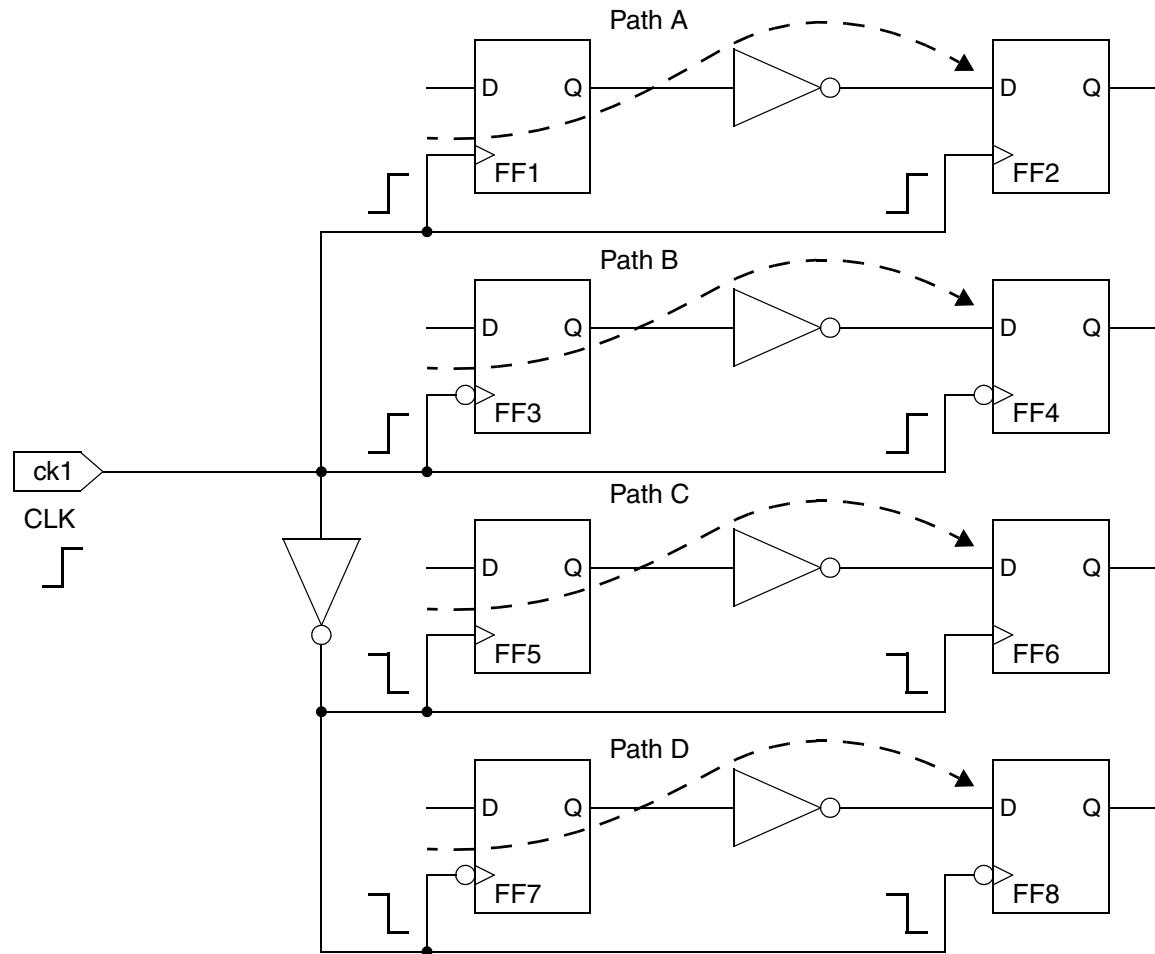
This means any path that starts at A1, passes through either B1 or B2, then passes through either C1 or C2, and ends at D1.

Rise/Fall From/To Clock

You can restrict the command to only to rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_to`, and so on. When the “from” or “to” object is a clock, the command specifies paths based on the launch of data at startpoints or the capture of data at endpoints for a specific edge of the source clock. The path selection considers any logical inversions along the clock path. For example, the `-rise_to clock` option specifies each path clocked by the specified clock at the endpoint, where a rising edge of the source clock captures data at the path endpoint. This can be a rising-edge-sensitive flip-flop without an inversion along the clock path, or a falling-edge-sensitive flip-flop with an inversion along the clock path. The data being captured at the path endpoint does not matter.

The following examples demonstrate the behavior with the `set_false_path` command. Consider the circuit shown in [Figure 7-1](#). FF1, FF2, FF5, and FF6 are rising-edge-triggered flip-flops; and FF3, FF4, FF7, and FF8 are falling-edge-triggered flip-flops. FF1 through FF4 are clocked directly, whereas the FF5 through FF8 are clocked by an inverted clock signal.

Figure 7-1 Circuit for Path Specification in Examples 1 to 3



Example 1

```
pt_shell> set_false_path -to [get_clocks CLK]
```

In [Figure 7-1](#), all paths clocked by CLK at the path endpoint (all four paths) are declared to be false.

Example 2

```
pt_shell> set_false_path -rise_from [get_clocks CLK]
```

In [Figure 7-1](#), all paths clocked by CLK at the startpoint that have data launched by a rising edge of the source clock are declared to be false, so Path A and Path D are false.

Example 3

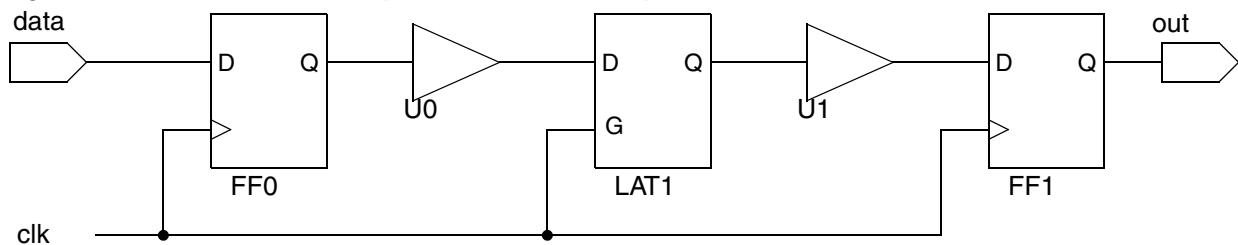
```
pt_shell> set_false_path -fall_to [get_clocks CLK]
```

In [Figure 7-1](#), all paths clocked by CLK at the endpoint that capture data on a falling edge of the source clock are declared to be false, so Path B and Path C are false.

Example 4

Consider the circuit shown in [Figure 7-2](#). The two flip-flops latch data on the positive-going edge of the clock. The level-sensitive latch opens on the rising edge and closes on the falling edge of the clock.

Figure 7-2 Circuit for Path Specification in Example 4



Using `-rise_from clock` option of the `report_timing` command reports the paths with data launched by a rising edge of the clock, from data to FF0, from FF0 to LAT1, from LAT1/G to FF1, and from FF1 to out:

```
pt_shell> report_timing -rise_from [get_clocks clk] -nworst 100
```

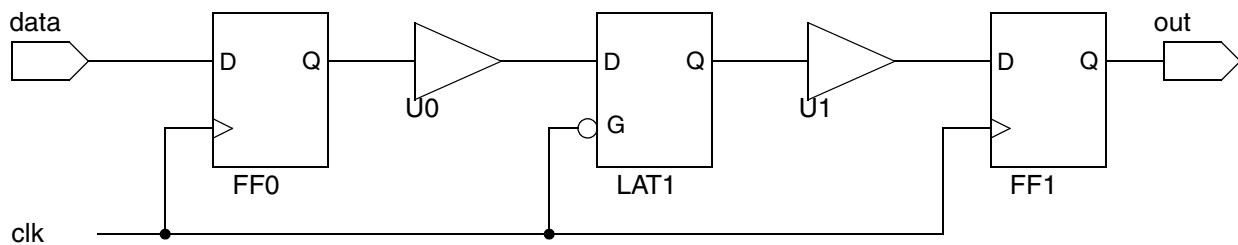
Using the `-fall_to` option of the command instead reports the path with data captured by a falling edge of the clock, which is from FF0 to LAT1:

```
pt_shell> report_timing -fall_to [get_clocks clk] -nworst 100
```

Example 5

The circuit shown in [Figure 7-3](#) is the same as in the previous example, except that the level-sensitive latch opens on the falling edge and closes on the rising edge of the clock.

Figure 7-3 Circuit for Path Specification in Example 5



Using the `-fall_from clock` option of the `report_timing` command reports the paths from LAT1/D to FF1 and from LAT1/G to FF1:

```
pt_shell> report_timing -fall_from [get_clocks clk] -nworst 100
```

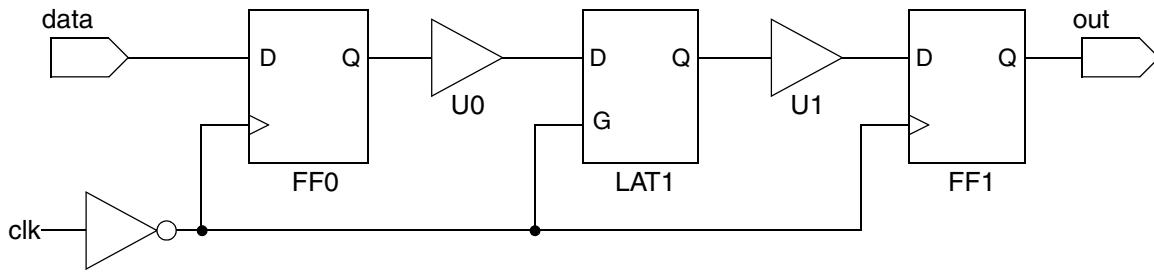
Using the `-rise_to` option of the command reports the paths from data to FF0, from FF0 to LAT1, from LAT1 to FF1, and from FF1 to out:

```
pt_shell> report_timing -rise_to [get_clocks clk] -nworst 100
```

Example 6

The circuit shown in [Figure 7-4](#) is the same as in Example 4 ([Figure 7-2](#)), except that the clock signal is inverted.

Figure 7-4 Circuit for Path Specification in Example 6



Using the `-fall_from clock` option of the `report_timing` command reports the paths with data launched by a falling edge of the clock: from FF0 to LAT1, from LAT1/D to FF1, from LAT1/G to FF1, and from FF1 to out:

```
pt_shell> report_timing -fall_from [get_clocks clk] -nworst 100
```

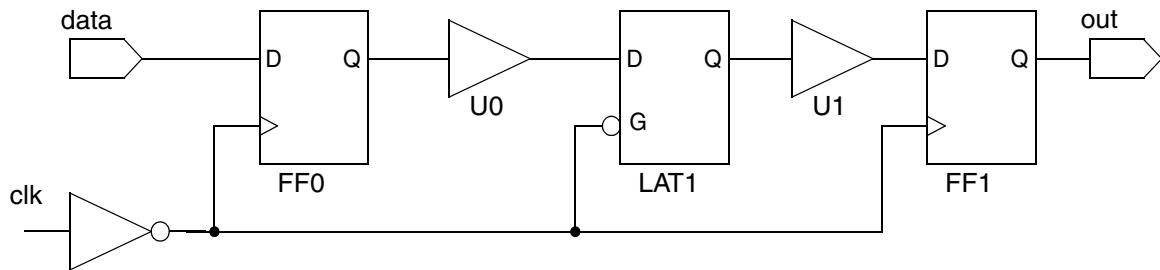
Using the `-rise_to` option of the command reports the path with data captured by a rising edge of the clock, which is from FF0 to LAT1 and from FF1 to out:

```
pt_shell> report_timing -rise_to [get_clocks clk] -nworst 100
```

Example 7

The circuit shown in [Figure 7-5](#) is the same as in Example 5 ([Figure 7-3](#)), except that the clock signal is inverted.

Figure 7-5 Circuit for Path Specification in Example 7



Using the `-rise_from clock` option of the `report_timing` command reports the paths with data launched by a rising edge of the clock: from `data` to `FF0` and from `LAT1/G` to `FF1`:

```
pt_shell> report_timing -rise_from [get_clocks clk] -nworst 100
```

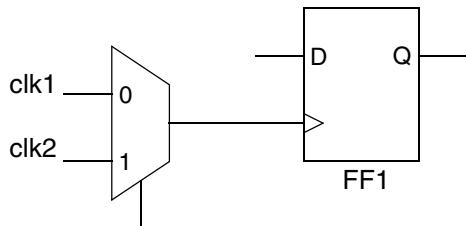
Using the `-fall_to` option of the command reports the paths from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, and from `LAT1/G` to `FF1`:

```
pt_shell> report_timing -fall_to [get_clocks clk] -nworst 100
```

Example 8

In the circuit shown in Figure 7-6, multiple clocks reach the flip-flop.

Figure 7-6 Circuit for Path Specification in Example 8



Suppose that you want to set false path ending at the `D` pin of the flip-flop, but only for `clk1`, not for `clk2`. You can do this by using the `-through` and `-to` options:

```
pt_shell> set_false_path -through FF1/D -to [get_clocks clk1]
```

To set false path ending at the `D` pin of the flip-flop, but only paths that capture data on the rising edge of `clk1`:

```
pt_shell> set_false_path -through FF1/D -rise_to [get_clocks clk1]
```

Reporting of Invalid Startpoints or Endpoints

The `timing_report_always_use_valid_start_end_points` variable specifies how PrimeTime reports invalid startpoints and endpoints. This variable specifies how `report_timing`, `report_bottleneck`, and `get_timing_paths` commands handle invalid startpoints when you use the `-from`, `-rise_from`, and `-fall_from` options. It also specifies how they handle invalid endpoints when you use the `-to`, `-rise_to`, or `-fall_to` options.

When the `timing_report_always_use_valid_start_end_points` variable is set to `false` (the default), the `from_list` is interpreted as all pins within the specified scope of the listed objects. Objects specified in a general way can include many invalid startpoints or endpoints. For example, `-from [get_pins FF/*]` includes input, output, and asynchronous pins. By default, PrimeTime considers these invalid startpoints and endpoints as through points, and continues path searching. It is easy to specify objects this way, but it wastes runtime on useless searching. When this variable is set to `true`, each reporting command ignores invalid startpoints and invalid endpoints.

Irrespective of this variable setting, it is recommended that you always specify valid startpoints (input ports and register clock pins) in the `from_list` and valid endpoints (output ports and register data pins) in the `to_list`.

Timing Exceptions

By default, PrimeTime assumes that data launched at a path startpoint is captured at the path endpoint by the very next occurrence of a clock edge at the endpoint. For paths that are not intended to operate in this manner, you need to specify a timing exception. Otherwise, the timing analysis does not match the behavior of the real circuit.

To learn more about timing exceptions, see

- [Timing Exception Overview](#)
- [Single-Cycle \(Default\) Path Delay Constraints](#)
- [Setting False Paths](#)
- [Setting Maximum and Minimum Path Delays](#)
- [Setting Multicycle Paths](#)
- [Specifying Exceptions Efficiently](#)
- [Exception Order of Precedence](#)
- [Reporting Exceptions](#)
- [Checking Ignored Exceptions](#)

- [Removing Exceptions](#)
- [Transforming Exceptions](#)

Timing Exception Overview

To specify timing exceptions, you can

- Set false paths – To specify a logic path that exists but should not be analyzed, use the `set_false_path` command. Setting a false path removes the timing constraints on the path.
- Set minimum and maximum path delay values – To override the default setup and hold constraints with specific maximum and minimum time values, use the `set_max_delay` and `set_min_delay` commands.
- Set multicycle paths – To specify the number of clock cycles required to propagate data from the start to the end of the path, use the `set_multicycle_path` command.

Each timing exception command can apply to a single path or to a group of related paths, such as all paths from one clock domain to another, or all paths that pass through a specified point. For more information, see [Path Specification Methods](#).

When you specify a path by its startpoint and endpoint, be sure to specify a timing path that is valid in PrimeTime. A path startpoint must be either a register clock pin or an input port. A path endpoint must be either a register data input pin or an output port.

To view a list of timing exceptions that have been applied to a design, use the `report_exceptions` command. You can also preserve source location information, namely the source file and line number when tracking and reporting information for constraints. For more information, see [Reporting Exceptions Source File and Line Number Information](#).

To restore the default timing constraints on a path, use the `reset_path` command.

Single-Cycle (Default) Path Delay Constraints

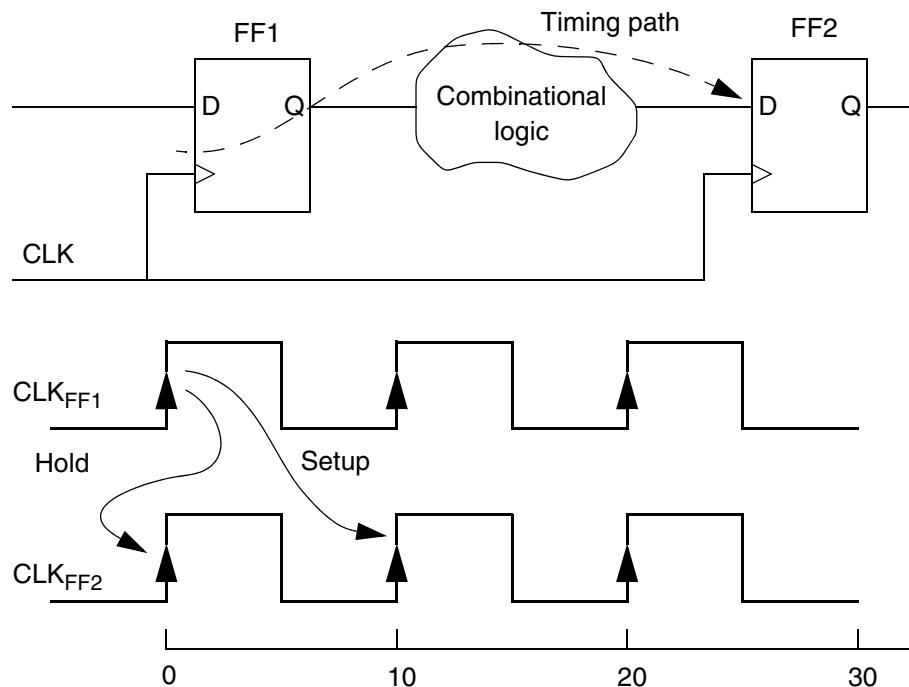
To determine whether you need to set a timing exception, you must understand how PrimeTime calculates maximum and minimum delay times for paths. You must also understand how this calculation is similar or dissimilar to the operation of the design that is being analyzed. For an introduction to static timing analysis principles, see [Overview of Static Timing Analysis](#).

Path Delay for Flip-Flops Using a Single Clock

[Figure 7-7](#) shows how PrimeTime determines the setup and hold constraints for a path that begins and ends on rising-edge-triggered flip-flops. In this example, the two flip-flops are triggered by the same clock. The clock period is 10 ns.

PrimeTime performs a setup check to verify that the data launched from FF1 at time=0 arrives at the D input of FF2 in time for the capture edge at time=10. If the data takes too long to arrive, it is reported as a setup violation.

Figure 7-7 Single-Cycle Setup and Hold for Flip-Flops



Similarly, PrimeTime performs a hold check to verify that the data launched from FF1 at time 0 does not get propagated so soon that it gets captured at FF2 at the clock edge at time 0. If the data arrives too soon, it is reported as a hold violation.

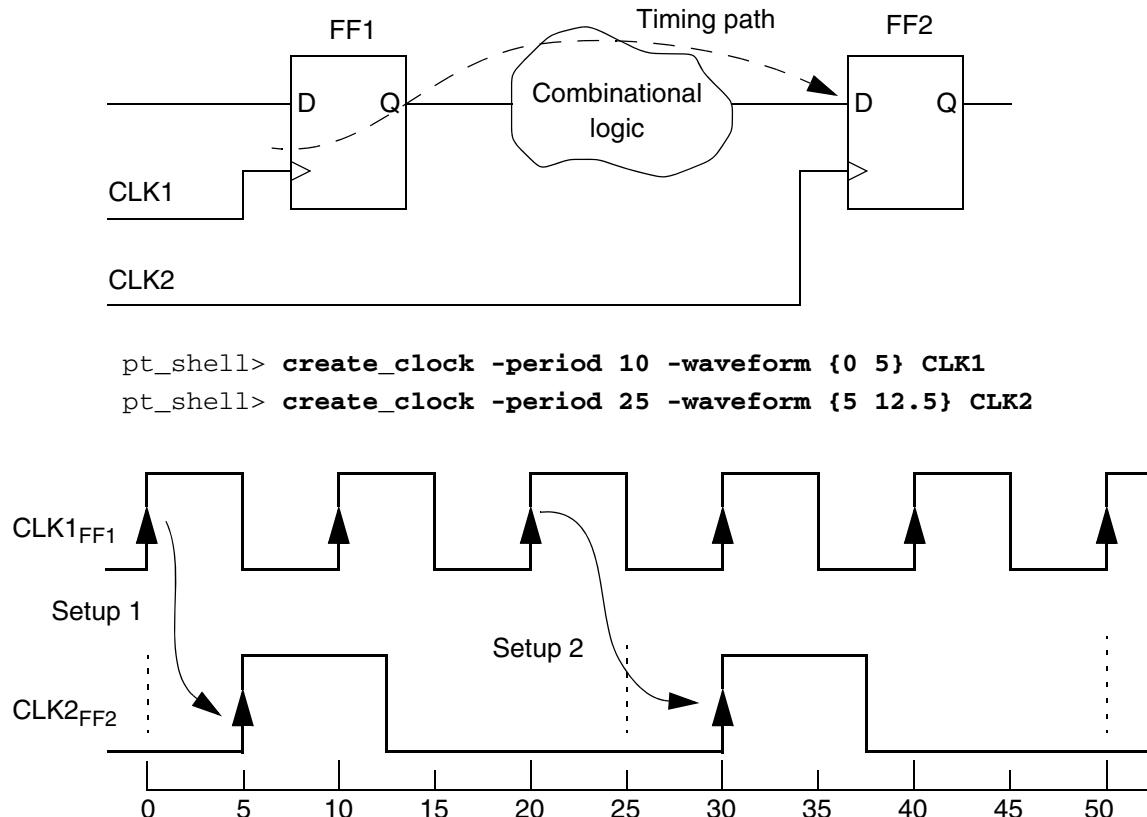
The setup and hold requirements determined by PrimeTime for sequential elements take into account all relevant parameters such as the delay of the combinational logic elements in the path, the setup and hold requirements of the flip-flop elements as defined in the logic library, and the net delays between the flip-flops.

Path Delay for Flip-Flops Using Different Clocks

The algorithm for calculating path delays is more complicated when the launch and capture flip-flops belong to different clock domains.

Consider the circuit shown in [Figure 7-8](#). The flip-flops at the beginning and end of the timing path are clocked by different clocks, CLK1 and CLK2. The two clocks are declared by the `create_clock` commands shown in the figure.

Figure 7-8 Setup Constraints for Flip-Flops With Different Clocks



By default, PrimeTime assumes that the two clocks are synchronous to each other, with a fixed phase relationship. If this is not the case for the real circuit (for example, because the two clocks are never active at the same time or because they operate asynchronously), then you need to declare this fact by using any of several methods. Otherwise, PrimeTime spends time checking constraints and reporting violations that do not exist in the actual circuit.

Setup Analysis

PrimeTime looks at the relationship between the active clock edges over a full repeating cycle, equal to the least common multiple of the two clock periods. For each capture (latch) edge at the destination flip-flop, PrimeTime assumes that the corresponding launch edge is the nearest source clock edge occurring before the capture edge.

In [Figure 7-8](#), there are two capture edges in the period under consideration. For the capture edge at time=5, the nearest preceding launch edge is at time=0. The corresponding setup relationship is labeled Setup 1.

For the capture edge at time=30, the nearest preceding launch edge is at time=20. This setup relationship is labeled Setup 2. The source clock edge at time=30 occurs at the same time as the capture edge, not earlier, so it is not considered the corresponding launch edge.

Setup 1 allows less time between launch and capture, so it is the more restrictive constraint. It determines the maximum allowed delay for the path, which is 5 ns for this example.

Hold Analysis

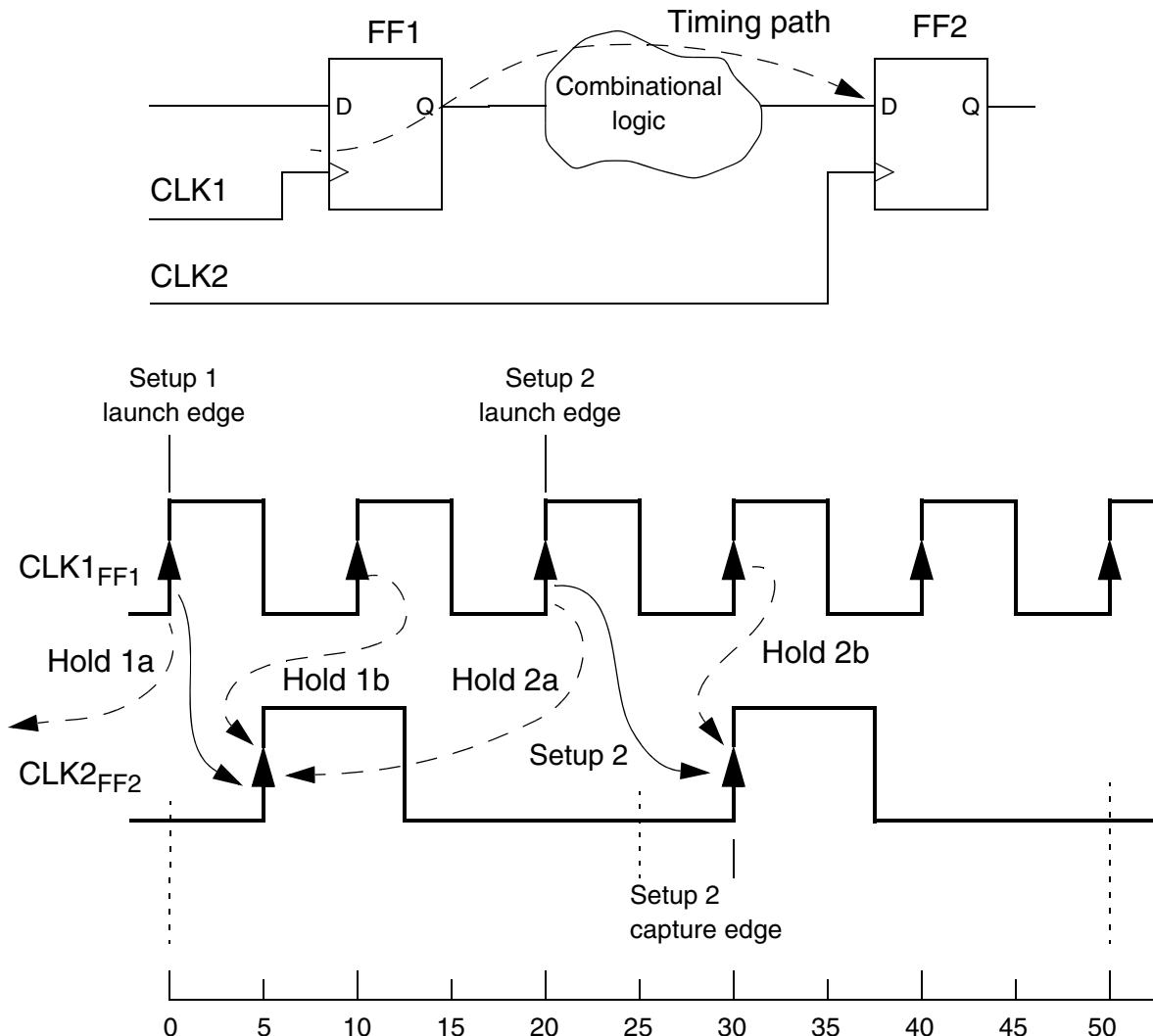
The hold relationships checked by PrimeTime are based on the clock edges adjacent to those used to determine the setup relationships. To determine the most restrictive hold relationship, PrimeTime considers all valid setup relationships, including both Setup 1 and Setup 2 in [Figure 7-8](#).

For each setup relationship, PrimeTime performs two different hold checks:

- The data launched by the setup launch edge must not be captured by the *previous* capture edge.
- The data launched by the *next* launch edge must not be captured by the setup capture edge.

[Figure 7-9](#) shows the hold checks performed by PrimeTime for the current example. First consider the setup relationship labeled Setup 2. PrimeTime confirms that the data launched by the setup launch edge is not captured by the previous capture edge; this relationship is labeled Hold 2a. It also confirms that the data launched by the next clock edge at the source is not captured by the setup capture edge; this relationship is labeled Hold 2b.

Figure 7-9 Hold Constraints for Flip-Flops With Different Clocks



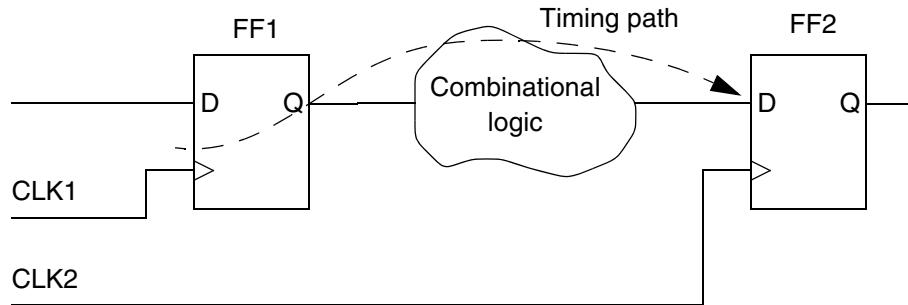
PrimeTime similarly checks the hold relationships relative to the clock edges of Setup 1, as indicated in the figure. The most restrictive hold check is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b in this example. Therefore, Hold 2b determines the minimum allowed delay for this path, 0 ns.

Single-Cycle Path Analysis Examples

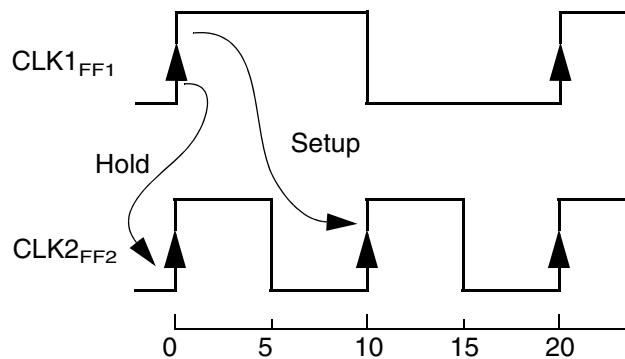
The following examples further show how PrimeTime calculates the delay requirements for edge-triggered flip-flops in the absence of timing exceptions.

In Figure 7-10, CLK1 has a period of 20 ns and CLK2 has a period of 10 ns. The most restrictive setup relationship is the launch edge at time=0 to the capture edge at time=10. The most restrictive hold relationship is the launch edge at time=0 to the capture edge at time=0.

Figure 7-10 Delay Requirements With 20ns/10ns Clocks

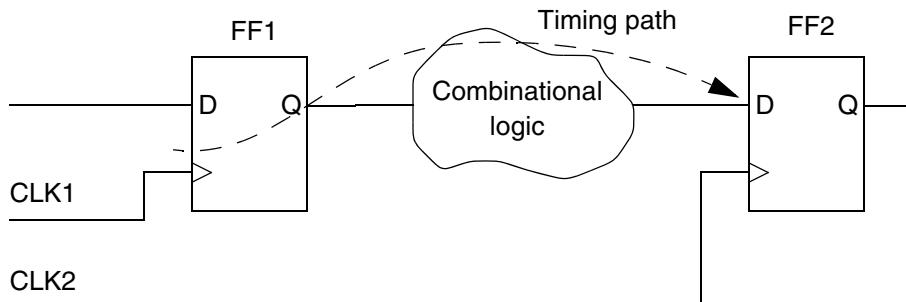


```
pt_shell> create_clock -period 20 -waveform {0 10} CLK1
pt_shell> create_clock -period 10 -waveform {0 5} CLK2
```

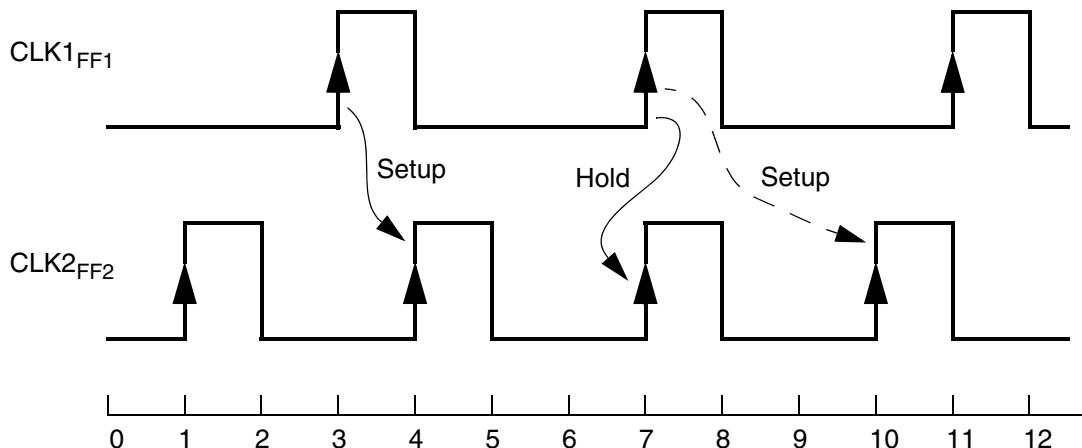


In Figure 7-11, CLK1 has a period of 4 ns and CLK2 has a period of 3 ns. The most restrictive setup relationship is the launch edge at time=3 to the capture edge at time=4. The most restrictive hold relationship is the launch edge at time=7 to the capture edge at time=7, based on the setup relationship shown by the dashed-line arrow in the timing diagram.

Figure 7-11 Delay Requirements With 4ns/3ns Clocks



```
pt_shell> create_clock -period 4 -waveform {3 4} CLK1
pt_shell> create_clock -period 3 -waveform {1 2} CLK2
```



Setting False Paths

A false path is a logic path that exists but should not be analyzed for timing. For example, a path can exist between two multiplexed logic blocks that are never selected at the same time, so that path is not valid for timing analysis.

For example, to declare a false path from pin FFB1/CP to pin FFB2/D:

```
pt_shell> set_false_path -from [get_pins FFB1/CP] \
           -to [get_pins FFB2/D]
```

Declaring a path to be false removes all timing constraints from the path. PrimeTime still calculates the path delay, but does not report it to be an error, no matter how long or short the delay.

Setting a false path is a point-to-point timing exception. This is different from using the `set_disable_timing` command, which disables timing analysis for a specified pin, cell, or port. Using the `set_disable_timing` command removes the affected objects from timing analysis, rather than removing the timing constraints from the paths. If all paths through a pin are false, using `set_disable_timing [get_pins pin_name]` is more efficient than using `set_false_path -through [get_pins pin_name]`.

Another example of a false path is a path between flip-flops belonging to two clock domains that are asynchronous with respect to each other.

To declare all paths between two clock domains to be false, you can use a set of two commands such as the following:

```
pt_shell> set_false_path -from [get_clocks ck1] \
           -to [get_clocks ck2]

pt_shell> set_false_path -from [get_clocks ck2] \
           -to [get_clocks ck1]
```

For efficiency, be sure to specify each clock by its clock name, not by the pin name (use `get_clocks`, not `get_pins`).

An alternative is to use the `set_clock_groups` command to exclude paths from consideration that are launched by one clock and captured by another. Although this has the same effect as declaring a false path between the two clocks, it is not considered a timing exception and is not reported by the `report_exceptions` command.

Setting Maximum and Minimum Path Delays

By default, PrimeTime calculates the maximum and minimum path delays by considering the clock edge times. To override the default maximum or minimum time with your own specific time value, use the `set_max_delay` or `set_min_delay` command. For example, to set the maximum path delay between registers REGA and REGB to 12, use this command:

```
pt_shell> set_max_delay 12 \
           -from [get_cells REGA] -to [get_cells REGB]
```

With this timing exception, PrimeTime ignores the clock relationships. A path delay between these registers that exceeds 12 time units minus the setup requirement of the endpoint register is reported as a timing violation. Similarly, to set the minimum path delay between registers REGA and REGB to 2, use this command:

```
pt_shell> set_min_delay 2.0 \
           -from [get_cells REGA] -to [get_cells REGB]
```

Again, PrimeTime ignores the clock relationships. A path delay between these registers that is less than 2 time units plus the hold requirement of the endpoint register is reported as a

timing violation. You can optionally specify that the delay value apply only to rising edges or only to falling edges at the endpoint.

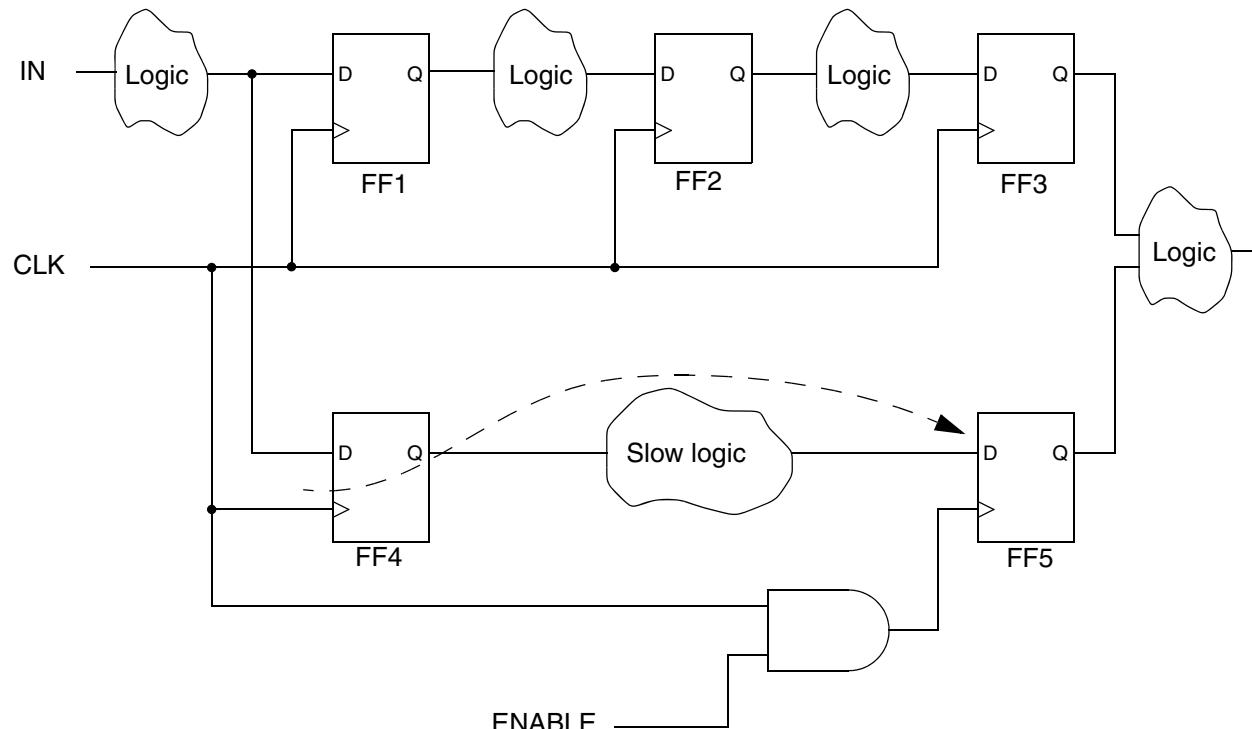
Be sure to select a valid path startpoint with the `-from` or similar option and a valid path endpoint with the `-to` or similar option unless you want to override the paths checked by PrimeTime. If you specify an invalid startpoint or endpoint, PrimeTime performs the timing check exactly as specified and ignores the remaining valid portion of the timing path.

Applying such an exception on a clock path prevents propagation of the clock forward from the point where the exception is applied. PrimeTime issues a UITE-217 warning message when you specify an invalid startpoint or endpoint.

Setting Multicycle Paths

The `set_multicycle_path` command specifies the number of clock cycles required to propagate data from the start of a path to the end of the path. PrimeTime calculates the setup or hold constraint according to the specified number of cycles. For example, consider the circuit shown in [Figure 7-12](#). The path from FF4 to FF5 is designed to take two clock cycles rather than one. However, by default, PrimeTime assumes single-cycle timing for all paths. Therefore, you need to specify a timing exception for this path.

Figure 7-12 Multicycle Path Example



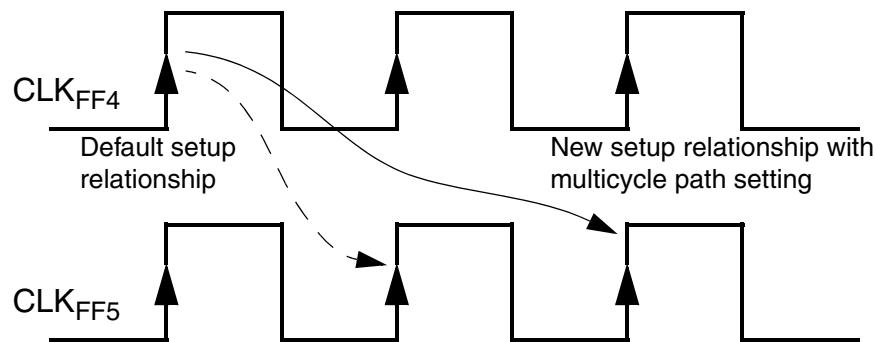
One timing exception method is to specify an explicit maximum delay value with the `set_max_delay` command. However, you might want to use the `set_multicycle_path` command instead because the maximum delay value is automatically adjusted when you change the clock period.

To set the multicycle path for the design shown in [Figure 7-12](#), use this command:

```
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]
```

The first command tells PrimeTime that the path takes two clock cycles rather than one, establishing the new setup relationship shown in [Figure 7-13](#). The second capture edge (rather than the first) following the launch edge becomes the applicable edge for the end of the path.

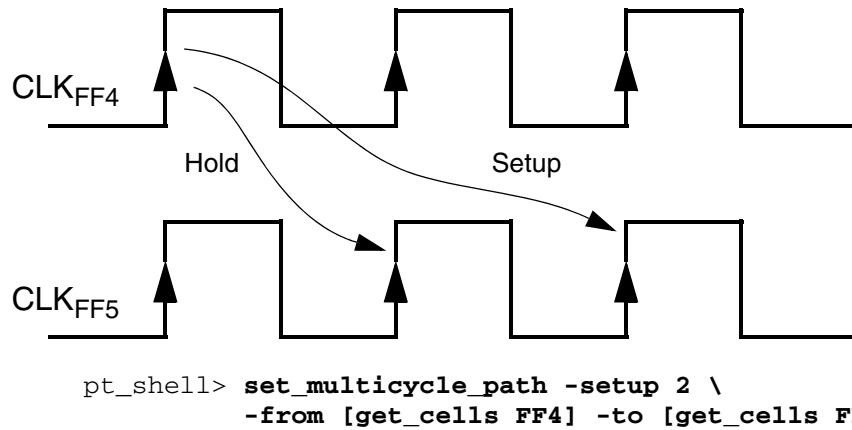
Figure 7-13 Multicycle Path Setup



```
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]
```

Changing the setup relationship implicitly changes the hold relationship as well because all hold relationships are based on the valid setup relationships. PrimeTime verifies that the data launched by the setup launch edge is not captured by the previous capture edge. The new hold relationship is shown in [Figure 7-14](#).

Figure 7-14 Multicycle Path Hold Based on New Setup



The hold relationship shown in [Figure 7-14](#) is probably not the correct relationship for the design. If FF4 does not need to hold the data beyond the first clock edge, you need to specify another timing exception.

Although you could use the `set_min_delay` command to specify a particular hold time, it is better to use another `set_multicycle_path` command to move the capture edge for the hold relationship backward by one clock cycle. For example,

```

pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]

pt_shell> set_min_delay 0 -from [get_cells FF4] -to [get_cells FF5]

```

or preferably:

```

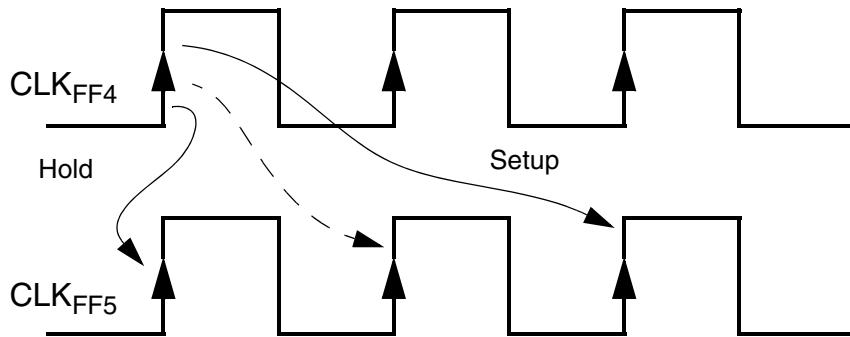
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]

pt_shell> set_multicycle_path -hold 1 \
           -from [get_cells FF4] -to [get_cells FF5]

```

[Figure 7-15](#) shows the setup and hold relationships set correctly with two `set_multicycle_path` commands. The second `set_multicycle_path` command moves the capture edge of the hold relationship backward by one clock cycle, from the dashed-line arrow to the solid-line arrow.

Figure 7-15 Multicycle Path Hold Set Correctly



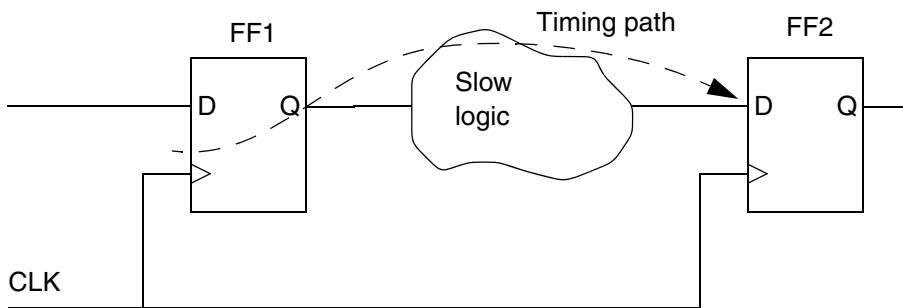
```
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]
```

```
pt_shell> set_multicycle_path -hold 1 \
           -from [get_cells FF4] -to [get_cells FF5]
```

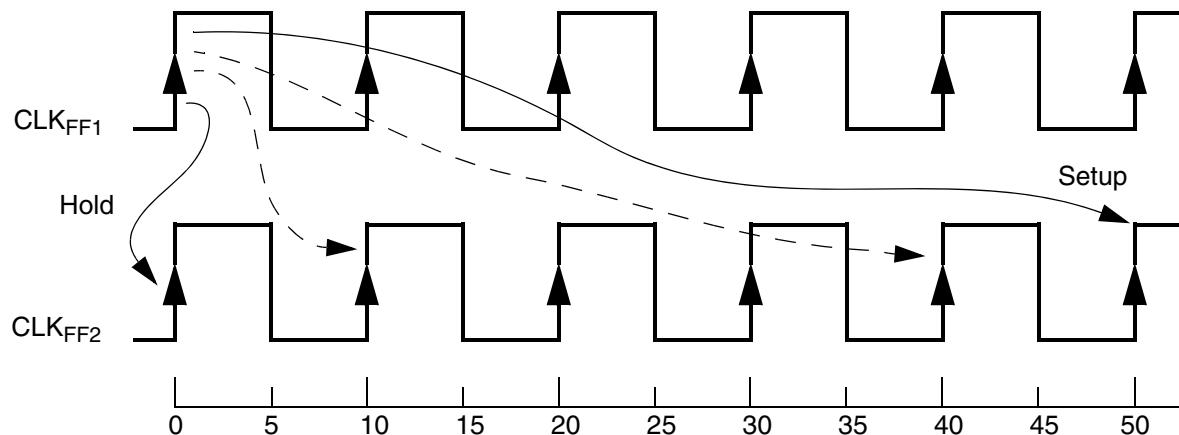
Note that PrimeTime interprets the `-setup` and `-hold` values in the `set_multicycle_path` command differently. The integer value for the `-setup` argument specifies the number of clock cycles for the multicycle path. In the absence of a timing exception, the default is 1. The integer value for the `-hold` argument specifies the number of clock cycles to move the capture edge backward with respect to the default position (relative to the valid setup relationship); the default setting is 0.

The example shown in [Figure 7-16](#) further demonstrates the setting of multicycle paths. In the absence of timing exceptions, the setup relationship is from time=0 to time=10, as indicated by the dashed-line arrow, and the hold relationship is from time=0 to time=0.

Figure 7-16 Multicycle Path Taking Five Clock Cycles



```
pt_shell> create_clock -period 10 -waveform {0 5} CLK1
```



```
pt_shell> set_multicycle_path -setup 5 \
    -from [get_cells FF1] -to [get_cells FF2]
```

```
pt_shell> set_multicycle_path -hold 4 \
    -from [get_cells FF1] -to [get_cells FF2]
```

With `set_multicycle_path -setup 5`, the setup relationship spans five clock cycles rather than one, from time=0 to time=50, as shown by the long solid-line arrow. This implicitly changes the hold relationship to the prior capture edge at time=40, as shown by the long dashed-line arrow.

To move the capture edge for the hold relationship back to time=0, you need to use `set_multicycle_path -hold 4` to move the capture edge back by four clock cycles.

To summarize, PrimeTime determines the number of hold cycles as follows:

$$(\text{hold cycles}) = (\text{setup option value}) - 1 - (\text{hold option value})$$

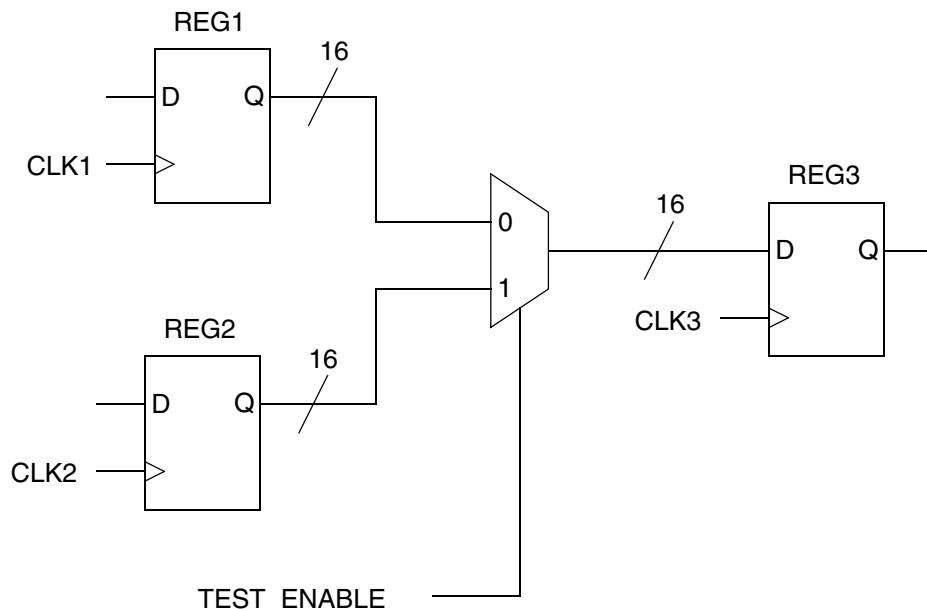
By default, hold cycles = $1 - 1 - 0 = 0$. For [Figure 7-15](#), hold cycles = $2 - 1 - 1 = 0$. For [Figure 7-16](#), hold cycles = $5 - 1 - 4 = 0$.

You can optionally specify that the multicycle path exception apply only to rising edges or only with falling edges at the path endpoint. If the startpoint and endpoint are clocked by different clocks, you can specify which of the two clocks is considered for adjusting the number of clock cycles for the path.

Specifying Exceptions Efficiently

In many cases, you can specify the exception paths many different ways. Choosing an efficient method can reduce the analysis runtime. For example, consider the circuit shown in [Figure 7-17](#). The three 16-bit registers are clocked by three different clocks. Each register represents 16 flip-flops. Register REG2 is only used for test purposes, so all paths from REG2 to REG3 are false paths during normal operation of the circuit.

Figure 7-17 Multiplexed Register Paths



To prevent analysis of timing from REG2 to REG3, you can use any of the following methods:

- Use case analysis to consider the case when the test enable signal is 0. (See [Case and Mode Analysis](#).)

- Set an exclusive relationship between the CLK1 and CLK2 clock domains. (See [Exclusive Clocks](#).)
- Declare the paths between clock domains CLK2 and CLK3 to be false.

```
pt_shell> set_false_path \
           -from [get_clocks CLK2] -to [get_clocks CLK3]
```

This method is an efficient way to specify the false paths because PrimeTime only needs to keep track of the specified clock domains. It does not have to keep track of exceptions on registers, pins, nets, and so on.

- Declare the 16 individual paths from REG2 to REG3 to be false.

```
pt_shell> set_false_path -from [get_pins REG2[0]/CP] \
           -to [get_pins REG3[0]/D]
```

```
pt_shell> set_false_path -from [get_pins REG2[1]/CP] \
           -to [get_pins REG3[1]/D]
```

```
pt_shell> ...
```

This method is less efficient because PrimeTime must keep track of timing exceptions for 16 different paths.

- Declare all paths from REG2 to REG3 to be false.

```
pt_shell> set_false_path -from [get_pins REG2[*]/CP] \
           -to [get_pins REG3[*]/D]
```

This method is even less efficient because PrimeTime must keep track of paths from each clock pin of REG2 to each data pin of REG3, a total of 256 paths.

- Declare all paths from REG2 to be false.

```
pt_shell> set_false_path -from [get_pins REG2[*]/CP]
```

This method is similar to the previous one. PrimeTime must keep track of all paths originating from each clock pin of REG2, a total of 256 paths.

In summary, look at the root cause that is making the exceptions necessary and find the simplest way to control the timing analysis for the affected paths. Before using false paths, consider using case analysis (`set_case_analysis`), declaring an exclusive relationship between clocks (`set_clock_groups`), or disabling analysis of part of the design (`set_disable_timing`). These alternatives can be more efficient than using the `set_false_path` command.

If you must set false paths, avoid specifying a large number of paths using the `-through` argument, by using wildcards, or by listing the paths one at a time. After you set false paths and other timing exceptions, you might be able to simplify the set of exception-setting commands by using the `transform_exceptions` command. For more information, see [Transforming Exceptions](#).

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception PrimeTime uses for that path depends on the exception types or the path specification methods used in the conflicting commands. A set of rules establishes the order of priority for different exception-setting situations.

Note that PrimeTime applies the exception precedence rules independently on each path (not each command). For example, suppose that you use these commands:

```
pt_shell> set_max_delay -from A 5.1
pt_shell> set_false_path -to B
```

The `set_false_path` command has priority over the `set_max_delay` command, so any paths that begin at A and end at B are false paths. However, the `set_max_delay` command still applies to paths that begin at A but do not end at B.

Exception Type Priority

The following pairs of timing exception types are not considered to be in conflict, so both settings can be valid for a path:

- Two `set_false_path` settings
- `set_min_delay` and `set_max_delay` settings
- `set_multicycle_path -setup` and `-hold` settings

In case of conflicting exceptions for a particular path, the timing exception types have the following order of priority, from highest to lowest:

1. `set_false_path`
2. `set_max_delay` and `set_min_delay`
3. `set_multicycle_path`

For example, if you declare a path to be false and also set its maximum delay to some value, the false path declaration has priority. The maximum delay setting is ignored. You can list ignored timing exceptions by using the `report_exceptions -ignored` command.

Path Specification Priority

If you apply the same type of timing exception using commands with different path specifications, the more specific command has priority over the more general one. Exceptions of any type on more specific objects, such as pins or ports, take precedence over exceptions applied to more general objects, such as clocks. For example,

```
pt_shell> set_max_delay 12 -from [get_clocks CLK1]  
pt_shell> set_max_delay 15 -from [get_clocks CLK1] -to [get_clocks CLK2]
```

The first command sets the maximum delay of all paths starting from CLK1. However, the second command is more specific, so it overrides the first command for paths starting at CLK1 and ending at CLK2. The remaining paths starting from CLK1 are still controlled by the first command.

The various `-from/-to` path specification methods have the following order of priority, from highest to lowest:

1. `-from pin, -rise_from pin, -fall_from pin`
2. `-to pin, -rise_to pin, -fall_to pin`
3. `-through, -rise_through, -fall_through`
4. `-from clock, -rise_from clock, -fall_from clock`
5. `-to clock, -rise_to clock, -fall_to clock`

Use the preceding list to determine which of two conflicting timing exception commands has priority (for example, two `set_max_delay` commands). Starting from the top of the list:

1. A command containing `-from pin, -rise_from pin, or -fall_from pin` has priority over a command that does not contain `-from pin, -rise_from pin, or -fall_from pin`.
2. A command containing `-to pin, -rise_to pin, or -fall_to pin` has priority over a command that does not contain `-to pin, -rise_to pin, or -fall_to pin`.

... and so on down the list until the priority is resolved.

Here are some possible path specification combinations, listed in order of priority from highest to lowest, according to the preceding priority rules:

1. `-from pin -to pin`
2. `-from pin -to clock`
3. `-from pin`
4. `-from clock -to pin`

5. -to pin
6. -from clock -to clock
7. -from clock
8. -to clock

Reporting Exceptions

To report timing exceptions that have been set, use the `report_exceptions` command. You can reduce the scope of the report by using the path specification arguments `-from`, `-to`, `-through`, `-rise_from`, `-fall_to`, and so on, to match the path specifiers used when the original exceptions were created.

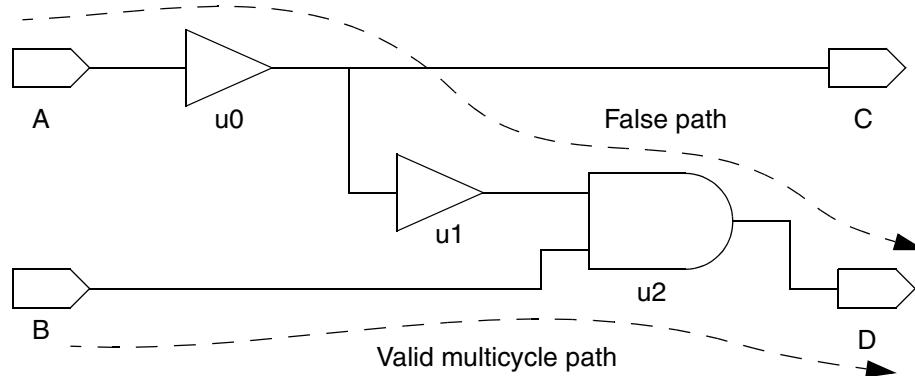
The `report_exceptions` command causes a complete timing update, so be sure to use it only after you have set up all timing assertions and you are ready to receive the report.

Exception-setting commands are sometimes partially or fully ignored for a number of reasons. For example, a command that specifies a broad range of paths can be partially overridden by another exception-setting command that specifies a subset of those paths. For a partially ignored command, the exception report shows the reason that some part of the command is being ignored.

By default, a command that is fully ignored is not reported by the `report_exceptions` command. To report the commands that are fully ignored, use the `-ignored` option with the `report_exceptions` command.

Consider the design shown in [Figure 7-18](#) and the following exception-setting and exception-reporting commands.

Figure 7-18 Design to Demonstrate Ignored Exceptions



```
pt_shell> set_false_path -from A -to D
pt_shell> set_multicycle_path 2 -from A -to D
```

```
pt_shell> set_multicycle_path 2 -from {A B C} -to D
pt_shell> report_exceptions
```

...

Reasons:

- f - invalid startpoint(s)
- t - invalid endpoint(s)
- p - non-existent paths
- o - overridden paths

From	To	Setup	Hold	Ignored
A	D	FALSE	FALSE	
{ A B C }	D	2	*	f,o

```
pt_shell> report_exceptions -ignored
```

...

From	To	Setup	Hold	Ignored
A	D	2	*	o

The first exception-setting command sets a false path from port A to port D. This is the highest-priority command, so it is fully enforced by PrimeTime.

The second exception-setting command attempts to set a multicycle path from port A to port D. It is fully ignored because it is overridden by the higher-priority false path command. Because this command is fully ignored, it is reported only when you use the `-ignored` option of the `report_exceptions` command. In the report, the letter code “o” in the “Ignored” column shows the reason the command is being ignored.

The third exception-setting command attempts to set multicycle paths from ports A to D, B to D, and C to D. It is partially valid (not fully ignored), so it is reported by the `report_exceptions` command without the `-ignored` option. The path from A to D is ignored because it is overridden by the false path command. The path from B to D is valid, so that multicycle path is enforced by PrimeTime. The path from C to D is invalid because port C is not a valid startpoint for a path. In the report, the letter codes in the “Ignored” column indicate the reasons that some of the paths specified in the command are being ignored.

Reporting Timing Exceptions

The `report_timing` command has an `-exceptions` option that allows you to report the timing exceptions that apply to an individual path. You can choose to report the following:

- Exceptions that apply to a path
- Exceptions that were overridden by higher-priority exceptions

- Unconstrained path debugging that includes unconstrained startpoint and unconstrained endpoint
- Timing path attributes that show the unconstrained reasons

When using either the `report_timing -exceptions all` or `get_timing_paths` command to report why timing paths are unconstrained, you must first set the `timing_report_unconstrained_paths` variable to `true`. As shown earlier, to report the unconstrained reasons, you can use the `report_timing` command with the three options:

```
pt_shell> report_timing -exceptions dominant
pt_shell> report_timing -exceptions overridden
pt_shell> report_timing -exceptions all
```

In the following example where you specify conflicting exceptions, the maximum-delay exception has higher priority:

```
pt_shell> set_multicycle_path -through buf1/Z -setup 2
pt_shell> set_max_delay -through buf1/Z 1
```

If you use `report_timing -exceptions dominant` report the timing of the path containing `buf1/Z`, the report includes a section showing the dominant timing exception that affected the path:

The dominant exceptions are:				
From	Through	To	Setup	Hold
*	buf1/Z	*	max=1	

If you use `report_timing -exceptions overridden`, the report includes a section showing the overridden timing exception that had no effect on the timing calculation:

The overridden exceptions are:				
From	Through	To	Setup	Hold
*	buf1/Z	*	cycles=2	*

If you use `report_timing -exceptions all`, the timing report includes a section showing both the dominant and overridden timing exceptions. Alternatively, you can use the `get_timing_paths` command to create a collection of timing paths for custom reporting or for other processing purposes. You can then pass this timing path collection to the `report_timing` command. By adding the `-exceptions all` argument, you obtain the additional debugging information. For example,

```
report_timing -from A -through B -to C -exceptions all
set x [get_timing_paths -from A -through B -to C]
report_timing $x -exceptions all
```

With the `get_timing_paths` command, you can access the path attributes that show the reasons why the path is unconstrained. [Table 7-1](#) shows the timing path attributes along with their possible values, in relation to unconstrained paths:

Table 7-1 Timing Path Attributes and Values

Timing path attribute	Reason code	Reason code description
dominant_exception	false_path	False paths that are not considered.
	min_delay, max_delay	Timing path is a minimum or maximum delay.
	multicycle_path	Multicycle path.
endpoint_unconstrained_reason	no_capture_clock	No clock is capturing the data at the endpoint.
	dangling_end_point	Timing path is broken by a disabled timing component because it ends at a dangling (floating) point that has no timing constraints information.
startpoint_unconstrained_reason	fanin_of_disabled	Path ending at a fanin of a disabled timing arc or component. The internal pin is either without constrains, pin connected to a black box, or there are unconnected pins. The endpoint of the timing path is part of the fanin of a disabled timing arc or component.
	no_launch_clock	No clock is launching the data from the startpoint.
	dangling_start_point	Path starting from a dangling pin. The timing path is broken by a disabled timing component because it starts from a dangling (floating) point that has no timing constraints information.
fanout_of_disabled		Path starting from a fanout of a disabled timing arc or component. The internal pin is without constrains, pin connected to a black box, or there are unconnected pins. The startpoint of the timing path is part of the fanout of a disabled timing arc or component.

The `report_timing -exceptions` command always shows three categories: the dominant exception, startpoint unconstrained reason, and endpoint unconstrained reason. If no exception information is present in a category, that section is empty.

The exception attributes on the `timing_paths` objects (`dominant_exception`, `startpoint_unconstrained_reason`, and `endpoint_unconstrained_reason`) are only present on a path if information is present in that category. This makes it easy to filter for the presence or absence of exception information affecting a path. For example, to filter all paths from a collection that are affected by exceptions:

```
filter_collection $paths {defined(dominant_exception)}
filter_collection $paths {undefined(dominant_exception)}
```

The following report shows the output of `report_timing -exceptions all`:

```
pt_shell> report_timing -exceptions all
*****
Report : timing
          -path_type full_clock_expanded
...
  (Path is unconstrained)

The dominant exceptions are:
From      To      Setup      Hold
-----
1r6_ff/CP    cr6_ff/D    FALSE    FALSE

The overridden exceptions are:
From      To      Setup      Hold
-----
F1/CP      F2/D      cycles=3    cycles=0

The unconstrained reasons (except for false path) are:
Reason            Startpoint            Endpoint
-----
no_launch_clock      F1/CP                  -
```

Reporting Exceptions Source File and Line Number Information

For some constraints, PrimeTime can track and report the source file and line number information for the constraints. Source files are read into the PrimeTime shell using the `source` or `read_sdc` commands or the `-f` command line option.

To enable the source file name and line number information for the current design constraints, set the `sdc_save_source_file_information` variable to `true`. By default, this variable is set to `false`.

Note:

You can modify the value of this variable only if you have not yet input exceptions.

This implies that you can set the variable to `true` in the setup file or inside `pt_shell` before applying a timing exception. If at least one exception command has already been successfully input when you try to set this variable, you receive an error and the value remains unchanged. For example,

```
pt_shell> echo $sdc_save_source_file_information
false
pt_shell> set_max_delay -to port1 0.7
pt_shell> set sdc_save_source_file_information true
Error: can't set "sdc_save_source_file_information":
        Use error_info for more info. (CMD-013)
pt_shell> echo $sdc_save_source_file_information
false
```

Currently, the scope of source-location tracking applies to the following timing exceptions commands:

- `set_false_path`
- `set_multicycle_path`
- `set_max_delay`
- `set_min_delay`

To report the source of the constraints, use the `report_exceptions` or `report_timing_exceptions` commands. Consider the following:

- Commands entered interactively do not have source location information.
- For commands that are input inside control structures, such as `if` statements and `foreach` loops, the line number of the closing bracket is reported.
- For commands that are input inside procedure calls, the line number invoking the procedure is reported.

[Example 7-1](#) and [Example 7-2](#) show examples of exception reports containing the source file and line number information.

Example 7-1 Exceptions Report

```
pt_shell> report_exceptions
*****
Report : exceptions
...
*****
Reasons : f invalid start points
           t invalid endpoints
           p non-existent paths
           o overridden paths
From      To          Setup      Hold      Ignored
-----
a[1]      lar5_ff1/D  cycles=3  *        [ location = multi.tcl:17 ]
a[2]      lar5_ff2/D  cycles=4  *        [ location = multi.tcl:18 ]
data[16]   lr16_ff/D  FALSE     FALSE    [location=scripts/false_path.tcl:11]
```

Example 7-2 Timing Report With the -exceptions Option

```
pt_shell> report_timing -exceptions all -from a[1] -to lar5_ff1/D
```

```
*****
Report : timing
  -path_type full
  -delay_type max
  -max_paths 1
  -exceptions all
...
*****
...
The dominant exceptions are:
From      To          Setup          Hold
-----
a[1]      lar5_ff1/D  cycles=3      cycles=0
          [ location = multi.tcl:17 ]

The overridden exceptions are:
  None
```

Checking Ignored Exceptions

A timing exception that you entered, but is not accepted by PrimeTime is called an ignored exception. There are several reasons an exception is ignored:

- Specified startpoint or endpoint is not valid. The startpoint must be a register clock pin or input port. The endpoint must be a register data input pin or output port.
- Specified path is not constrained (for example, the startpoint is an input port with no input delay set, or the capture flip-flop at the endpoint is not clocked by a defined clock signal).

- Path is invalid because of `set_disable_timing`, constant propagation, loop breaking, or case analysis.
- Exception has a lower priority than another exception applied to the same path.

The `report_exceptions` command reports exceptions that are fully and partially valid. To report all fully ignored exceptions, use `report_exceptions -ignored`. It is a good idea to examine these reports to confirm that you have specified all exceptions correctly.

If ignored exceptions are reported, determine the cause and correct them by changing the path specifications or removing the exception-setting commands. Large numbers of ignored exceptions can increase memory usage and analysis runtime.

If the reason that an exception is partially or fully ignored is not immediately apparent, check the reasons listed in the “Ignored” column of the exception report and consider the possible causes listed earlier. It might be helpful to get more information using the following commands:

```
transform_exceptions -dry_run  
report_exceptions -ignored
```

To find out if there is a logical path between two points, use this command:

```
all_fanout -from point_a -to point_b
```

After a timing update, to examine the path timing, use this command:

```
report_timing -from point_a -to point_b
```

To convert the current set of exception-setting commands to a simpler, equivalent set of commands, use the `transform_exceptions` command. For an example of a report on ignored exceptions, see [Figure 7-18](#).

Removing Exceptions

To remove a timing exception previously set with `set_false_path`, `set_max_delay`, `set_min_delay`, or `set_multicycle_path`, use the `reset_path` command.

You control the scope of exception removal in the `reset_path` command by specifying `-from`, `-to`, `-through`, `-rise_from`, `-fall_to`, and so on. The path specification and

object (such as pin, port, or clock) must match the original path specification and object used to set the exception. Otherwise, the `reset_path` command has no effect. For example,

```
pt_shell> set_false_path -from [get_clocks CLK]
pt_shell> reset_path -from [get_pins ff1/CP]
          # ff1 clocked by CLK

pt_shell> set_false_path -through [get_pins {d/z g/z}]
pt_shell> reset_path -through [get_pins a/z]
          # where a fans out to d
```

In each of these two examples, the object in the `reset_path` command does not match the original object, so the paths are not reset, even though they might have exceptions applied.

You can use the `-reset_path` option in an exception-setting command to reset all of the exceptions set on a path before the new exception is applied. For example,

```
pt_shell> set_false_path -through [get_pins d/z] -reset_path
```

This example first resets all timing exceptions previously applied to the paths through the specified point, then applies the false path exception to these paths.

To remove all exceptions from the design, you can use the `reset_design` command, which removes all user-specified clocks, path groups, exceptions, and attributes (except those defined with the `set_user_attribute` command).

Transforming Exceptions

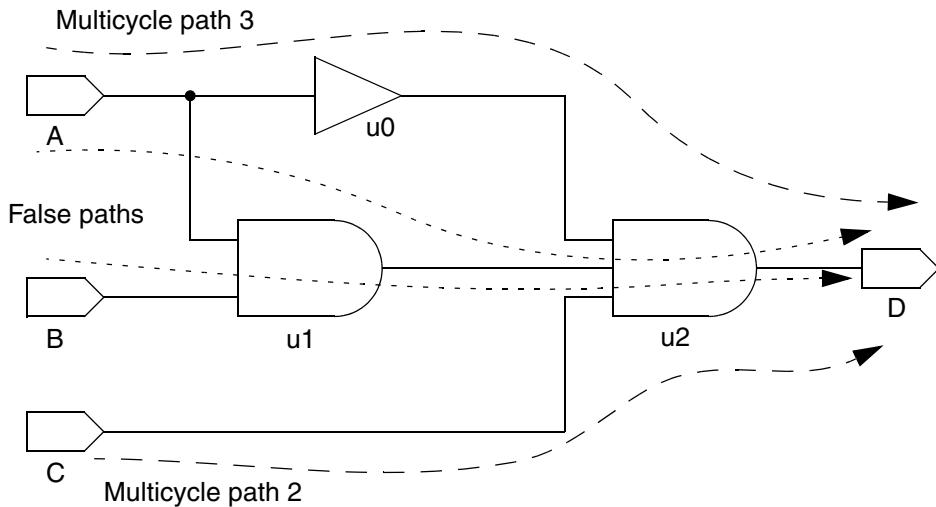
Exceptions that you set are sometimes partially or completely ignored for various reasons. Multiple exception-setting commands with overlapping path specifications can make it difficult to understand the scope of each command.

To gain a better understanding of these exceptions, you can use the `report_exceptions` command. The generated report indicates the reasons that paths were ignored for each exception-setting command, based on the exceptions originally entered by you. However, the report does not always make it clear what exceptions apply to a particular path.

The `transform_exceptions` command has the ability to transform the current set of exceptions into a new, simpler set of equivalent exception-setting commands. The command can be used to get information about the exceptions that apply to a particular path or group of related paths, or to transform a complex set of exception-setting commands into a simpler, equivalent set of commands.

The design and exception-setting commands shown in [Figure 7-19](#) demonstrate some of the principles of exception transformation.

Figure 7-19 Design to Demonstrate Transformed Exceptions



```
pt_shell> set_false_path -through [get_pins u1/Z]
pt_shell> set_multicycle_path 2 \
           -from [get_ports A] -to [get_ports D]

pt_shell> set_multicycle_path 3 -from [get_ports {B C}]
           -through [get_pins u2/Z] -to [get_ports D]
```

The false path through **u1/Z** sets the two false paths shown in the figure, starting at ports A and B and both ending at port D.

The **set_multicycle_path 2** command specifies two paths that start at port A and end at port D: one through **u0** and the other through **u1**. However, the path through **u1** is overridden by the **set_false_path** command, which has a higher priority.

The **set_multicycle_path 3** command specifies two paths: one starting at port B and the other starting at port C, with both ending at port D. However, the path starting at port B is overridden by the **set_false_path** command. There are many ways you could specify this same set of exceptions. For example, you could simplify the last command as follows:

```
pt_shell> set_multicycle_path 3 \
           -from [get_ports C] -to [get_ports D]
```

After you set the exceptions on a design, you can use the **transform_exceptions** command to automatically eliminate invalid, redundant, and overridden exceptions and find a simpler, equivalent set of commands. You can then write out the new set of commands with **write_sdc** or **write_script**. A simpler set of commands is easier to understand and is easier for back-end tools to use. To write out just the exception-setting commands (without

the other timing context commands), use `-include {exceptions}` in the `write_sdc` or `write_script` command.

By default, `transform_exceptions` transforms all exceptions throughout the design. For example, using the command on the design shown in [Figure 7-19](#) gives the following results:

```
pt_shell> transform_exceptions
...
Transformations Summary      false multicycle delay Total
-----
non-existent paths          0      1          0      1
-----
Total transformations        5

Exceptions Summary          false multicycle delay Total
-----
Modified                   0      1          0      1
Analyzed                  1      2          0      3
```

To restrict the scope of the design on which the command operates, you can use the options `-from`, `-through`, `-to`, `-rise_from`, and so on. In that case, PrimeTime only transforms exception commands that operate on at least one of the specified paths.

To generate a transformation report without actually performing the transformation, use the `-dry_run` option. The generated report can help you understand why certain paths are being ignored in the original exception-setting commands.

Exception transformation depends on the design context as well as the set of exception-setting commands that have been used. If you change the design, the transformed exceptions might not be equivalent to the original exceptions. To keep a record of the current set of exception-setting commands, use `write_sdc` or `write_script` before doing the transformation.

Exception Removal

A major effect of exception transformation is the removal of invalid, redundant, and overridden paths from the original exception-setting commands. To find out the reasons that

paths are being eliminated, use the `-verbose` option of the `transform_exceptions` command. For example,

```
pt_shell> transform_exceptions -verbose
...
From      Through      To      Setup      Hold
-----+-----+-----+-----+-----+
*        u1/Z        *      FALSE      FALSE
A        *            D      cycles=2    cycles=0
{ B C }   u2/C        D      cycles=3    cycles=0
NON_EXISTENT_PATH
  -from  B
  -through u2/C
  -to    D
...

```

Each time PrimeTime eliminates paths from the exception-setting commands, it reports one of the following reasons:

- Non-existent path – The path does not exist or has been disabled (for example, by `set_disable_timing`, `set_false_path`, or case analysis).
- Overridden – The exception applied to the path is overridden by another exception of higher priority.
- Invalid startpoint – The path startpoint is not a primary input or a clock pin of a sequential element.
- Invalid endpoint – The path endpoint is not a primary output or a data input pin of a sequential element.
- Unconstrained path – The path is not constrained (for example, a primary input with no input delay specified).
- Clock network path – The path is part of a clock network, which by default is unconstrained.
- Single-cycle MCP – The exception is a multicycle path with the number of cycles set to the default (1 for a setup check or 0 for a hold check).
- Mode analysis – The path has been invalidated by mode analysis (for example, a path that involves writing to RAM with the RAM module set to read mode).
- Exclusive domains – The path crosses between clock domains that have been defined to be exclusive by the `set_clock_groups` command.

By default, `transform_exceptions` removes exceptions for all of the reasons listed earlier. To selectively restrict exception removal to certain reasons, use the `-remove_ignored` option and specify the types of ignored exceptions to remove. For example,

```
pt_shell> transform_exceptions -remove_ignored overridden
```

These are the possible settings for the `-remove_ignored` option:

- `invalid_specifiers` – Paths specified with an invalid startpoint or endpoint, single-cycle multicycle paths.
- `no_path` – Paths that do not exist or are inactive due to case analysis, `set_disable_timing`, or other reasons.
- `overridden` – Paths overridden by other exceptions or mode analysis; and paths between exclusive clock domains.
- `clock_path` – Clock network paths.
- `unconstrained_path` – Paths that are not constrained.

Exception Flattening

The `transform_exceptions` command performs flattening when you use the `-flatten` option. Flattening separates multiple `-from`, `-through`, or `-to` objects in a single command into multiple commands. For example,

```
pt_shell> set_false_path -through {u0/Z u1/Z}
pt_shell> transform_exceptions -flatten
...
pt_shell> write_script -include {exceptions}
```

The script written by the `write_script` command contains the result of flattening the original commands:

```
set_false_path -through [get_pins {u0/Z}]
set_false_path -through [get_pins {u1/Z}]
```

The single `set_false_path` command is flattened into two commands by separating the two `-through` objects.

8

Operating Conditions

Semiconductor device parameters vary with process, voltage, and temperature (PVT) conditions. The ASIC vendor typically characterizes the device parameters in the laboratory under varying conditions, and then specifies the parameter values under different sets of conditions in the logic library. The set of operating conditions used for timing analysis affects the analysis results.

To learn about specifying operating conditions for timing analysis, see

- [Operating Conditions](#)
- [Operating Condition Analysis Modes](#)
- [Minimum and Maximum Delay Calculations](#)
- [Specifying the Analysis Mode](#)
- [Using Two Libraries for Analysis](#)
- [Setting Derating Factors](#)
- [Clock Reconvergence Pessimism Removal](#)
- [Clock On-Chip Variation Pessimism Reduction](#)

Operating Conditions

Integrated circuits exhibit different performance characteristics for different operating conditions: fabrication process variations, power supply voltage, and temperature. The logic library defines nominal values for these parameters and specifies delay information under those conditions.

A set of operating conditions contains the following values:

Operating condition	Description
Process derating factor	This value is related to the scaling of device parameters resulting from variations in the fabrication process. A process number less than the nominal value usually results in smaller delays.
Ambient temperature	The chip temperature affects device delays. The temperature of the chip depends on several factors, including ambient air temperature, power consumption, package type, and cooling method.
Supply voltage	A higher supply voltage usually results in smaller delays.
Interconnect model type	This value defines an RC tree topology that PrimeTime uses to estimate net capacitance and resistance during prelayout analysis.

The delay information for each timing arc is specified at nominal process, temperature, and voltage conditions. If your operating conditions are different from this, PrimeTime applies scaling factors to account for the variations in these conditions. Many libraries use linear scaling for process, temperature, and voltage.

If the logic library contains scaled cell information, you can include the exact delay tables or coefficients for specific operating conditions. This method can be very accurate for library cells that do not scale linearly. For more information, see the Library Compiler and Design Compiler reference manuals.

You can use a single set of operating conditions to do analysis (for setup and hold) or you can specify minimum and maximum conditions. If you do not set operating conditions on your design, PrimeTime uses the default set of operating conditions if the main library contains them, or the nominal values of the main library.

Interconnect Model Types

PrimeTime uses interconnect model information when it calculates net delays for prelayout designs, when annotated net delays and parasitic information are not available. Two nets with the same total resistance and capacitance, but different RC tree topologies, can have different pin-to-pin delays.

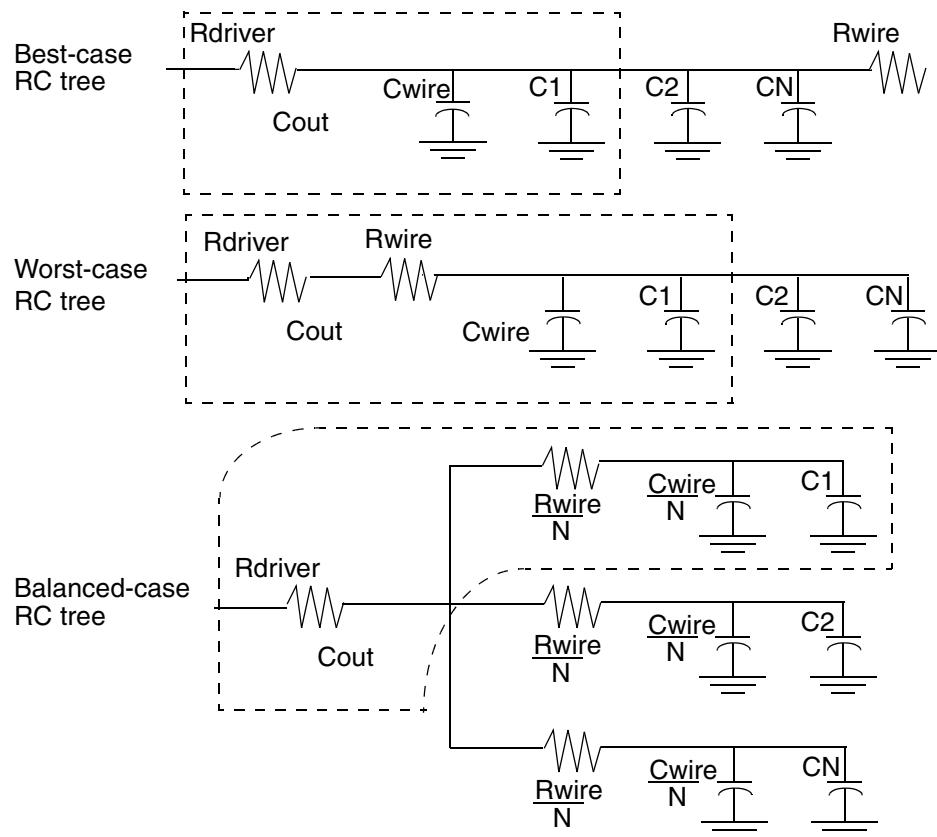
This topic provides background information about interconnect model types. You cannot modify the types in PrimeTime. For more information, see the Library Compiler reference manuals.

The interconnect model is defined by the `tree_type` specification in each logic library's set of operating conditions. A `tree_type` specification indicates the type of wire resistance and capacitance topology: `best_case_tree`, `worst_case_tree`, or `balanced_tree`. For example,

```
operating_conditions(BEST) {
    process      : 1.1;
    temperature : 11.0;
    voltage     : 4.6;
    tree_type   : "best_case_tree";
}
operating_conditions(TYPICAL) {
    process      : 1.3;
    temperature : 31.0;
    voltage     : 4.6;
    tree_type   : "balanced_tree";
}
operating_conditions(WORST) {
    process      : 1.7;
    temperature : 55.0;
    voltage     : 4.2;
    tree_type   : "worst_case_tree";
}
```

If the logic library does not define the tree type, PrimeTime uses the `balanced_tree` model. [Figure 8-1](#) shows the tree type model networks.

Figure 8-1 RC Interconnect Topologies for Fanout of N



Setting Operating Conditions

To specify the process, temperature, and voltage conditions for timing analysis, use the `set_operating_conditions` command.

The operating conditions you specify must be defined in a specified library or a library in the link path. To create custom operating conditions for a library, use the `create_operating_conditions` command. Use the `report_lib` command to get a list of the available operating conditions in a logic library before you use the `set_operating_conditions` command.

To set WCCOM from the `tech_lib` library as a single operating condition, enter

```
pt_shell> set_operating_conditions WCCOM -library tech_lib
```

To set WCCOM as the maximum condition and BCCOM as the minimum condition for on-chip variation analysis, enter

```
pt_shell> set_operating_conditions -analysis_type on_chip_variation \
-min BCCOM -max WCCOM
```

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

Creating Operating Conditions

A logic library contains a fixed set of operating conditions. To create new operating conditions in a library, use the `create_operating_conditions` command. You can use these custom operating conditions to analyze your design during the current session. You cannot write these operating conditions to a library .db file.

To see the operating conditions defined for a library, use the `report_lib` command. To set operating conditions on the current design, use the `set_operating_conditions` command.

To create a new operating condition called WC_CUSTOM in the library `tech_lib`, enter

```
pt_shell> create_operating_conditions -name WC_CUSTOM \
-library tech_lib -process 1.2 \
-temperature 30.0 -voltage 2.8 \
-tree_type worst_case_tree
```

Operating Condition Information

These commands report, remove, or reset operating condition information.

Command	Action
<code>report_design</code>	Lists the operating condition settings for the design
<code>remove_operating_conditions</code>	Removes operating conditions from the current design
<code>reset_design</code>	Resets operating conditions to the default and remove all user-specified data, such as clocks, input and output delays

Operating Condition Analysis Modes

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. In PrimeTime, the `set_operating_conditions` command specifies the operating conditions for analysis, so that PrimeTime can use the appropriate set of parameter values in the logic library.

PrimeTime provides the following methods of setting operating conditions for timing analysis:

- Single operating condition mode – Uses a single set of delay parameters for the entire design, based on one set of process, temperature, and voltage conditions.
- On-chip variation (OCV) mode – Performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For a setup check, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For a hold check, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.
- Advanced on-chip variation (AOCV) mode – Determines derating factors based on metrics of path logic depth and the physical distance traversed by a particular path.

Table 8-1 shows the clock arrival times, delays, operating conditions, and delay derating used for setup and hold checks under each of the operating condition analysis modes.

Table 8-1 Timing Parameters Used for Setup and Hold Checks

Analysis mode	Timing check	Launch clock path	Data path	Capture clock path
Single operating condition	Setup	Late clock, maximum delay in clock path, single operating condition (no derating).	Maximum delay, single operating condition (no derating)	Early clock, minimum delay in clock path, single operating condition (no derating).
	Hold	Early clock, minimum delay in clock path, single operating condition (no derating).	Minimum delay, single operating condition (no derating)	Late clock, maximum delay in clock path, single operating condition (no derating).
OCV mode	Setup	Late clock, maximum delay in clock path, late derating, worst-case operating condition.	Maximum delay, late derating, worst-case operating condition	Early clock, minimum delay in clock path, early derating, best-case operating condition.
	Hold	Early clock, minimum delay in clock path, early derating, best-case operating condition.	Minimum delay, early derating, best-case operating condition	Late clock, maximum delay in clock path, late derating, worst-case operating condition.

Minimum and Maximum Delay Calculations

The `set_operating_conditions` command defines the operating conditions for timing analysis and specifies the analysis type, either single or on-chip variation. The operating conditions must be defined in a specified library or pair of libraries.

By default, PrimeTime performs analysis under one set of operating conditions at a time (single operating condition mode). Using this mode, you need to perform multiple analysis runs to handle multiple operating conditions. Typically, you need to analyze at least two operating conditions to ensure that the design has no timing violations: best case (minimum path report) for hold checks and worst case (maximum path report) for setup checks.

In the on-chip variation (OCV) mode, PrimeTime performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For

setup checks, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For hold checks, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.

In the OCV mode, when a path segment is shared between the clock paths that launch and capture data, the path segment might be treated as having two different delays at the same time. Not accounting for the shared path segment can result in a pessimistic analysis. For a more accurate analysis, this pessimism can be corrected. For more information, see [Clock Reconvergence Pessimism Removal](#).

A minimum-maximum analysis considers the minimum and maximum values specified for the following design parameters:

- Input and output external delays
- Delays annotated from Standard Delay Format (SDF)
- Port wire load models
- Port fanout number
- Net capacitance
- Net resistance
- Net wire load model
- Clock latency
- Clock transition time
- Input port driving cell

For example, to calculate a maximum delay, PrimeTime uses the longest path, worst-case operating conditions, latest-arriving clock edge, maximum cell delays, longest transition times, and so on.

You can perform minimum-maximum analysis using a single library with minimum and maximum operating conditions specified, or two libraries, one for the best-case conditions and one for the worst-case conditions. For more information, see [Using Two Libraries for Analysis](#).

Enable minimum-maximum analysis by using one of these methods:

- Set the minimum and maximum operating conditions with the `set_operating_conditions` command:

```
pt_shell> set_operating_conditions -analysis_type on_chip_variation \
           -min BCCOM -max WCCOM
```

- Read in an SDF file using the option to read both the minimum and maximum delay values from the SDF file:

```
pt_shell> read_sdf -analysis_type on_chip_variation my_design.sdf
```

Minimum-Maximum Cell and Net Delay Values

Each timing arc can have a minimum and a maximum delay to account for variations in operating conditions. You can specify these values in either of the following ways:

- Annotate delays from one or two SDF files
- Have PrimeTime calculate the delay

To annotate delays from one or two SDF files, use one of the following:

- Minimum and maximum from the SDF triplet
- Two SDF files
- Either of the preceding choices, with additional multipliers for maximum and minimum values

To have PrimeTime calculate the delay, use one of the following:

- A single operating condition with timing derating factors to model variation
- Two operating conditions (best-case and worst-case) to model the possible OCV
- Either of the preceding choices with timing derating factors for the minimum and maximum value
- Separate derating factors for cells versus nets
- Separate derating factors for different timing checks (setup, hold, and so forth)

Table 8-2 summarizes the usage of minimum and maximum delays from SDF triplet data.

Table 8-2 Minimum-Maximum Delays From SDF Triplet Data

Analysis mode	Delay based on operating conditions	One SDF file	Two SDF files
Single operating condition	Setup <ul style="list-style-type: none"> • Max data at operating condition • Min capture clock at operating condition Hold <ul style="list-style-type: none"> • Min data at operating condition • Max capture clock at operating condition 	Setup <ul style="list-style-type: none"> - (a:b:c) - (a:b:c) Hold <ul style="list-style-type: none"> - (a:b:c) - (a:b:c) 	
On-chip variation	Setup <ul style="list-style-type: none"> • Max data at worst case • Min capture clock at best case Hold <ul style="list-style-type: none"> • Min data best case • Max capture clock at worst case 	Setup <ul style="list-style-type: none"> - (a:b:c) - (a:b:c) Hold <ul style="list-style-type: none"> - (a:b:c) - (a:b:c) 	Setup <ul style="list-style-type: none"> SDF1 - (a:b:c) SDF2 - (a:b:c) Hold <ul style="list-style-type: none"> SDF1 - (a:b:c) SDF2 - (a:b:c)

PrimeTime uses the triplet value displayed in ***bold italic***.

Setup and Hold Checks

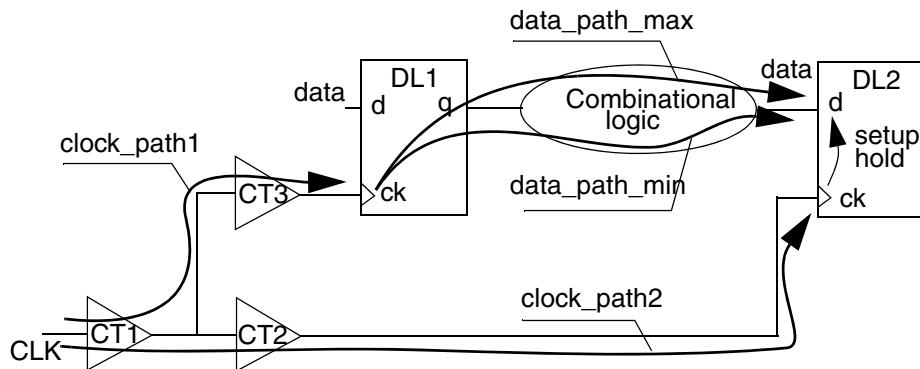
For examples of how setup and hold timing checks are done for a single operating condition and for OCV, see

- [Path Delay Tracing for Setup and Hold Checks](#)
- [Setup Timing Check for Worst-Case Conditions](#)
- [Hold Timing Check for Best-Case Conditions](#)

Path Delay Tracing for Setup and Hold Checks

[Figure 8-2](#) shows how setup and hold checks are done in PrimeTime.

Figure 8-2 Design Example

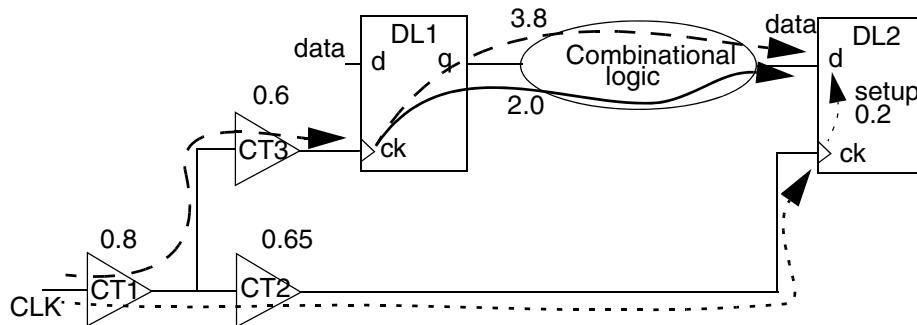


- The setup timing check from pin *DL2/ck* to *DL2/d* considers
 - Maximum delay for *clock_path1*
 - Maximum delay for data path (*data_path_max*)
 - Minimum delay for *clock_path2*
- The hold timing check from pin *DL2/ck* to *DL2/d* considers
 - Minimum delay for *clock_path1*
 - Minimum delay for data path (*data_path_min*)
 - Maximum delay for *clock_path2*

The *data_path_min* and *data_path_max* values can be different due to multiple topological paths in the combinational logic that connects *DL1/q* to *DL2/d*.

Setup Timing Check for Worst-Case Conditions

[Figure 8-3](#) shows how cell delays are computed for worst-case conditions. To simplify the example, the net delays are ignored.

Figure 8-3 Setup Check Using Worst-Case Conditions

PrimeTime checks for a setup violation as follows:

$$\text{clockpath1} + \text{datapathmax} - \text{clockpath2} + \text{setup} \leq \text{clockperiod}$$

where

$$\text{clockpath1} = 0.8 + 0.6 = 1.4$$

$$\text{datapathmax} = 3.8$$

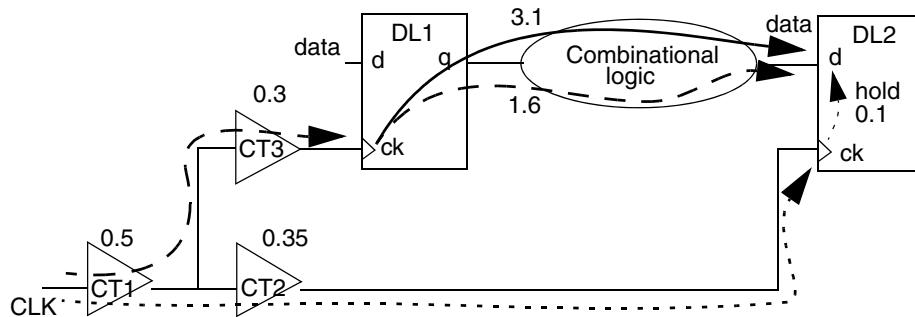
$$\text{clockpath2} = 0.8 + 0.65 = 1.45$$

$$\text{setup} = 0.2$$

The clock period must be at least $1.4 + 3.8 - 1.45 + 0.2 = 3.95$.

Hold Timing Check for Best-Case Conditions

[Figure 8-4](#) shows how cell delays are computed for best-case conditions. Note that the cell delays are different from the delays in [Figure 8-3](#).

Figure 8-4 Hold Check Using Best-Case Conditions

PrimeTime checks for a hold violation as follows:

$$\text{clockpath1} + \text{datapathmax} - \text{clockpath2} - \text{hold} \geq 0$$

where

$$\text{clockpath1} = 0.5 + 0.3 = 0.8$$

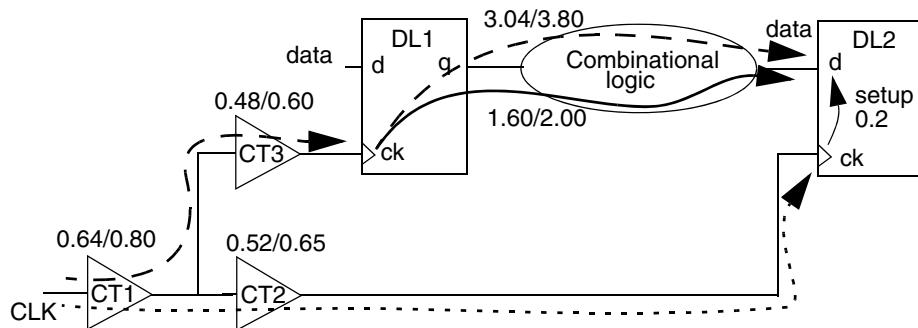
```
datapathmax = 1.6
clockpath2 = 0.5 + 0.35 = 0.85
hold = 0.1
```

No hold violation exists because $0.8 + 1.6 - 0.85 - 0.1 = 1.45$, which is greater than 0.

Path Tracing in the Presence of Delay Variation

In [Figure 8-5](#), each delay of a cell or a net has an uncertainty because of OCV. For example, you can specify that OCV can be between 80 percent and 100 percent of the nominal delays for worst-case conditions. [Figure 8-5](#) shows the resulting delays.

Figure 8-5 OCV for Worst-Case Conditions



In this mode, for a given path, the maximum delay is computed at 100 percent of worst case, and the minimum delay is computed at 80 percent of worst case. PrimeTime checks for a setup violation as follows:

$$\text{clockpath1} + \text{datapathmax} - \text{clockpath2} - \text{setup} \leq 0$$

where

$$\text{clockpath1} = 0.80 + 0.60 = 1.40 \text{ (at 100% of worst case)}$$

$$\text{datapath_max} = 3.80 \text{ (at 100% of worst case)}$$

$$\text{clockpath2} = 0.64 + 0.52 = 1.16 \text{ (at 80% of worst case)}$$

$$\text{setup} = 0.2$$

The clock period must be at least $1.40 + 3.80 - 1.16 + 0.2 = 4.24$

On-chip variation affects the clock latencies; therefore, you only need it when you are using propagated clock latency. If you specify ideal clock latency, you can have PrimeTime consider OCV by increasing the clock uncertainty values with the `set_clock_uncertainty` command.

Specifying the Analysis Mode

For examples that show how to run timing analysis with different operating conditions, see

- [Single Operating Condition Analysis](#)
- [On-Chip Variation Analysis](#)

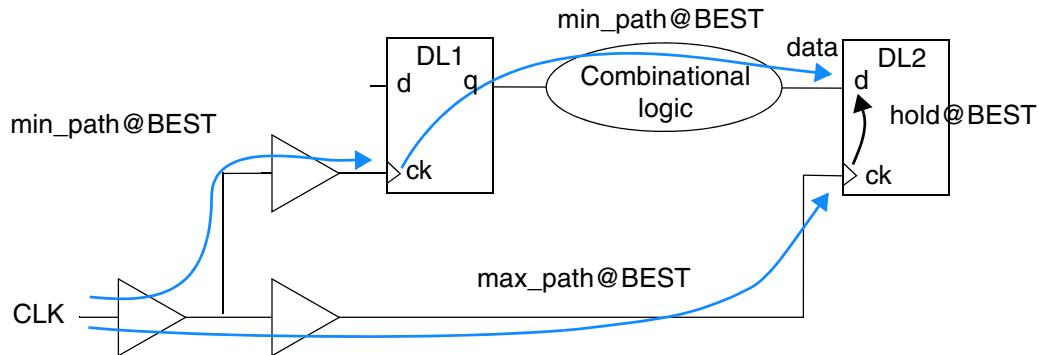
Single Operating Condition Analysis

The following example runs timing analysis with the best-case operating conditions:

```
pt_shell> set_operating_conditions BEST
pt_shell> report_timing -delay_type min
```

[Figure 8-6](#) shows the reported path.

Figure 8-6 Timing Path for One Operating Condition – Best Case

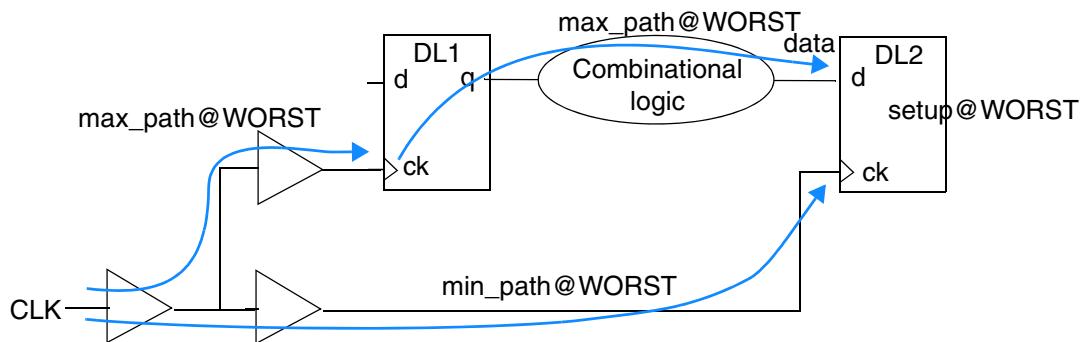


The following example runs timing analysis with the worst-case operating conditions:

```
pt_shell> set_operating_conditions WORST
pt_shell> report_timing -delay_type max
```

[Figure 8-7](#) shows the reported path.

Figure 8-7 Timing Path for One Operating Condition – Worst Case



On-Chip Variation Analysis

To perform OCV analysis,

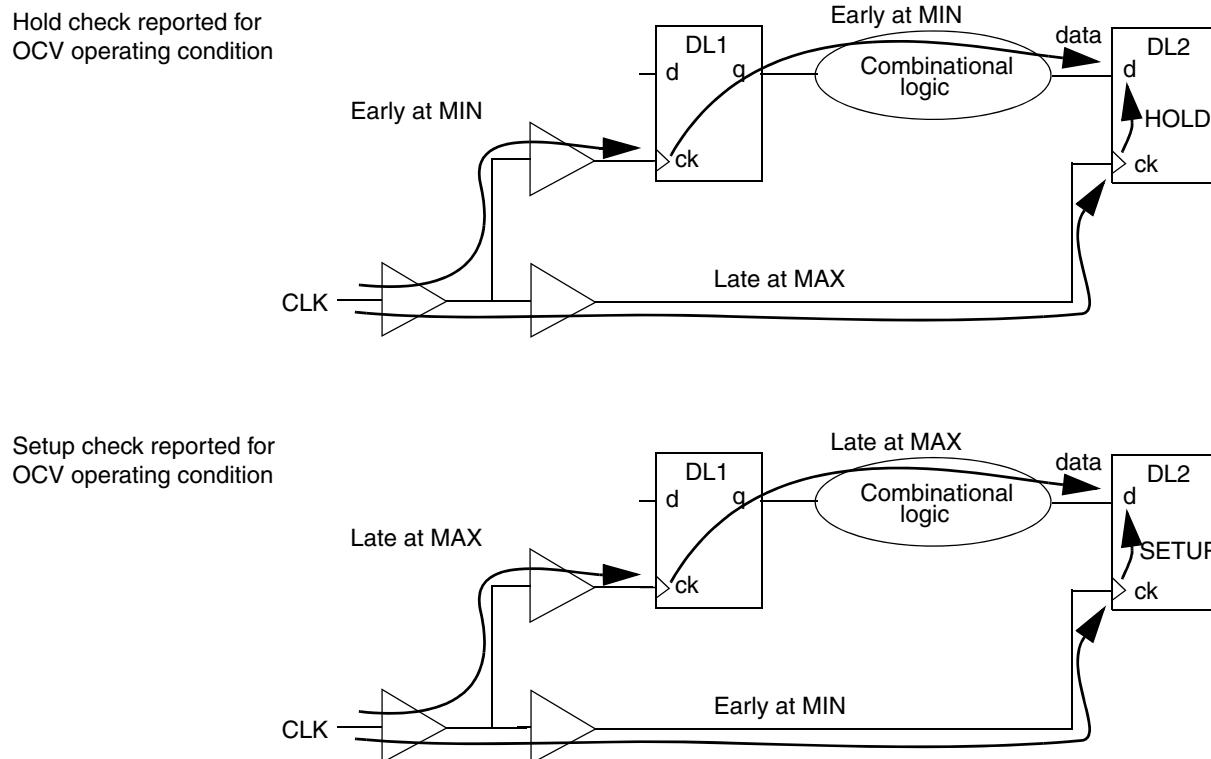
1. Specify the operating conditions:

```
pt_shell> set_operating_conditions \
           -analysis_type on_chip_variation \
           -min MIN -max MAX
```

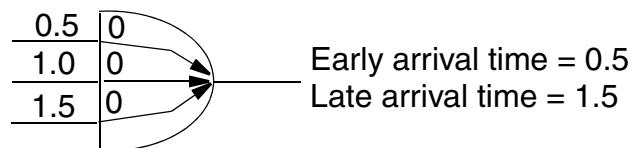
2. Report the timing for setup and hold analysis:

```
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

[Figure 8-8](#) shows the paths reported when you run OCV analysis.

Figure 8-8 Timing Path Reported During OCV Analysis

As shown in [Figure 8-9](#), the early arrival time at the AND gate is 0.5 ns and the late arrival time is 1.5 ns, assuming the gate delay is zero.

Figure 8-9 Early and Late Arrival Time

Example 1

This command sequence performs timing analysis for OCV 20 percent below the worst-case commercial (WCCOM) operating condition.

```
pt_shell> set_operating_conditions -analysis_type \
           on_chip_variation WCCOM

pt_shell> set_timing_derate -early 0.8

pt_shell> report_timing
```

Example 2

This command sequence performs timing analysis for OCV between two predefined operating conditions: WCCOM_scaled and WCCOM.

```
pt_shell> set_operating_conditions -analysis_type \
           on_chip_variation -min WCCOM_scaled -max WCCOM

pt_shell> report_timing
```

Example 3

This command sequence performs timing analysis with OCV. For cell delays, the OCV is between 5 percent above and 10 percent below the SDF back-annotated values. For net delays, the OCV is between 2 percent above and 4 percent below the SDF back-annotated values. For cell timing checks, the OCV is 10 percent above the SDF values for setup checks and 20 percent below the SDF values for hold checks.

```
pt_shell> set_timing_derate -cell_delay -early 0.90
pt_shell> set_timing_derate -cell_delay -late 1.05
pt_shell> set_timing_derate -net_delay -early 0.96
pt_shell> set_timing_derate -net_delay -late 1.02
pt_shell> set_timing_derate -cell_check -early 0.80
pt_shell> set_timing_derate -cell_check -late 1.10
```

Using Two Libraries for Analysis

The `set_min_library` command directs PrimeTime to use two logic libraries simultaneously for minimum-delay and maximum-delay analysis. For example, you can choose two libraries that have the following characteristics:

- Best-case and worst-case operating conditions
- Optimistic and pessimistic wire load models
- Minimum and maximum timing delays

To perform an analysis of this type, use the `set_min_library` command to create a minimum/maximum association between two libraries. Specify one library to be used for maximum delay analysis and another to be used for minimum delay analysis. Only the maximum library should be present in the link path.

When you use the `set_min_library` command, PrimeTime first checks the library cell in the maximum library, and then looks in the minimum library to see if a match exists. If a library cell with the same name, the same pins, and the same timing arcs exists in the minimum library, PrimeTime uses that timing information for minimum analysis. Otherwise, it uses the information in the maximum library.

Setting Derating Factors

You can have PrimeTime adjust minimum delays and maximum delays by specified factors to model the effects of operating conditions. This adjustment of calculated delays is called derating. Derating affects the delay and slack values reported by the `report_timing` command and other reporting commands.

The `set_timing_derate` command specifies the adjustment factors and the scope of the design to be affected by derating. For example,

```
pt_shell> set_timing_derate -early -cell_delay 0.9  
pt_shell> set_timing_derate -late -cell_delay 1.2
```

The first of these two commands causes all early (shortest-path) delays to be decreased by 10%, such as the capture clock path in a setup check. The second command causes all late (longest-path) delays to be increased by 20%, such as the launch clock path and the data path in a setup check. These changes result in a more conservative analysis than leaving delays at their original calculated values.

The `-early` or `-late` option specifies whether the shortest-path or longest-path delays are affected by the derating factor. Exactly one of these two options must be used in the command. To set both early and late derating, use two `set_timing_derate` commands. Early path delays include the capture clock path for a setup check, and the launch clock path and data path for a hold check. Late path delays include the launch clock path and data path for a setup check, and the capture clock path for a hold check.

The fixed-point value in the command specifies the derating factor applied to the delays. Use a value of less than 1.0 to reduce the delay of any given path or cell check. Similarly, use a derating factor greater than 1.0 to increase the delay of any given path or cell check. Typically, derating factors less than 1.0 are used for early (shortest-path) delays and greater than 1.0 for late (longest-path) delays. This approach results in a more conservative analysis.

The last derating value to be set overrides any previously set values. To reset the derating factor globally to 1.0, use the `reset_timing_derate` command.

Delays are adjusted according to the following formula:

$$\text{delay_new} = \text{old_delay} + ((\text{derating_factor} - 1.0) * \text{abs}(\text{old_delay}))$$

When the delay is a positive value (the usual case), this equation is reduced to:

$$\text{delay_new} = \text{old_delay} * \text{derating_factor}$$

For negative delay, the delay adjustment equation is reduced to:

$$\text{delay_new} = \text{old_delay} * (2.0 - \text{derating_factor})$$

Delays can be negative in some unusual situations. For example, if the input transition is slow and the output transition is fast for a cell, the time at which the input signal reaches the 50% trip point can be later than the time at which the output signal reaches the 50% trip point. The resulting delay from input to output is negative. If you are using PrimeTime SI, a similar situation can occur for a net due to a change in delay caused by crosstalk.

The `-rise` or `-fall` option specifies whether rise delays or fall delays are affected by derating. If neither option is used, both types of delays are affected. The `-clock` or `-data` option specifies whether clock paths or data paths are affected by derating. If neither option is used, both types of paths are affected. A clock path is a path from a clock source to the clock input pin of a launch or capture register. A data path is a path starting from an input port or from the data output of a launch register, and ending at an output port or at the data input of a capture register.

The `-net_delay`, `-cell_delay`, and `-cell_check` options let you specify the functional scope of the design to be affected by derating. The `-net_delay` option derates net delays (the delay from a net driver to a net load). The `-cell_delay` option derates cell delays (the delay from a cell input to a cell output). The `-cell_check` option derates cell timing checks (cell hold and removal times for early derating, or cell setup and recovery times for late derating). You can use any combination of these three options. If you do not use any of these options, the derating factor applies to net delays and cell delays, but not cell timing checks.

If you want only some cells or nets to be affected by derating, you can list them in the command. The list can include library cells, hierarchical cells, leaf-level cells, and nets. In the absence of a list, the whole design is affected.

To set a derating factor on all nets within a hierarchy, use the `-net_delay` option. For example,

```
pt_shell> set_timing_derate -net_delay -early 0.8 \
           [get_cells hier_cell]
```

If you do not specify the `-net_delay` option, only the cells have the derating factor set on them. This feature allows you to specify different derating factors for delta net delays and

non-delta net delays. To set a derating factor for non-delta net delays only, use the `-static` option:

```
set_timing_derate -net_delay -static -early/late -data/clock $net $value1
```

To set a derating factor for delta net delays only, use the `-dynamic` option:

```
set_timing_derate -net_delay -dynamic -early/late  
-data/clock $net $value2
```

If both `-static` and `-dynamic` options are omitted, the derating factor is applied to both delta and non-delta net delays.

Different sets of derating factors can be specified for a deterministic PrimeTime analysis and a variational PrimeTime analysis. To set a derating factor for deterministic delays, use the `-scalar` option. To set a derating factor for delays that have been computed using variational information, use the `-variation` option.

If you set a derating factor on a net that crosses a hierarchical boundary, the derating affects the whole net, not just the part of the net listed in the command. Also, if you set a derating factor on a hierarchical cell, the derating factor extends to cells and nets within and below the hierarchy. PrimeTime issues a warning message when it extends derating across a hierarchical boundary.

In case of conflict between different `set_timing_derate` values specified for different levels of the design, the more detailed command (with the narrowest scope) has precedence. For example, the following commands have decreasing order of precedence:

```
pt_shell> set_timing_derate -late -cell_delay 1.4 [get_cells inv*]  
          # derates a collection of cells  
pt_shell> set_timing_derate -late -cell_delay 1.3 \  
          [get_lib_cells class/ND*]  
          # derates cells in a collection of cells in a library class  
pt_shell> set_timing_derate -late -cell_delay 1.2  
          # derates all cells  
pt_shell> set_timing_derate -late -1.1  
          # derates all nets and cells
```

Derating is applied to cells in the following order of precedence, from highest to lowest priority:

1. Leaf-level cell
2. Library cell
3. Hierarchical cell (containing leaf-level cells)
4. Design

This derate precedence behavior is consistent with the derate precedence rules used in Design Compiler and IC Compiler. It is also consistent with other precedence-based cell attributes in PrimeTime, such as the `dont_touch` attribute.

PrimeTime stores and checks global timing derating factors against block scope. These stored values are checked to make sure that the top-level ranges are completely within the block-level range. For cases where a single value is used at the block level (that is, when the early derate is the same as the late derate), the top-level values must be the same as well. If derating factors are not specified at either the block or top levels, the derating factor used is 1.0 and the scope check is performed against this value.

To enable or disable timing constraint deratings for minimum pulse width and minimum period constraints, use the `timing_use_constraint_derates_for_pulse_checks` variable. By default, this variable is set to `false`. When you set this variable to `true`, PrimeTime uses factors defined by the `set_timing_derate` command to derate the minimum pulse width and minimum period constraints in the following way:

- Use the `-late -cell_check -rise` options to apply the minimum pulse width constraints for high pulses
- Use the `-late -cell_check -fall` options to apply the minimum pulse width constraints for low pulses

PrimeTime applies the greater of these values to the minimum period constraints.

If you use the `set_timing_derate` command while the analysis is set to single operating condition mode, PrimeTime automatically changes to the OCV mode, like using this command:

```
pt_shell> set_operating_conditions -analysis_type on_chip_variation
```

If you use the `-derate` option with the `report_timing` command, the global, net-specific, or instance-specific derating factor is applied to each cell or net is reported in a column next to the incremental delay for each item.

To report the current timing derating factors that have been set, use the `report_timing_derate` command. For example,

```
pt_shell> report_timing_derate
          # reports all derating settings
pt_shell> report_timing_derate [get_cells U*]
          # reports derating settings on all cells named U*
```

Use the `-include_inherited` option to include reporting of derating values inherited from other objects, such as a lower-level cell that inherits its derating values from a higher-level cell. Otherwise, the command reports only the derating values set directly on the specified object.

Clock Reconvergence Pessimism Removal

Clock reconvergence pessimism is an accuracy limitation that occurs when two different clock paths partially share a common physical path segment and the shared segment is assumed to have a minimum delay for one path and a maximum delay for the other path. This condition can occur any time that launch and capture clock paths use different delays, most commonly with OCV analysis (see [Table 8-3](#)). Automated correction of this inaccuracy is called clock reconvergence pessimism removal (CRPR).

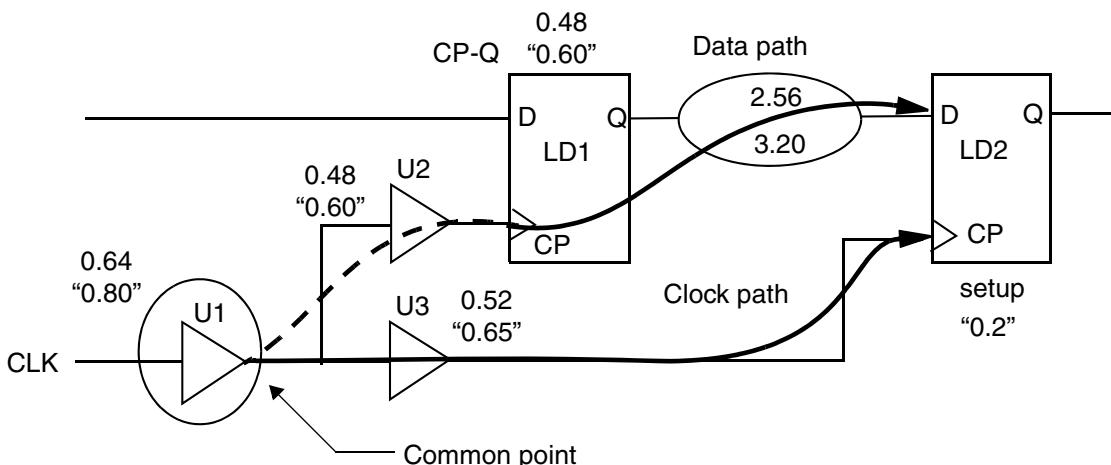
By default, PrimeTime performs CRPR at the same time as regular timing analysis. You need to enable CRPR before you do timing analysis. For information about enabling CRPR, see [Using CRPR Commands](#). The following examples demonstrate how the pessimism removal works.

On-Chip Variation Example

Consider the following command sequence for running OCV analysis and the corresponding design in [Figure 8-10](#):

```
pt_shell> set_operating_conditions -analysis_type \
           on_chip_variation -min MIN -max MAX
pt_shell> set_timing_derate -net_delay -early 0.80
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

Figure 8-10 Clock Reconvergence Pessimism Example



[Figure 8-10](#) shows how the analysis is done. Each delay (considered equal for rising and falling transitions to simplify this example) has a minimum value and a maximum value computed for the minimum and maximum operating conditions.

The setup check at LD2/CP considers the clock path to the source latch (CLK to LD1/CP) at 100 percent worst case, and the clock path to the destination latch (CLK to LD2/CP) at 80 percent worst case.

Although this is a valid approach, the test is pessimistic because clock path1 (CLK to LD1/CP) and clock path2 (CLK to LD2/CP) share the clock tree until the output of U1. The shared segment is called the *common portion*, consisting of just cell U1 in this example. The last cell output in the shared clock segment is called the *common point*, which is the output of U1 in this case.

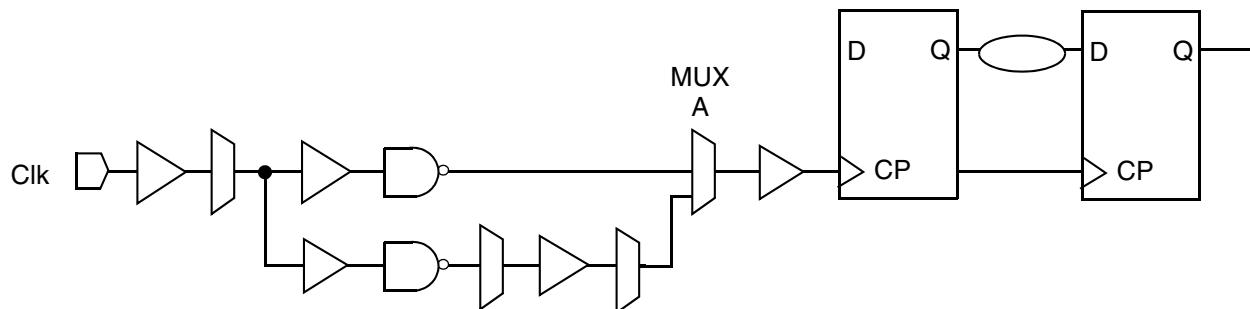
The setup check considers that cell U1 simultaneously has two different delays, 0.64 and 0.80, resulting in a pessimistic analysis in the amount of 0.16. This amount, obtained by subtracting the earliest arrival time from the latest arrival time at the common point, is called the *clock reconvergence pessimism*.

This inaccuracy also occurs in an analogous way for the hold test at the LD2 latch.

Reconvergent Logic Example

[Figure 8-11](#) shows a situation where clock reconvergence can occur, even in the absence of OCV analysis. In this example, there is reconvergent logic in the clock network. The two clock paths that feed into the multiplexer cannot be active at the same time, but an analysis could consider both the shorter and longer paths for one setup or hold check.

Figure 8-11 Reconvergent Logic in a Clock Network



Minimum Pulse Width Checking Example

The `report_constraint` command checks for minimum pulse width violations in clock networks (as specified by the `set_min_pulse_width` command) and at cell inputs (as specified in the logic library). CRPR, when enabled, can increase the accuracy of minimum pulse width checking. For information about enabling CRPR, see [Using CRPR Commands](#).

For example, consider the circuit shown in [Figure 8-12](#). The external clock source has early and late source latency set on it. In addition, the two buffers in the path have minimum and maximum rise and fall delay values defined.

The `report_constraint` command checks the pulse width of the clock signal in the clock network and upon reaching the flip-flop. For level-high pulse width checking, PrimeTime considers maximum delay for the rising edge and minimum delay for the falling edge of the clock (and conversely for level-low pulse width checking).

For the example shown in [Figure 8-12](#), in the absence of CRPR, the worst-case pulse width is very small and violates the pulse width constraint of the flip-flop. However, this analysis is pessimistic because it assumes simultaneous worst-case delays for rising and falling edges. In a real circuit, rising-edge and falling-edge delays are at least somewhat correlated. For example, for the delay from the external clock source to the CLK input port, if the rising-edge delay is at the minimum, -1.3 , the falling-edge delay is probably equal or close to -1.3 , and not at the maximum of $+1.4$.

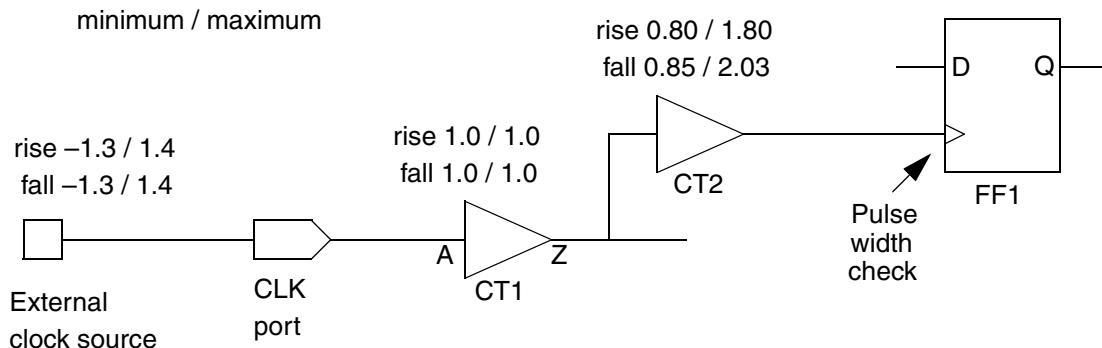
To account for correlation between rising-edge and falling-edge delays, enable CRPR. In that case, PrimeTime adds a certain amount of slack back into the minimum pulse width calculation. The amount added is equal to the range of minimum rise delay or the range maximum fall delay for the path, whatever is smaller:

$$\text{crp} = \min[(\text{Mr} - \text{mr}), (\text{Mf} - \text{mf})]$$

where

`crp` = clock reconvergence pessimism
`Mr` = cumulative maximum rise delay
`mr` = cumulative minimum rise delay
`Mf` = cumulative maximum fall delay
`mf` = cumulative minimum fall delay

For an example of this calculation applied to pulse width checking, see [Figure 8-12](#).

Figure 8-12 Minimum Pulse Width Analysis

$$\text{Minimum rise} = -1.3 + 1.0 + 0.80 = 0.50$$

$$\text{Maximum rise} = 1.4 + 1.0 + 1.80 = 4.20$$

$$\text{Rise range} = 4.20 - 0.50 = 3.70$$

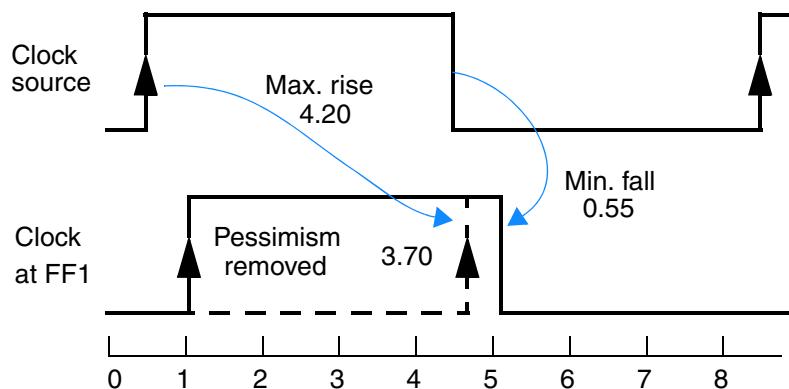
$$\text{Minimum fall} = -1.3 + 1.0 + 0.85 = 0.55$$

$$\text{Maximum fall} = 1.4 + 1.0 + 2.03 = 4.43$$

$$\text{Fall range} = 4.43 - 0.55 = 3.88$$

Clock reconvergence pessimism = smaller of rise range or fall range = 3.70

Minimum pulse width check: maximum rise = 4.20, minimum fall = 0.55



Using CRPR Commands

By default, CRPR is enabled. To disable CRPR, set the `timing_remove_clock_reconvergence_pessimism` variable to `false` before you begin timing analysis. Using CRPR results in a more accurate (less pessimistic) analysis, but increases the runtime and memory usage. Any change in this variable setting causes a complete timing update.

When CRPR is enabled, PrimeTime uses the correction algorithm and reports the results in all the report_timing, report_constraint, report_analysis_coverage, and report_bottleneck major types of reports. The following example shows the timing results with CRPR enabled for the design in [Figure 8-10](#):

```
pt_shell> report_timing -delay_type max
*****
Report : timing
    -path full
    -delay max
    -max_paths 1
...
*****
Startpoint: LD1 (rising edge-triggered flip-flop clocked by CLK)
Endpoint: LD2 (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point           Incr      Path
-----
clock CLK (rise edge)      0.00      0.00
clock network delay (propagated) 1.40      1.40
LD1/CP (FD2)            0.00      1.40 r
LD1/Q (FD2)             0.60      2.00 f
U1/z (AN2)              3.20      5.20 f
data arrival time        5.20
                           5.20

clock CLK (rise edge)      6.00      6.00
clock network delay (propagated) 1.16      7.16
clock reconvergence pessimism 0.16      7.32
clock uncertainty          0.00      7.32
LD2/CP (FD2)              7.32 r
library setup time         -0.20      7.12
data required time         7.12
                           7.12

data required time         7.12
data arrival time          -5.20
                           -5.20

-----
```

Point	Incr	Path
clock CLK (rise edge)	0.00	0.00
clock network delay (propagated)	1.40	1.40
LD1/CP (FD2)	0.00	1.40 r
LD1/Q (FD2)	0.60	2.00 f
U1/z (AN2)	3.20	5.20 f
data arrival time		5.20
clock CLK (rise edge)	6.00	6.00
clock network delay (propagated)	1.16	7.16
clock reconvergence pessimism	0.16	7.32
clock uncertainty	0.00	7.32
LD2/CP (FD2)		7.32 r
library setup time	-0.20	7.12
data required time		7.12
data required time		7.12
data arrival time		-5.20
slack (MET)		1.92

To specify whether to perform CRPR on clock paths that have opposite-sense transitions in a shared path segment, set the timing_clock_reconvergence_pessimism variable. You can set this variable to normal (the default) or same_transition. If this variable is set to normal, PrimeTime still performs CRPR with opposite-sense transitions in the shared path segment. In that case, if the two transition types produce different correction values, the smaller of the two values is used. If you set the timing_clock_reconvergence_pessimism variable to same_transition, the CRPR value is removed at the last common point where the launch and capture edge have the same sense. The common point is identified by tracing backward from the last topological common point on a unique common clock path. In

case PrimeTime reaches a common point at which multiple clock paths converge, that point is returned as the common point.

Table 8-3 summarizes the application of either rising or falling clock reconvergence pessimism based on the transition types at the common point in the clock path and `timing_clock_reconvergence_pessimism` variable setting.

Table 8-3 Application of Rise/Fall CRP Value Based on Variable Setting

Transition types at common point in clock paths	<code>timing_clock_reconvergence_pessimism = normal</code>	<code>timing_clock_reconvergence_pessimism = same_transition</code>
Both rising	<code>crp_rise</code> pessimism removed	<code>crp_rise</code> pessimism removed
Both falling	<code>crp_fall</code> pessimism removed	<code>crp_fall</code> pessimism removed
One rising, one falling	smaller of <code>crp_rise</code> or <code>crp_fall</code> removed	Pessimism at last common point with same sense removed

For the minimum pulse width checking example shown in [Figure 8-12](#), you could set up the analysis with a script similar to this:

```
set_operating_conditions -analysis_type on_chip_variation
create_clock -period 8 CLK
set_propagated_clock [all_clocks]
set_clock_latency -source -early -1.3 CLK
set_clock_latency -source -late 1.4 CLK
set_annotated_delay -cell -from CT1/A -to CT1/Z 1
set_annotated_delay -cell -from CT2/A -to CT2/Z -min -rise 0.8
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -rise 1.8
set_annotated_delay -cell -from CT2/A -to CT2/Z -min -fall 0.85
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -fall 2.03
```

With CRPR enabled, a report_constraint path report looks like this:

Point	Incr	Path
clock CLK (rise edge)	0.00	0.00
clock network delay (propagated)	4.20	4.20 r
FF1/CP	0.00	4.20 r
open edge arrival time		4.20
clock CLK (fall edge)	4.00	4.00
clock network delay (propagated)	0.55	4.55 f
FF1/CP	0.00	4.55 f
clock reconvergence pessimism	3.70	8.25
close edge arrival time		8.25
required pulse width (high)		3.50
actual pulse width		4.05
slack		0.55

You can set a threshold that allows PrimeTime to ignore small amounts of clock reconvergence pessimism. For computational efficiency, PrimeTime merges multiple points for CRPR calculations when the CRP differences between adjacent points are smaller than the specified threshold. This merging of nodes can reduce CPU and memory usage, without a significant loss of accuracy when an appropriate threshold is set.

The `timing_crpr_threshold_ps` variable specifies the threshold. The units are picoseconds, irrespective of the time units of the logic library. By default, the variable is set to 20, which allows adjacent nodes to be merged when the difference in clock reconvergence pessimism is 20 ps or less.

For a good balance between performance and accuracy, try setting this variable to one-half the stage delay of a typical gate in the clock network. (The stage delay is gate delay plus net delay.) You might want to use a larger value during the design phase for faster analysis, and then a smaller value for sign-off accuracy.

CRPR and Crosstalk Analysis

When you perform crosstalk analysis using PrimeTime SI, a change in delay due to crosstalk along the common segment of a clock path can be pessimistic, but only for a zero-cycle check. A zero-cycle check occurs when the same clock edge drives both the launch and capture events for the path. For other types of paths, a change in delay due to crosstalk is not pessimistic because the change cannot be assumed to be identical for the launch and capture clock edges.

Accordingly, the CRPR algorithm removes crosstalk-induced delays in a common portion of the launch and capture clock paths only if the check is a zero-cycle check. In a zero-cycle

check, aggressor switching affects both the launch and capture signals in the same way at the same time.

Here are some cases where the CRPR might apply to crosstalk-induced delays:

- Standard hold check
- Hold check on a register with the Q-bar output connected to the D input, as in a divide-by-2 clock circuit
- Hold check with crosstalk feedback due to parasitic capacitance between the Q-bar output and D input of a register
- Hold check on a multicycle path set to zero, such as circuit that uses a single clock edge for launch and capture, with designed-in skew between launch and capture
- Certain setup checks where transparent latches are involved

CRPR With Dynamic Clock Arrivals

A similar scenario to CRPR with crosstalk can occur if dynamic annotations have been set in the clock network. Dynamic annotations include dynamic clock latency and dynamic rail voltage, set by `set_clock_latency` and `set_voltage` commands respectively. These dynamic annotations can lead to dynamic clock arrivals. CRPR handles these dynamic clock arrivals in the same way as it handles delays due to crosstalk. The CRPR value is calculated using dynamic clock arrivals for zero cycle paths only. For all other paths, only the static component of the clock arrival is used thus producing more accurate results.

An example of such dynamic annotations is as follows:

```
pt_shell> set_clock_latency -early -source 2.5 \
           -dynamic -0.5 [get_clocks CLK]

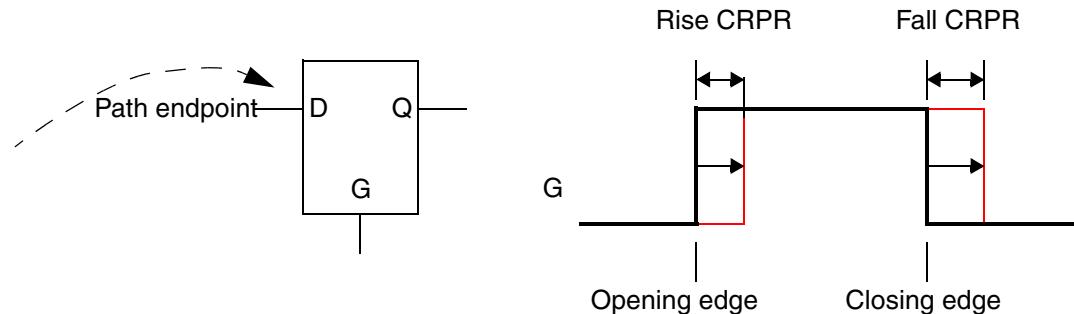
pt_shell> set_clock_latency -source -late 5.5 \
           -dynamic 0.5 [get_clocks CLK]
```

The first of these two commands specifies a total early source latency of 2.5, consisting of static source latency of 3.0 and dynamic source latency of -0.5. The second command specifies a total late source latency of 5.5, consisting of static source latency of 5.0 and dynamic source latency of 0.5. In this case, the static CRP is equal to 2 (5 minus 3). The dynamic CRP is equal to 3 (5.5 minus 2.5).

Transparent Latch Edge Considerations

For a path that ends at a transparent latch, PrimeTime calculates two clock reconvergence pessimism values: one for rising edges and one for falling edges at the common node. The opening and closing clock edges are effectively shifted, each by an amount equal to its corresponding pessimism value, as indicated in [Figure 8-13](#).

Figure 8-13 CRPR for Latches



In practice, the opening-edge pessimism value affects the slack of nonborrowing paths and also reduces the amount of time borrowed for borrowing paths. Meanwhile, the closing-edge pessimism value increases the maximum amount of time borrowing allowed at a latch and reduces the amount of the violation for a path that fails to meet its setup constraint.

To get a report about the calculation of clock reconvergence pessimism values for level-sensitive latches, use the `report_crpr` command.

Reporting CRPR Calculations

The `report_crpr` command reports the calculation of clock reconvergence pessimism (CRP) between two register clock pins or ports. It reports the time values calculated for both static and dynamic conditions, and the choice from among those values actually used for pessimism removal. In the command, you specify the pins of the launch and capture registers, the clock, and type of check (setup or hold). For example,

```
pt_shell> report_crpr -from [get_pins ffa/CP] \
           -to [get_pins ffd/CP] \
           -from_clock CLK -setup
```

The command reports the location of the common node, the launch and capture edge types (rising or falling), the four calculated arrival times at the common point (early/late, rise/fall), the calculated CRP values (rise and fall), and the values actually used for opening-edge and closing-edge pessimism removal.

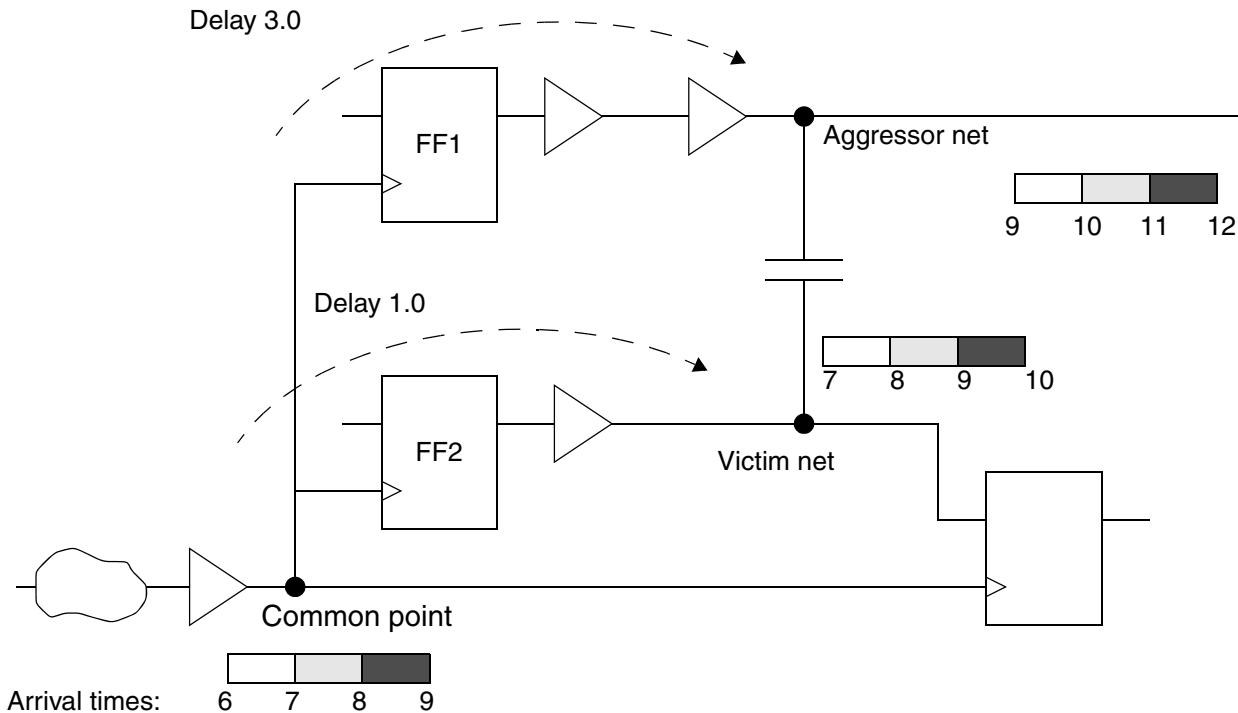
The amount of CRP reported by the `report_crpr` command can be slightly different from the amount reported by the `report_timing` command. This is because the `report_timing` command, for computational efficiency, “merges” multiple points for CRPR calculations when the CRP differences between adjacent points are too small to be significant. The `timing_crpr_threshold_ps` variable sets the time threshold for merging, which is 20 picoseconds by default. When the `report_crpr` and `report_timing` commands give different CRP values, the value reported by `report_crpr` command is more accurate because it does not merge adjacent points.

Clock On-Chip Variation Pessimism Reduction

The `set_timing_derate` command can be used to model the effects of on-chip variation (OCV). The command specifies a factor by which the delays of long path delays are increased or the delays of short paths are decreased. When there is a common segment between the launch and capture clock paths, the clock reconvergence pessimism removal (CRPR) algorithm, if enabled, removes the pessimism caused by the derating of delay times along the common segment. However, by default, pessimism removal occurs only after the final slack has been calculated for the timing path. No pessimism removal occurs during the calculation of crosstalk arrival windows. If the clock paths leading up to the aggressor net and victim net share a common segment, then the calculated arrival windows are wider than necessary, possibly causing the aggressor and victim arrival windows to marginally overlap. This can cause a crosstalk situation to occur that would not occur with accurate CRPR accounting during arrival window overlap analysis.

[Figure 8-14](#) is a simple example that demonstrates the pessimism caused by derating a long clock path. There is some cross-coupling capacitance between two wires in the fanout of FF1 and FF2, both of which are clocked by the same clock signal.

Figure 8-14 Clock Reconvergence Pessimism in Crosstalk Arrival Windows



In the absence of derating, the arrival windows are 1.0 time units wide. Because of the differences in delay along the paths leading up to the aggressor and victim nets, the windows do not overlap and no crosstalk delay effects are possible. For example, if the aggressor transition occurs between 9.0 and 10.0 time units, the victim transition has already occurred, at some time between 7.0 and 8.0 time units. However, with derating applied, the long clock path leading up to the common point has an early arrival at time 6.0 and a late arrival at time 9.0. This widens the aggressor and victim windows to 3.0 time units, causing them to overlap and produce a crosstalk situation where none could actually exist in the real circuit.

To get the best possible accuracy during path-based analysis, you can optionally have CRPR applied during arrival window overlap analysis, thus removing the pessimism caused by different early and late arrival times at the common point leading up to the aggressor and victim. To invoke this option, set the

`pba_enable_xtalk_delay_ocv_pessimism_reduction` variable to `true`. The default setting is `false`. When the variable is set to `true`, and if CRPR is enabled, during path-based analysis, PrimeTime SI applies CRPR during calculation of aggressor and victim arrival windows, resulting in more accurate arrival windows, at the cost of some additional runtime. Path-based analysis occurs when you use the `-pba_mode` option with the `get_timing_paths` or `report_timing` command. CRPR is enabled when the `timing_remove_clock_reconvergence_pessimism` variable is set to `true`.

9

Delay Calculation

To perform delay calculation accurately and efficiently, PrimeTime uses models to represent the driver, RC network, and capacitive loads on the net. An ideal model produces the same delays and slews as a SPICE simulation at the output of the driver and at the input of each receiver.

To learn about the different models and analysis modes for delay calculation, see

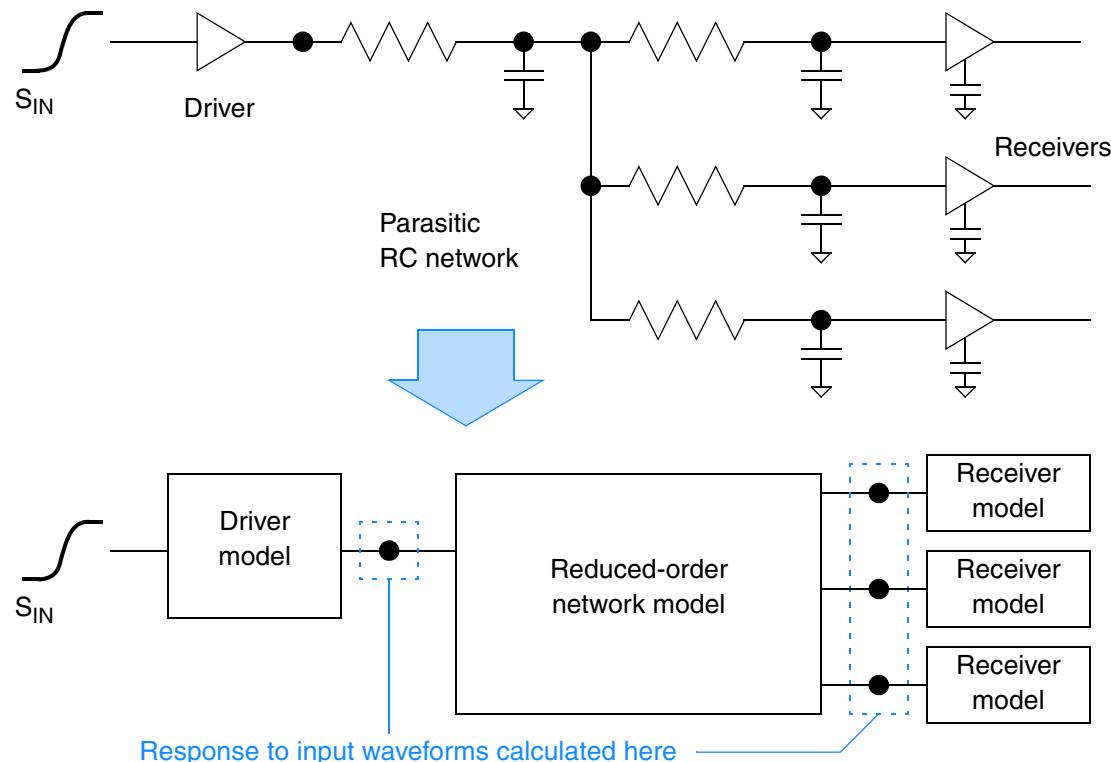
- [Overview of Delay Calculation](#)
- [Nonlinear Delay Models](#)
- [Composite Current Source Timing Models](#)
- [CCS Cross-Library Scaling and Exact Matching](#)
- [Fast Multidrive Delay Analysis](#)
- [Parallel Driver Reduction](#)
- [Path-Based Timing Analysis](#)
- [Multi-Input Switching Analysis](#)

Overview of Delay Calculation

To perform static timing analysis, PrimeTime must accurately calculate the delay and slew (transition time) at each stage of each timing path. A stage consists of a driving cell, the annotated RC network at the output of the cell, and the capacitive load of the network load pins. The goal is to compute the response at the driver output and at the network load pins, given the input slew or waveform at the driver input, using the least amount of runtime necessary to get accurate results.

To perform stage delay calculation accurately and efficiently, PrimeTime uses models to represent the driver, RC network, and capacitive loads on the net. See [Figure 9-1](#). An ideal model produces exactly the same delays and slews as a SPICE simulation at the output of the driver and at the input of each receiver.

Figure 9-1 Models Used to Calculate Stage Delays and Slew



The driver model is intended to reproduce the response of the driving cell's underlying transistor circuitry when connected to an arbitrary RC network, given a specific input slew.

The reduced-order network model is a simplified representation of the full annotated network that has nearly the same response characteristics as the original network. PrimeTime uses the Arnoldi reduction method to create this model.

The receiver model is intended to represent the complex input capacitance characteristics of a cell input pin, including the effects of the rise and fall transition, the slew at the pin, the receiver output load, the state of the cell, and the voltage and temperature conditions.

To specify the types of driver and receiver models for RC delay calculation, set the `rc_driver_model_mode` and `rc_receiver_model_mode` variables to one of the following values:

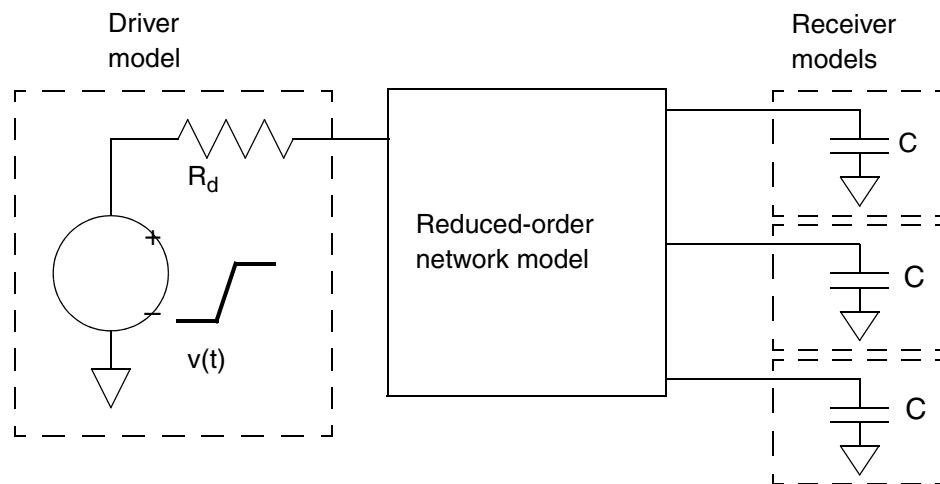
- `basic` – RC delay calculation uses the basic nonlinear delay model (NLDM) present in the cell libraries.
- `advanced (default)` – RC delay calculation uses the more advanced Composite Current Source (CCS) timing model, if CCS data is present in the cell libraries; otherwise, the NLDM model is used.

The advanced CCS timing model has many advantages, one of which is the solution to the problem described by the RC-009 warning message. This warning occurs when the drive resistance of the driver model is much less than the network impedance to ground. The CCS timing model is also better at handling the Miller Effect, dynamic IR drop, and multivoltage analysis.

Nonlinear Delay Models

The nonlinear delay model (NLDM) is the earlier, established method of representing the driver and receiver of a path stage. The driver model uses a linear voltage ramp in series with a resistor (a Thevenin model), as shown in [Figure 9-2](#). The resistor helps smooth out the voltage ramp so that the resulting driver waveform is similar to the curvature of the actual driver driving the RC network.

Figure 9-2 NLDM Driver and Receiver Models



The driver model has three model parameters: the drive resistance R_d , the ramp start time t_z , and the ramp duration delta t . PrimeTime chooses parameter values to match the output waveforms as closely as possible. It builds a different simplified driver model for each gate timing arc (for example, from U1/A to U1/Z) and for each sense (for example, rising edge).

When the drive resistor is much less than the impedance of the network to ground, the smoothing effect is reduced, potentially reducing the accuracy of RC delay calculation. When this condition occurs, PrimeTime adjusts the drive resistance to improve accuracy and issues an RC-009 warning.

The NLDM receiver model is a capacitor that represents the load capacitance of the receiver input. A different capacitance value can apply to different conditions such as the rising and falling transitions or the minimum and maximum timing analysis. A single capacitance value, however, applies to a given timing check, which does not support accurate modeling of the Miller Effect.

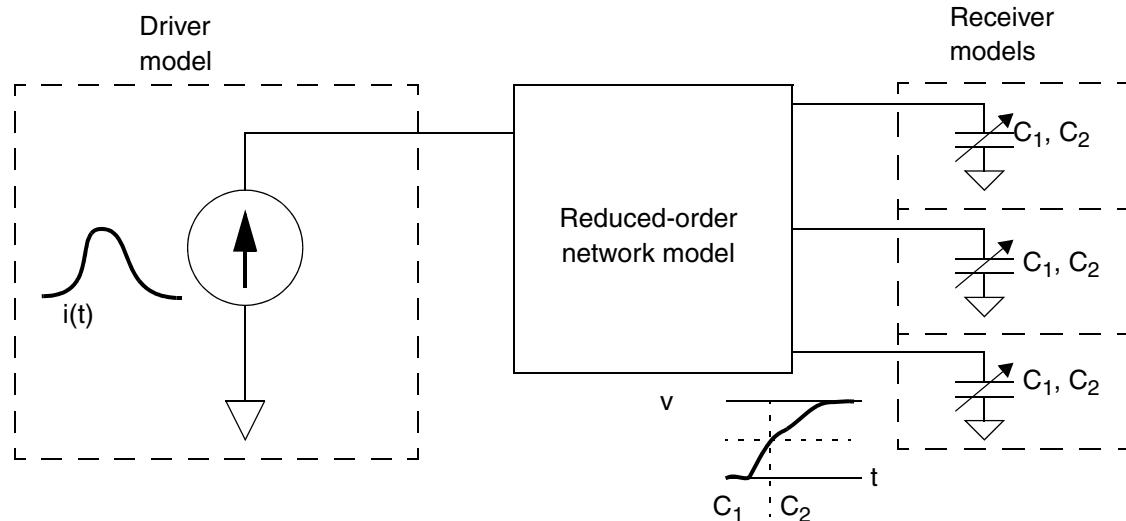
Note:

The Miller Effect is the effective change in capacitance between transistor terminals that occurs with a change in voltage across those terminals.

Composite Current Source Timing Models

With the advent of smaller nanometer technologies, the Composite Current Source (CCS) timing approach of modeling cell behavior has been developed to address the effects of deep submicron processes. The driver model uses a time-varying current source, as shown in [Figure 9-3](#). The advantage of this driver model is its ability to handle high-impedance nets and other nonmonotonic behavior accurately.

Figure 9-3 CCS Timing Driver and Receiver Models



The CCS timing receiver model uses two different capacitor values rather than a single lumped capacitance. The first capacitance is used as the load up to the input delay threshold. When the input waveform reaches this threshold, the load is dynamically adjusted to the second capacitance value. This model provides a much better approximation of loading effects in the presence of the Miller Effect.

In some cases, different input signals can affect the input capacitance of the receiver. Conditional pin-based models are used in the library to describe the pin capacitance for different input signals. If there are conditional pin-based receiver models in the library, PrimeTime considers all receiver models and chooses the worst among the enabled pin-based and arc-based receiver models for the analysis.

In PrimeTime, the CCS timing analysis requires detailed parasitics. It uses library information in the following order for delay calculation with detailed parasitics:

1. CCS timing driver and receiver models, if both are available.
2. CCS timing driver model, if available, and lumped pin capacitance for the receiving cells.
3. NLDM delay and transition tables and pin capacitance for the receiving cells. (Use a CCS timing receiver model only with a CCS timing driver model.)

After a timing update, to determine whether CCS timing data was used for delay calculation, you can use the `report_delay_calculation` command. In the command, specify the cell or net by using the `-from` and `-to` options, or specify a timing arc using the `-of_objects` option. The following examples show CCS timing driver and receiver model data used for a cell and net delay calculation.

```
pt_shell> report_delay_calculation -from cell1/A -to cell1/z
...
arc sense: positive_unate
arc type: cell

Calculation   Rise     Rise     Fall     Fall     Slew     Rail
Thresholds:  Delay    Slew    Delay    Slew    Derate   Voltage Temp.
-----
from-pin  50      30->70  50      70->30  0.400    1.100    125.0
          to-pin  50      30->70  50      70->30  0.400    1.100    125.0

RC network on pin 'cell1/Z' :
-----
Number of elements = 8 Capacitances + 7 Resistances
Total capacitance = 0.03062 pF
Total capacitance = 0.03062 (in library unit)
Total resistance  = 0.017983 Kohm
```

Advanced driver-modeling used for rise and fall.

```
Rise          Fall
-----
Input transition time = 0.100000  0.100000 (in library unit)
Effective capacitance = 0.002625  0.002800 (in pF)
Effective capacitance = 0.002625  0.002800 (in library unit)
Output transition time = 0.060388  0.047040 (in library unit)
Cell delay           = 0.050937  0.045125 (in library unit)

pt_shell> report_delay_calculation -from [get_pins cell/Z] \
           -to [get_pins receiver/A]
...
From pin: cell/Z
To pin: receiver/A
Main Library Units: 1ns 1pF 1kOhm
arc sense: unate
arc type: net
Calculation Rise  Rise  Fall  Fall  Slew   Rail
Thresholds: Delay Slew  Delay Slew  Derate Voltage Temp
-----
from-pin 50    30->70  50    70->30  1.000  0.900  125.0
to-pin   50    30->70  50    70->30  1.000  0.900  125.0

RC network on pin 'cell/Z' :
-----
Number of elements = 8 Capacitances + 7 Resistances
Total capacitance = 0.003062 pF
Total capacitance = 0.003062 (in library unit)
Total resistance = 0.017983 Kohm
Advanced receiver-modeling used for rise and fall.

Advanced Receiver Model
Rise          Fall
-----
Receiver model capacitance 1 = 0.001966  0.001888 (in library unit)
Receiver model capacitance 2 = 0.002328  0.002160 (in library unit)

Rise          Fall
-----
Net delay      = 0.000024  0.000064 (in library unit)
Transition time = 0.059192  0.037599 (in library unit)
From_pin transition time = 0.059078  0.037666 (in library unit)
To_pin transition time = 0.059192  0.037599 (in library unit)
Net slew degradation = 0.000114 -0.000067 (in library unit)
```

Pin Capacitance Reporting

Timing modeling with CCS consists of a driver model and a receiver model. As the input capacitance of a cell varies during the input signal transition, the CCS receiver model uses two capacitances, C1 and C2, to model this variation and guarantee accuracy. C1 and C2 correspond to the input capacitances before and after the delay trip point, respectively.

Typically, the values of C1 and C2 are functions of input slew and output load. Therefore, it is possible that the values of C1 and C2 are larger than the library pin capacitance that is derived from CCS receiver models.

By default, the CCS receiver model information is used to compute the pin capacitance behaviors reported for load pins. The CCS receiver models are used for checking maximum capacitance violations, which ensures that all possible load extrapolations reported by RC-011 can be reported by the `report_constraint` command. The actual total capacitance is defined as the sum of wire capacitance and the maximum across each pin's (C1, C2) capacitance.

The following commands reflect this CCS pin capacitance reporting behavior:

- `report_attribute`
- `report_constraint -min_capacitance and -max_capacitance`
- `report_delay_calculation`
- `report_net`
- `report_timing -capacitance`

The following attributes reflect this CCS pin capacitance reporting behavior:

- `total_ccs_capacitance_max_fall`
- `total_ccs_capacitance_max_rise`
- `total_ccs_capacitance_min_fall`
- `total_ccs_capacitance_min_rise`

This reporting behavior is controlled by the `report_capacitance_use_ccs_receiver_model` variable, which has a default of `true`. When you set this variable to `false`, the previous method of reporting library-derived lumped pin capacitances is used.

Note:

There might be inconsistencies with reporting capacitance in PrimeTime, IC Compiler, and Design Compiler. In PrimeTime PX, there might be a mismatch of capacitance values between the reports displayed when using the `report_delay_calculation` and `report_power_calculation` commands.

Guidelines for Characterizing Design Rule Constraints

The `max_transition` pin attributes are normally present on the input and output pins of library cells. For input pins, the `max_transition` attribute value should not exceed the maximum slew index in the NLDM and CCS driver and CCS receiver_capacitance2 tables.

Use the lowest value of the maximum slew index between the NLDM and CCS tables as a reference. The tables used as reference are for the rising and falling timing arcs from the relevant input pin for which the `max_transition` attribute is being characterized. Take both the arc-based and pin-based tables into account.

The `max_capacitance` pin attributes are normally present on the output pins of library cells. For output pins, the `max_capacitance` attribute value should not exceed the maximum load index in the NLDM and CCS driver as well as CCS receiver_capacitance1 and receiver_capacitance2 tables. Use the lowest value of the maximum load index between the NLDM and CCS tables as a reference. The tables used as reference are for the rising and falling timing arcs to the relevant output pin for which the `max_capacitance` attribute is being characterized. Take both the arc-based and pin-based tables into account.

Resolving the CCS Extrapolation Warning Message (RC-011)

Because large driver or receiver load extrapolations can cause inaccurate results, the tool issues an RC-011 message when it attempts to calculate an RC delay with a slew or load that is

- Smaller than the minimum library slew or load indexes

To resolve this issue, add a small first slew index or load index in the library.

- Larger than the maximum library slew or load indexes

If you use libraries that follow the [Guidelines for Characterizing Design Rule Constraints](#), resolve this issue by fixing the `max_transition` and `max_capacitance` violations reported by the `report_constraint` command. To ensure that all the `max_capacitance` violations in RC-011 are reported by the `report_constraint` command, set the `report_capacitance_use_ccs_receiver_model` variable to `true`.

If the libraries are not compliant with the [Guidelines for Characterizing Design Rule Constraints](#), consider RC-011 warning messages to be important. You need to address the design rule constraints to fix these warnings.

CCS Cross-Library Scaling and Exact Matching

PrimeTime performs voltage and temperature scaling by interpolating data in libraries that was characterized at different voltage and temperature corners. This cross-library scaling allows you to analyze timing, noise, and power at voltage and temperature values that are different from that of the corner libraries. Examples of scaling data include the following:

- Composite Current Source (CCS) timing driver and receiver models
- CCS noise data

- Timing and design rule constraints
- Power data used in PrimeTime PX
- Nonlinear delay model (NLDM) delay and slew data – Interpolation is supported, but the level of accuracy does not match that of scaling with CCS models.

Besides modeling voltage and temperature changes, scaling reduces the number of libraries required for the analysis of multivoltage designs and therefore reduces the library characterization effort.

As an alternative to library scaling, you can use library data in an exact-matching flow, where operating conditions must exactly match one of the libraries. In this flow, PrimeTime chooses a matching library based on the operating conditions applied to the specific design cell instances. If no matching library is found, PrimeTime issues an error message and does not perform scaling.

Defining Library Groups for Scaling

Library groups specify the scaling relationships between multiple libraries that are characterized at different voltages and temperatures. To define a library group for scaling, use the `define_scaling_lib_group` command. For example:

```
pt_shell> define_scaling_lib_group \
           {lib_0.9v_0c.db lib_1.05v_0c.db lib_1.3v_0c.db}
```

There is no limit on the number of libraries that you can specify in the command. To cover different portions of the design, use this command multiple times to define multiple library groups. However, each library can be part of only one library group.

Defining Library Groups for Exact Matching

To define a library group for exact matching, use the `define_scaling_lib_group` command with the `-exact_match_only` option:

```
pt_shell> define_scaling_lib_group -exact_match_only \
           {lib_0.9v_0c.db lib_1.0v_0c.db lib_1.1v_0c.db}
```

If no matching corner library is found, the tool issues an SLG-216 warning message. This warning helps you detect mistakenly applied operating conditions.

Guidelines for Scaling and Exact Matching

When working with library groups for scaling and exact matching, follow these guidelines:

1. Before scaling libraries in PrimeTime, you should check the libraries by using the Library Compiler Qualification System for scaling in version J-2014.09-SP2 or later. This tool detects library data inconsistency issues, which you must fix to avoid effects on scaling.
2. Read in the design netlist before using the `define_scaling_lib_group` command.
3. Ensure that only one library in the group exists in the link path, or is the `min_library` of a library in the link path. The remaining libraries are read in automatically during creation of the scaling group. Each library can be part of only one scaling group. A library cannot be concurrently specified in a scaling group and by the `link_path_per_instance` variable.
4. If the design is not already linked, the `define_scaling_lib_group` command automatically links the design and creates scaling relationships between the libraries in each group. If the design is already linked, `define_scaling_lib_group` creates scaling relationships without additional linking.
5. Verify that the libraries contain complete and consistent data. When you use the `define_scaling_lib_group` command, the tool reports the following information:

```
Completing scaling library groups ...
Loading db file './lib/lib_0.9V_0C.db'
Loading db file './lib/lib_1.05V_0C.db'
... the scaling library groups are complete.
```

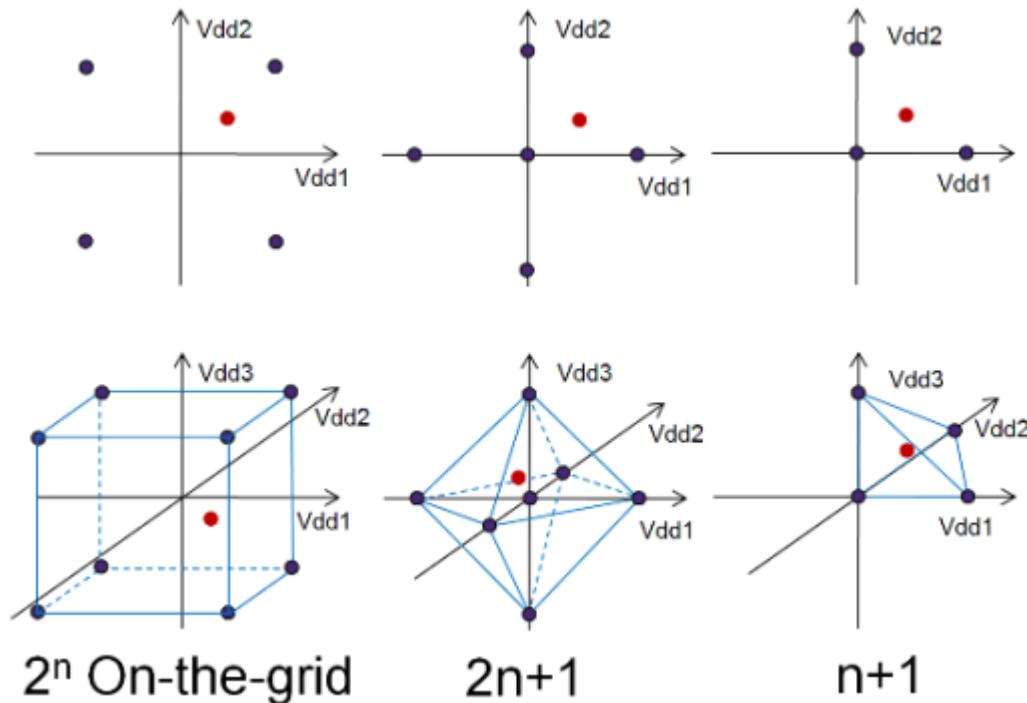
During this process, PrimeTime checks the libraries for consistency. The cell names, pin names, and timing arcs need to be identical between libraries. In addition, all libraries in the scaling group must have consistent library, arc, and pin data. For example, for a specific library pin, either all libraries have a CCS receiver model, or all libraries do not have any CCS receiver models.

PrimeTime checks the following:

- Basic scaling library data consistency to ensure that no data is missing from certain corner libraries. If missing data is detected, SLG warning and errors are issued. The checking prevents calculation failures. For more detailed library checks, run the Library Compiler scaling checking. The Library Compiler version J-2014.09-SP1 has more comprehensive scaling checks and can directly create scaling groups for use in PrimeTime.
- Scaling library formation in scaling mode to ensure that the voltage, and temperature characterization points of the libraries in the scaling group meets one of the three supported scaling formations. If libraries are missing, and the formation requirements cannot be met, scaling cannot be performed. For example, general two-dimensional scaling requires at least three libraries. PrimeTime currently supports the three

scaling formations shown in [Figure 9-4](#). Note that voltage and temperature are treated as a separate scaling dimensions. In an exact-matching flow, this formation requirement is waived.

Figure 9-4 Three Types of Scaling Formations



- Set the supply voltages and temperature of each cell by using the `create_operating_conditions` and `set_operating_conditions` commands. If you are using IEEE 1801, also known as the Unified Power Format (UPF), as part of a multivoltage design, use the `set_voltage` and `set_temperature` commands. Ensure that each specified voltage and temperature value is within the range of the applicable library group for scaling or corresponds to one of the libraries in an exact-matching library group.

- Check the operating conditions of the design by using the `check_timing` command:

```
check_timing -override_defaults operating_conditions
```

This operating condition check detects the following:

- For exact-matching library groups, issues a warning when operating conditions do not match any of the libraries in the scaling group
- For scaling library groups, issues a warning when the operating condition is set beyond the range of the libraries in the scaling group

- For cells that are not part of a library group, issues a warning if the operating condition is set different from the linked library

Use the `check_timing` command before using the `update_timing` command, and fix all reported SLG and operating condition issues. Many of those issues are simple Tcl script errors or straight-forward library problems but could severely affect QoR if left unresolved.

8. If a library is removed from memory with the `remove_lib` command, the affected library group is removed without warning.

For information about how the tool scales data for delay calculation, use the `report_delay_calculation` command. It reports which rise and fall transitions used scaling and which libraries were used for delay calculation. When used with the `-thresholds` option, it also reports cell instance operating condition and rail names. For example:

```
pt_shell> report_delay_calculation -from [get_pins cell/A] \
           -to [get_pins cell/Z] -thresholds
...
arc sense: negative_unate
arc type: cell
...
OC_and_rail_name    max_value   min_value   Lib#0      Lib#1
Temperature          125.00     125.00     125.00     125.00
VBN                 0.00       0.00       0.00       0.00
VBP                 0.88       0.88       0.95       0.85
VDD                 0.88       0.88       0.95       0.85
VSS                 0.00       0.00       0.00       0.00
...
RC network on pin 'cell/Z' :
-----
Number of elements = 4 Capacitances + 3 Resistances
Total capacitance = 0.015272 pF
Total capacitance = 15.271626 (in library unit)
Total resistance  = 0.027793 Kohm
Advanced driver-modeling used for rise and fall.
Scaling library arc group used for rise and fall.
Scaling libraries used for driver model :
  /remote/testcase/lib/ccs_lib_0.95.db:ccs_0.95
  /remote/testcase/lib/ccs_lib_0.85.db:ccs_0.85

          Rise      Fall
-----
Input transition time = 0.600000 0.600000 (in library unit)
Effective capacitance = 0.015272 0.015272 (in pF)
Effective capacitance = 15.271626 15.271626 (in library unit)
Output transition time = 0.091507 0.088209 (in library unit)
Cell delay            = 0.031275 0.112788 (in library unit)
```

To report all libraries with scaling relationships, use the `report_lib_groups` command. For example:

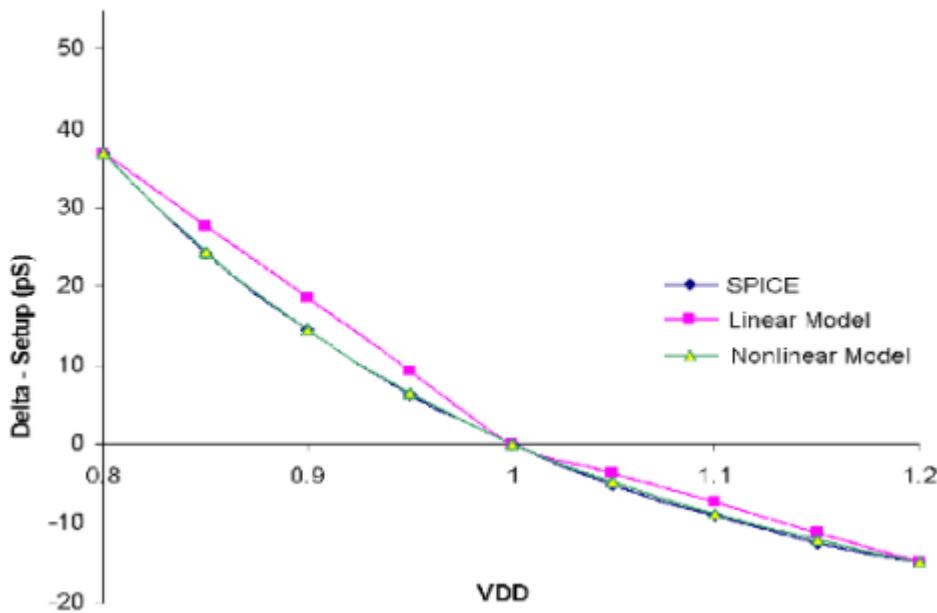
```
pt_shell> report_lib_groups -scaling -show {voltage temperature}
...
      Group      Library      Temperature      Voltage
-----
Group 1
lib_0.85      125.00        0.85
lib_0.95      125.00        0.95
```

Scaling Interpolation for Timing Constraints

For setup, hold, recovery, removal, minimum pulse width, and minimum period constraints, PrimeTime uses a nonlinear interpolation method for voltage variation. [Figure 9-5](#) shows a comparison of SPICE data, linear interpolation, and nonlinear interpolation for a D flip-flop setup constraint in the voltage range between 0.8 and 1.2 volts. The nonlinear interpolation method used by PrimeTime produces more accurate results than linear interpolation.

Figure 9-5 Nonlinear Interpolation of Timing Constraint

Setup Rising ck-sinp=.018 d-sinp=.17 10%



Scaling for Design Rule Constraints

If scaling occurs, the `report_constraint` command interpolates the design rule constraint values between the scaling libraries that were used for calculation on the corresponding pin. Design rule constraint support for Composite Current Source (CCS) scaling eliminates any possibility of missed design rule constraint violations.

Scaling for Multirail Cells

PrimeTime can perform scaling for multirail cells, such as level shifters. Level shifter cells operate across voltage differences, connecting driver and load pins of cells belonging to different power domains and placed in different voltage areas. Level shifter cells are frequently used in a multivoltage design. They are used to connect the different voltage power domains by stepping the voltage up or down, as needed. Accurate multirail scaling enables PrimeTime to analyze the dependencies on multiple variables, including multiple rail voltages and temperature.

PrimeTime can perform multirail cell scaling on timing, noise, and power data. This capability supports scaling library groups of power and ground (PG) pin-based CCS libraries, non-PG libraries, legacy rail_connection libraries, and power domain mode commands with multirail cells; all other scaling library groups are supported using single-rail scaling.

By default, multirail scaling is enabled for timing, noise, and power analysis. PrimeTime automatically detects a multirail scenario and applies multirail scaling when there are enough libraries in the scaling library group to support the required scaling formation for the rail dimensions.

To set up multirail scaling:

1. Set up scaling library groups to support voltage and temperature scaling by using the `define_scaling_lib_group` command.

Ensure that there is at least one scaling library group for each scaling scenario. Treat scaling with different VT and rail dimensions as different scaling scenarios. For example, have at least one scaling library group for regular one-rail cells and another for two-rail level shifters that require scaling in more dimensions. For example:

```
pt_shell> define_scaling_lib_group {std_lib_0.9v.db std_lib_1.1v.db}
pt_shell> define_scaling_lib_group {LS_lib_0.9V_0.9V.db \
    LS_lib_0.9V_1.1V.db \
    LS_lib_1.1V_0.9V.db \
    LS_lib_1.1V_1.1V.db }
```

2. Put cells that require scaling in different dimensions into different libraries, and put those libraries into different scaling library groups. For example, two-rail level shifter cells require at least eight libraries to perform voltage and temperature scaling if you are using

the on-the-grid formation. All other one-rail cells that do not need to consider underdrive and overdrive can be put in one library, and only four libraries are needed to perform simultaneous voltage and temperature scaling of these one-rail cells. If this condition is not met, error message is issued.

3. Set the design instance operating conditions by using the `set_voltage`, `set_rail_voltage`, `set_operating_conditions`, and `set_temperature` commands.
4. Report the library groups by using the `report_lib_groups` command. This report displays voltage rail information in the library scaling groups if multirail scaling is supported for that library group. For example:

```
pt_shell> report_lib_groups -scaling -show {voltage temp process}
*****
Report : lib_groups
          -scaling
          -show
...
*****
-----
```

Group	Library	Temperature	Voltage	Process
Group 1	mylib_wc_ccs	125.00	1.08	1.00
	mylib_wc0d72_ccs	125.00	0.86	1.00
Group 2	mylib_lvtwc0d720d72_ccs	125.00	{ V:0.86 VL:0.86 }	1.00
	mylib_lvtwc0d720d9_ccs	125.00	{ V:1.08 VL:0.86 }	1.00
	mylib_lvtwc0d90d72_ccs	125.00	{ VL:0.86 V:1.08 }	1.00
	mylib_lvtwc0d90d9_ccs	125.00	{ V:1.08 VL:1.08 }	1.00
Group 3	mylib_lvtwc_ccs	125.00	1.08	1.00
	mylib_lvtwc0d72_ccs	125.00	0.86	1.00

Before using multirail scaling, ensure that the libraries meet the following requirements:

- Library operation conditions in a scaling group should be in on-the-grid, $2n+1$, or $n+1$ type of library group formations, which are currently acceptable by the multirail scaling feature. On-the-grid library formation means the operation conditions of the libraries have to be on the grid in the Cartesian coordinate system. This requires at least $2n$ libraries for n dimensional scaling. In the preceding example, the second scaling library for two-rail level shifters is using on-the-grid library formation. The $2n+1$ type of library formation has a nominal library in the middle. In addition, there is one plus side library and one minus side library in each dimension. Therefore, the set of libraries have a total of $2n+1$ libraries. The $n+1$ type of library formation is similar to the $2n+1$ type, but it only has the plus or minus side. If the library operating condition distribution is improper, error message is issued and scaling is not performed.
- Cells that require scaling in different dimensions are put into different libraries. For example, two-rail level shifter cells require at least eight libraries to perform voltage and

temperature scaling, if you are using the on-the-grid formation. All other one-rail cells that do not need to consider underdrive and overdrive can be put in one library, and only four libraries are needed to perform simultaneous voltage and temperature scaling of these one-rail cells. If this condition is not met, error message is issued.

- PrimeTime and PrimeTime PX provides the capability to convert and update library power and ground (PG) pins and non-PG pin libraries through the UPF library PG pin conversion.
- Libraries can contain timing, noise, and power information. Multirail scaling is applied to each of these data types of multirail cells.
- Expanded CCS libraries, CCS libraries using the base curve technology format, and NLDM-only libraries are supported.

Fast Multidrive Delay Analysis

In large designs with annotated parasitics, the runtime for RC delay calculation can be large for massively multidriven networks. The primary bottleneck in this process is the runtime required to compute the trip times for measuring delays and slews at the load pins.

For massively multidriven networks with homogeneous drivers (drivers with similar input skews, slews, and operating conditions), a fast multidrive calculation mode significantly reduces the runtime. This mode shorts all of the driver nodes together with a network of resistors and uses a single driver model at the first driver node for all waveform calculations. The drive strength of the single driver is scaled up to be equivalent to the whole set of original drivers. The delay calculation results for the single driver are copied to all of the other drivers, including delay, slew, driver model parameters, and effective capacitance.

The maximum number of network drivers that are handled without invoking the fast multidrive analysis mode is nine. The presence of ten or more parallel drivers invokes the fast multidrive analysis mode.

When PrimeTime uses the fast multidrive analysis mode, it generates an RC-010 warning message that reports the number of drivers, the number of loads, the input slew and input skew spreads, and the matching or mismatching of driver library timing arcs and driver operating conditions. This information can help determine the accuracy of the analysis.

The accuracy of this mode is best when the driver library arcs are the same and operate under the same context (operating conditions, input slew, input skew, and so on), and operate with the same network impedance. These conditions are often typical of mesh networks. Any differences in operating conditions and input slews can cause small differences in output delay and output slew. For accuracy, the differences in input skews should be small compared to the delays through the network. For higher accuracy, you can calculate the delays and slews with an external tool and annotate them onto the network.

For more information, see the following topics:

- [Parallel Driver Reduction](#)
- [Reducing SDF for Clock Mesh/Spine Networks](#)

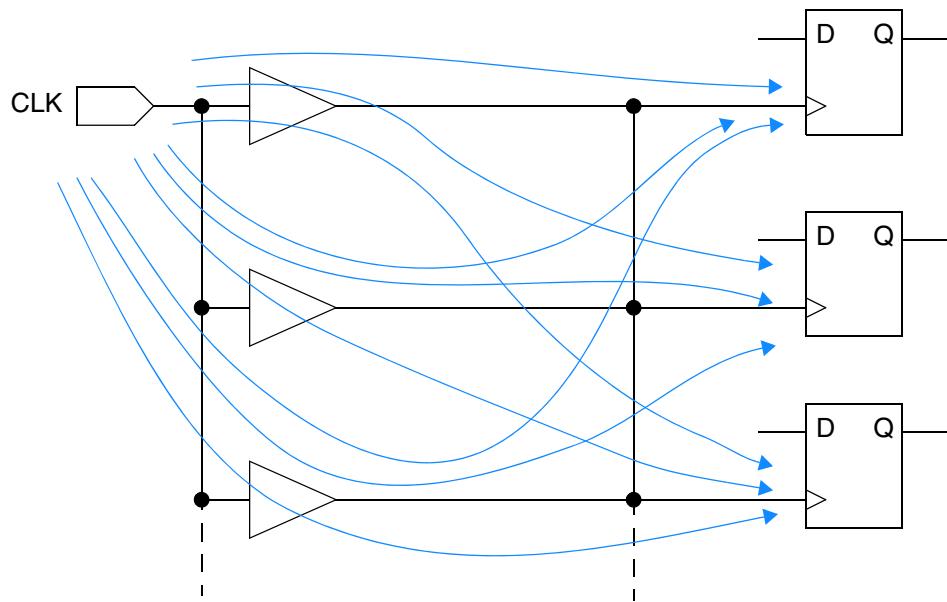
Parallel Driver Reduction

Because clock signals must typically drive a large number of loads, clocks are often buffered to provide the drive strength necessary to reduce clock skew to an acceptable level. A large clock network might use a large number of drivers operating in parallel. These drivers can be organized in a “mesh” or “spine” pattern to distribute the signal throughout the chip.

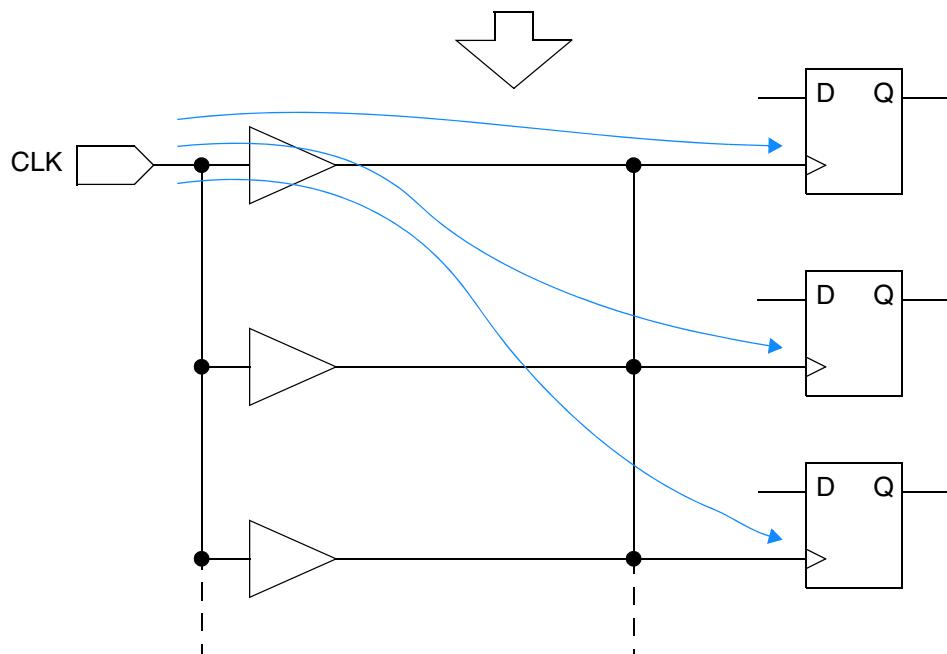
If a design has 1,000 drivers in parallel driving 1,000 loads, PrimeTime must keep track of one million different timing arcs between the drivers and loads. Timing analysis of such a network can consume a large amount of CPU and memory resources.

For better performance, PrimeTime can reduce the number of timing arcs that must be analyzed. [Figure 9-6](#) shows the reduction process. When the reduction feature is enabled, PrimeTime selects one driver in a parallel network and analyzes the timing arcs through that driver only.

Figure 9-6 Parallel Driver Reduction



```
pt_shell> set_app_var timing_reduce_multi_drive_net_arcs true
pt_shell> set_app_var timing_reduce_multi_drive_net_arcs_threshold 1000
pt_shell> link_design
```



Invoking Parallel Driver Reduction

To enable parallel driver reduction, set the `timing_reduce_multi_drive_net_arcs` variable to `true`. By default, parallel driver reduction is disabled. During design linking, the tool checks for the presence of nets with multiple drivers. If the tool finds a net with driver-load combinations exceeding the threshold specified by the `timing_reduce_multi_drive_net_arcs_threshold` variable, it reduces the number of timing arcs associated with the drivers of that net.

The `timing_reduce_multi_drive_net_arcs_threshold` variable specifies the minimum number of timing arcs that must be present to trigger a reduction. By default, it is set to 10,000, which means that PrimeTime reduces the timing arcs of a net if the number of drivers multiplied by the number of loads on the net is more than 10,000. In typical designs, this large number only occurs in clock networks.

The tool performs driver reduction for a net when all of the following conditions are true:

- The number of driver-load combinations is more than the variable-specified threshold (10,000 by default).
- Driver cells are nonsequential library cells (not flip-flops or latches and not hierarchical).
- All drivers of the net are instances of the same library cell.
- All the drivers are connected to the same input and output nets.

To expand a reduced network to its original form or to perform driver reduction with a different threshold, you must relink the design.

Working With Reduced Drivers

When layout is complete and detailed parasitic data is available, it is not possible to annotate this data on a reduced network. To get accurate results, use an external simulator such as SPICE to get detailed delay information for the network. Then you can annotate the clock latency values and transition times on the individual clock pins of the sequential cells, while still allowing PrimeTime to treat the reduced network as ideal, with zero delay. This technique provides reasonably accurate results, while being very efficient because the clock network timing only needs to be analyzed one time, even for multiple PrimeTime analysis runs.

If you back-annotate the design with the `read_sdf` command, any annotations on the reduced timing arcs are ignored. PrimeTime issues a PTE-048 message when this happens. Suppress these messages with the following command:

```
pt_shell> suppress_message PTE-048
```

If you write out SDF with the `write_sdf` command, no interconnect delays are generated for the reduced network.

Reducing parallel drivers only affects the timing arcs analyzed by PrimeTime, not the netlist. Therefore, it does not affect the output of the `write_changes` command. For information about RC delay calculation with massively multidriven networks, see [Fast Multidrive Delay Analysis](#).

Path-Based Timing Analysis

Static timing analysis tools are designed to be pessimistic to ensure detection of all timing violations. For example, PrimeTime considers both the worst arrival time and worst slew among all signals feeding into a path, even if the signal with the worst arrival time is different from the one with the worst slew.

You can reduce the pessimism for critical paths by analyzing those paths in isolation from other paths. This method is called path-based timing analysis. When recalculating a timing path, PrimeTime propagates the edge along each path of interest, ignoring slews from side arcs along the path, and it performs delay calculation to compute the path-specific slack. Thus, path-based analysis recalculates the timing in a timing path without considering outside paths that might otherwise affect the arrival times and slew values used in the calculation. The command annotates the path collection with new timing information such as arrival times, slews, and slack.

Timing path effects, such as clock reconvergence pessimism removal and crosstalk effects (delta delays), are also recomputed specifically to that path. Path-based analysis is useful when there are only a few violations remaining in the analysis, and you want to find out whether these violations are caused by pessimistic analysis of arrival and slew times.

One important property of path-based analysis is that the slack ordering of paths can change due to the path recalculation. In other words, the most critical path before recalculation might not be the most critical path after recalculation. Because of this property, multiple paths to an endpoint might need to be recalculated to determine the true post-recalculation critical path. It cannot be emphasized enough that recalculating the single worst failing path to an endpoint does not provide the worst recalculated slack to that endpoint.

There are two ways to recalculate a specific path or set of paths:

- Use the `report_timing` or `get_timing_paths` command with the `-pba_mode path` option. No path search is performed to determine whether those paths are truly the worst recalculated paths. This method is useful for quickly gaining an idea of how much improvement is provided by path-based analysis without the runtime of a full path search. The slack results either match those from a full path search or are slightly optimistic.

- Use the `report_timing` command with the `-pba_mode exhaustive` option. PrimeTime performs an exhaustive recalculated path search, recalculating as many paths as necessary to ensure that the paths returned are the worst recalculated paths that meet the specified options.

To ensure optimal path search runtime, the following usage is recommended:

- Use the `-pba_mode exhaustive` option with the `-slack_lesser_than` option to keep the search bounded.
- Perform an exhaustive path search only when `-pba_mode path` shows that the slacks are reasonably close to the desired threshold.
- Ensure that the reporting is as specific as possible. If reports are desired only for specific path groups or startpoints and endpoints, specify the required reporting options.
- Enable worst parallel-arc reporting by setting the `timing_report_use_worst_parallel_cell_arc` variable to `true`.

To see the critical path in each path group while applying exhaustive path-based recalculation, use this command:

```
pt_shell> report_timing -pba_mode exhaustive -slack_lesser_than 0
```

In the earlier example, if no paths are reported with negative slack, the design has no timing violations.

Path-based recalculation can also be combined with other timing path selection options, such as `-from/-through/-to` or `-max_paths/-nworst`. For `-pba_mode path`, the `-slack_lesser_than` option applies to the original slack used to obtain the paths. The reported recalculated slack might be improved.

When using `-max_paths` or `-nworst` with `-pba_mode exhaustive`, the path limits represent the desired path counts at the completion of the search. Additional paths are searched during the path search process to ensure a complete result. For example, to report the worst path to each violating endpoint with recalculation taken into account, enter:

```
pt_shell> report_timing -pba_mode exhaustive -nworst 1 \
           -slack_lesser_than 0 -max_paths 1000
```

Multiple paths to each failing endpoint are searched to find the worst recalculated failing path.

The `report_timing -pba_mode exhaustive` option does not support the `-start_end_pair` and `-slack_greater_than` options.

You can also use the `get_timing_paths -pba_mode path` command to recalculate a collection of timing paths. When passed a collection of timing paths with this option, the collection is recalculated and returned. For example,

```
pt_shell> set paths [get_timing_paths -max_paths 10]
pt_shell> set recalc_paths [get_timing_paths -pba_mode path $paths]
```

After a set of timing paths have been recalculated, use the `report_timing` command to report the recalculated timing values of the path collection. For example,

```
pt_shell> report_timing $recalc_paths
```

The recalculated values are accessible only by the `report_timing` and `get_attribute` commands, not by other commands, such as the `report_constraint` command. However, you can use the `report_attribute` command to report the newly calculated attribute values directly inside the path collection.

The `timing_path` objects have the `is_recalculated` Boolean attribute, which indicates whether the timing path has been recalculated. The `report_timing` command reports that a timing path is recalculated in the header text for the path, as shown in the following example:

```
*****
Report : timing
        -path full
        -delay max
        -max_paths 1000
...
*****
Startpoint: c (input port)
Endpoint:   z2 (output port)
Path Group: default
Path Type:  max (recalculated)
```

To control whether clock paths and latch-based borrowing paths are recalculated, set the `pba_recalculate_full_path` variable. When the variable is set to `false` (the default), clock paths and borrowing paths are never recalculated. Only the data portion of the path is recalculated. When you set the variable to `true`, the clock path and borrowing paths are always recalculated.

The `-path full_clock_expanded` option of the `report_timing` command indicates only that the clock path is included in the timing report. This option does not control whether the clock path is recalculated or not. If the `pba_recalculate_full_path` variable is set to `true` and the `-path full_clock_expanded` option is not specified, the clock path is obtained and recalculated; however, the expanded clock path is not shown in the report.

The `-path full_clock_expanded` option of the `get_timing_paths` command specifies that the clock path is obtained and stored along with the timing path collection objects. If the `pba_recalculate_full_path` variable is set to `true` and recalculated timing paths are obtained directly with the `get_timing_paths -pba_mode` option, the clock path is

recalculated and used to compute the path timing; however, the full clock path is not stored in the timing path collection objects. If a `timing_path` object previously obtained without the `-path full_clock_expanded` option is being recalculated and clock path recalculation is enabled with the `pba_recalculate_full_path` variable set to `true`, an error message is issued indicating that the clock path is needed for correct and consistent results.

Note:

When the `pba_recalculate_full_path` variable is set to `true`, path-based analysis recalculates the borrowing at transparent latches only if the `-trace_latch_borrow` option has been specified. Otherwise, the existing borrowing at the original path's startpoint is used.

The `timing_report_recalculation_status` variable facilitates debugging and eases the runtime concern. It works for both the `get_timing_paths -pba_mode exhaustive` and `report_timing -pba_mode exhaustive` commands.

The amount of time needed for the integrated path search depends on a few factors:

- Amount of slack improvement provided by recalculation
- Number of paths involved in the search (`-slack_lesser_than`, `-from/-through/-to`)
- Number of paths to be returned to the user (`-max_paths` and `-nworst`)

In particular, large `-nworst` values can significantly increase the runtime of the search. To prevent excessive runtime, set the `pba_exhaustive_endpoint_path_limit` variable, which limits the number of endpoints for exhaustive path searching.

When you perform a timing update on the original design (for example, with an `update_timing` command), any timing path collections created by the `get_timing_paths` command are automatically deleted.

Setting Recalculation Limits

You can control the behavior of the recalculation limits in path-based analysis. Path-specific recalculation with the `-pba_mode path` option has a limit of 2000000 paths per recalculation command. Exhaustive recalculation with the `-pba_mode exhaustive` option has a paths-per-endpoint limit controlled by the `pba_exhaustive_endpoint_path_limit` variable, which defaults to 25000.

When either limit is reached, the limiting behavior is controlled by the `pba_path_recalculation_limit_compatibility` variable. When set to its default of `true`, no further paths are obtained or recalculated when the limit is reached. For path-specific recalculation, the remaining paths beyond the limit are discarded. For exhaustive recalculation, the limited endpoint is not included in the search; however, any paths that are already found for that endpoint remain in the search.

When the `pba_path_recalculation_limit_compatibility` variable is set to `false`, graph-based analysis paths are used to fill in the recalculation results past the limit. For path-specific recalculation, the remaining paths beyond the limit retain their original nonrecalculated graph-based timing. For exhaustive recalculation, any additional paths obtained for the limited endpoint retains their original nonrecalculated graph-based timing. In both cases, recalculated paths can be readily identified in the resulting timing reports with the recalculation flag in the path headers.

CCS Receiver Model for Path-Based Analysis

For path-based analysis, PrimeTime uses the actual receiver models in the path being analyzed. For the cells that are side loads, the worst-case receiver models are used. During path-based analysis, receiver models are derived using the effective capacitance (`Ceff`) on the output of the load cell. You save this capacitance for path-based analysis by setting the `rc_cache_min_max_rise_fall_ceff` variable to `true` (the default is `false`).

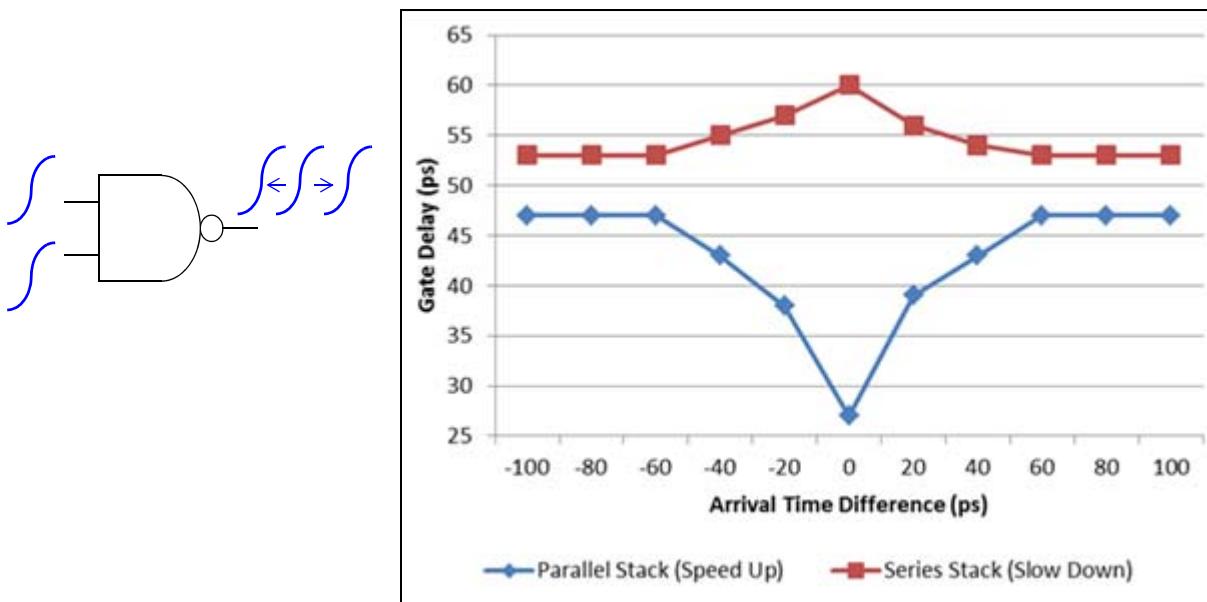
Note:

The `rc_cache_min_max_rise_fall_ceff` variable is automatically set to `true` if you have enabled crosstalk analysis or if you have invoked the GUI before running the `update_timing` command.

Multi-Input Switching Analysis

When multiple inputs of a cell switch simultaneously, the output delay can increase or decrease, depending on the switching direction of the inputs, as shown in [Figure 9-7](#). The delay decrease is more critical because the number of stages on a hold path is small. In a path with many stages, it is unlikely that multi-input switching can occur simultaneously at multiple stages. However, for short paths in hold analysis, the effect of multi-input switching at one stage can cause significant path delay changes. PrimeTime ADV multi-input switching (MIS) analysis considers this delay decrease during hold analysis to improve timing analysis accuracy.

Figure 9-7 Multi-Input Switching Affects the Output Delay and Transition Time



To learn about using PrimeTime ADV multi-input switching analysis, see

- [Enabling Multi-Input Switching Analysis](#)
- [Specifying the Coefficients for Multi-Input Switching Analysis](#)
- [Configuring the Window Alignment and Overlap Checking](#)
- [Multi-Input Switching Analysis Flow Example](#)

Enabling Multi-Input Switching Analysis

To enable multi-input switching analysis, set the following variable:

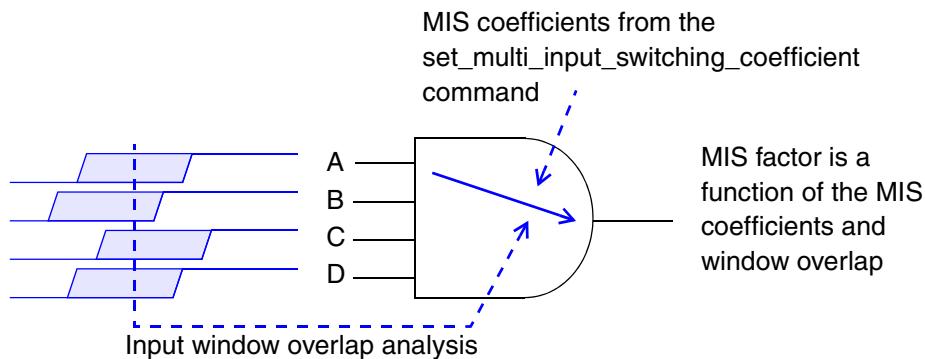
```
pt_shell> set_app_var si_enable_multi_input_switching_analysis true
```

The default of this variable is `false`.

Specifying the Coefficients for Multi-Input Switching Analysis

The tool calculates the actual MIS factors from user-specified delay and delta coefficients and the results of window overlap checking, as shown in [Figure 9-8](#).

Figure 9-8 Actual MIS Factor is Determined by the Window Overlap and MIS Coefficients



Window overlap with other inputs	Actual MIS factor (x)
Full window overlap	$x = \text{MIS coefficient}$
No window overlap	$x = 1$ (no MIS effect)
Partial window overlap	$\text{MIS coefficient} \leq x \leq 1$

To use the coefficients for multi-input switching analysis, use the commands in [Table 9-1](#).

Table 9-1 Commands for Using the Coefficients for Multi-Input Switching Analysis

To do this	Use this command
Specify the coefficients for multi-input switching analysis	set_multi_input_switching_coefficient
Report coefficients for multi-input switching analysis	report_multi_input_switching_coefficient
Reset the coefficients for multi-input switching analysis	reset_multi_input_switching_coefficient

You can specify multi-input switching coefficients for

- A library cell with two or more input pins
- A nonsequential library cell

Note:

If you reset the coefficient of a library cell, the tool does not perform multi-input switching analysis on that cell.

Configuring the Window Alignment and Overlap Checking

By default, PrimeTime ADV performs window alignment and overlap checking for multi-input switching analysis and adds a margin to the arrival window. To disable the default behavior, set the `si_enable_multi_input_switching_timing_window_filter` variable to `false`.

Multi-Input Switching Analysis Flow Example

The following script example shows the flow for multi-input switching analysis:

```
# Read design, coupled parasitics, constraints
set_app_var si_enable_analysis true
set search_path ". All_db"
set_link_path "cell.db"
read_verilog design.v
link_design
read_parasitics -keep_capacitive_coupling design.spf
source design.sdc

# Enable MIS
set_app_var si_enable_multi_input_switching_analysis true
# Specify MIS coefficients
set_multi_input_switching_coefficient \
[get_lib_cell lib1/OAI22X1] -rise -delay 0.45 -slew 0.25

# Update and report timing
update_timing -full
report_delay_calculation -min -from Inst1/A -to Inst1/Z
report_timing -delay_type min -crosstalk_delta -from Inst1/Z -to Inst2/A
```


10

Case and Mode Analysis

To restrict the scope of the timing analysis, you can place the design into specific operating modes by using the following techniques:

- **Case Analysis** – Specifies mode settings with user-specified logic constants or logic transitions on ports or pins.
- **Mode Analysis** – Uses mode settings specified in the library, such as read mode or write mode in a RAM cell.
- **Mode Merging for Scenario Reduction** – Reduces the number of scenarios by merging individual modes into superset modes.

Case Analysis

Case analysis lets you perform timing analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design.

To learn more about case analysis, see

- [Performing Case Analysis](#)
 - [Constant Propagation Based on Cell Logic](#)
 - [Setting Case Analysis Values](#)
 - [Evaluating Conditional Arcs Using Case Analysis](#)
 - [Reporting Case Analysis Values](#)
 - [Removing Case Analysis Values](#)
 - [Constant Propagation Log File](#)
 - [Case Analysis and Maximum Capacitance Checking](#)
-

Performing Case Analysis

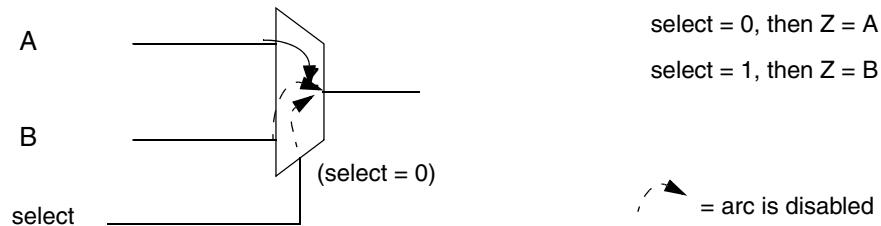
Case analysis lets you perform timing analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design. Setting a case analysis with a logic constant propagates the constant forward through the design, then disables paths where the constant is propagated. Setting a case analysis with a logic transition (rising or falling) eliminates certain paths by limiting the transitions considered during analysis.

When you perform case analysis, keep the following points in mind:

- Case analysis propagates the constant values specified on nets forward (but not backward) through the design.
- Case analysis does not affect sequential cells if the library was compiled with Library Compiler v3.3 or earlier.
- Propagating constants across the sequential cells is disabled by default. You can enable it by setting the `case_analysis_sequential_propagation` variable to `true`.
- Case analysis, by default, propagates user set case analysis logic values as well as logic constants from the netlist. You can disable the propagation of logic constants from the netlist by setting the `disable_case_analysis_ti_hi_lo` variable to `true`. You can disable the propagation of all logic values (from the user and netlist) by setting the `disable_case_analysis` variable to `true` and overrides the value of the `disable_case_analysis_ti_hi_lo` variable.

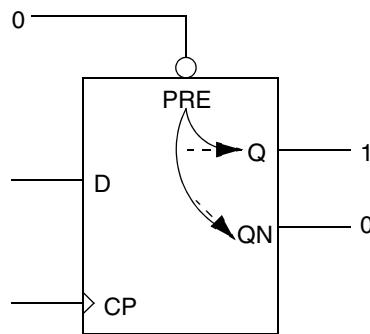
The multiplexer in [Figure 10-1](#) has two signals (A and B) going in, one signal (Z) going out, and 0 is set on a select signal coming into the multiplexer. The select disables the arc from B to Z because the constant is blocking the data from B to Z.

Figure 10-1 Constant Blocking of Data



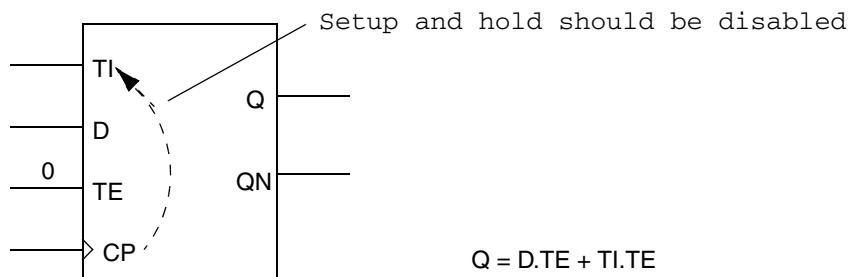
When case analysis propagates a constant to a sequential element's asynchronous preset or clear pin, the sequential cell outputs also propagate the constant 1 or 0, as shown in [Figure 10-2](#).

Figure 10-2 Constant Propagation Through an Asynchronous Input



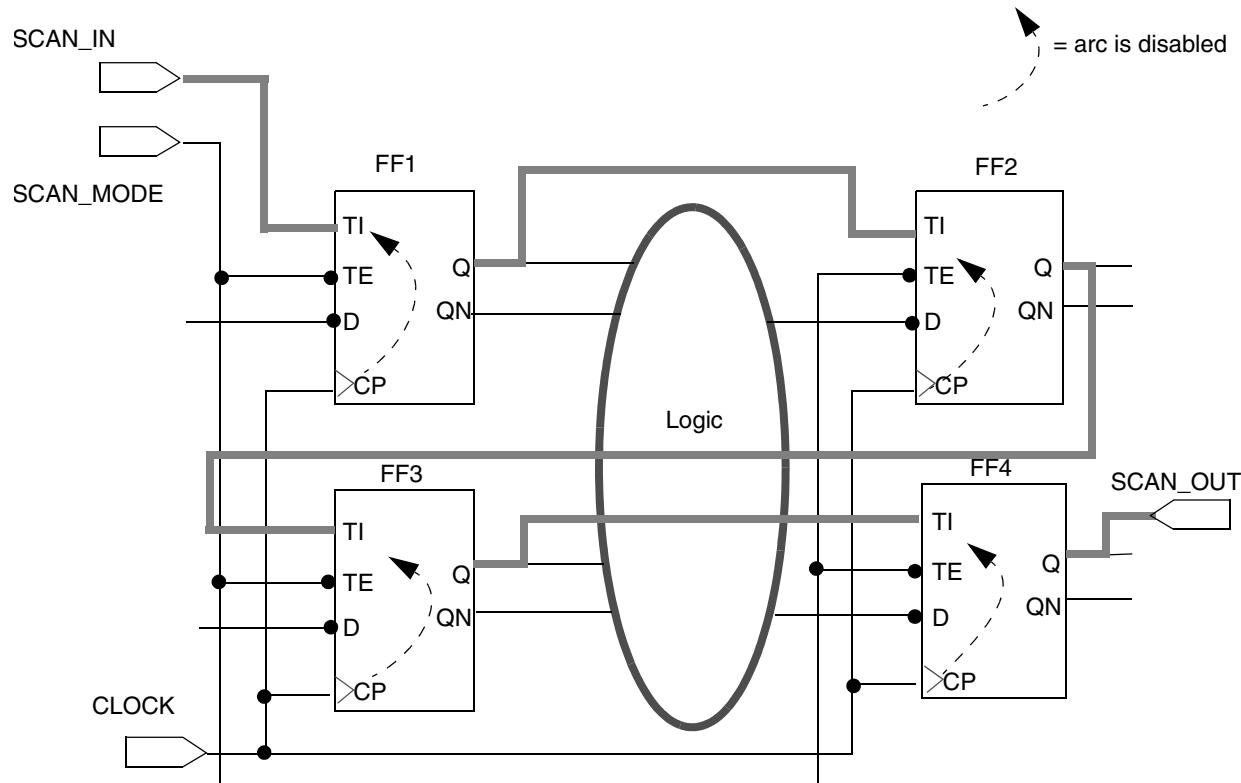
When your design uses scan flip-flops, case analysis is useful for disabling the scan chain. To do this, set the scan mode control pin to the constant value that disables the test mode. PrimeTime propagates the test-mode constant value to the scan-mode pin of each scan flip-flop. Case analysis can determine from the constant values which timing arcs to disable, as shown in [Figure 10-3](#).

Figure 10-3 Case Analysis Disabling Timing Checks



The most common use of case analysis is to disable a scan chain, as shown in [Figure 10-4](#). Setting the SCAN_MODE port to 0 disables the scan chain. As a result, the scan chain is not reported by the `report_timing` command.

Figure 10-4 Using Case Analysis to Disable a Scan Chain



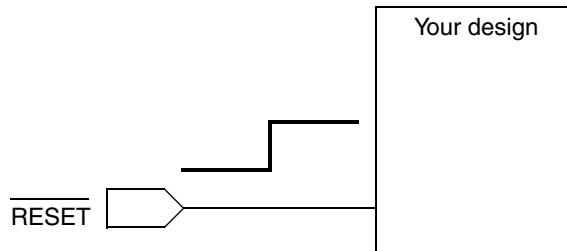
By setting a logic constant of 0 on the **SCAN_MODE** port, PrimeTime disables the scan chain that starts at **SCAN_IN** and ends at **SCAN_OUT**. PrimeTime disables the scan chain because the logic 0 set on the **SCAN_MODE** port propagates to the **FF1/TE**, **FF2/TE**, **FF3/TE**, and **FF4/TE** pins. A logic constant on the **TE** pin of each scan flip-flop disables the setup and hold arcs from **CP** to **TI** because of the sequential case analysis on the **FF1**, **FF2**, **FF3**, and **FF4** cells.

To set case analysis with a 0 value, enter

```
pt_shell> set_case_analysis 0 [get_ports "SCAN_MODE"]
```

Setting a case analysis for a transition can be useful for analyzing the behavior of a circuit for a specific situation. For example, in the circuit shown in [Figure 10-5](#), a rising-edge transition on the reset input brings the device out of the reset mode.

Figure 10-5 Case Analysis for a Rising Edge



To analyze the timing of the circuit coming out of reset mode, you are only concerned about rising edges on the reset input, not logic 0, logic 1, or falling edges. To set case analysis in this situation, enter:

```
pt_shell> set_case_analysis rising [get_ports RESET]
```

Note:

The `rising` and `falling` transition values are ignored by PrimeTime PX during power calculation.

Constant Propagation Based on Cell Logic

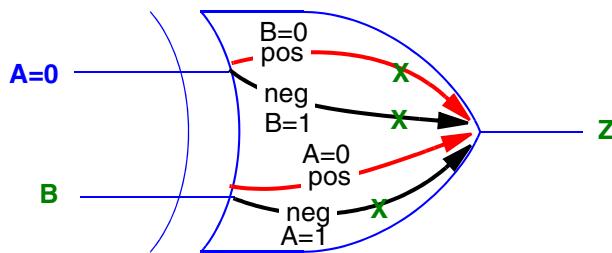
Case analysis can disable timing arcs and propagate case values for these types of cells:

- Combinational cells with the `function` attribute specified in the library description
- Sequential cells with Boolean State Table (BST) specified in the library description
- Cells with timing arcs that have state-dependent conditions specified using the `when` statement in Library Compiler. (See the Library Compiler user guides for the correct usage of the `when` statement and state-dependent delays.)

If a cell has any input at a logic constant case value, arcs that could never propagate a transition are disabled. If any other arcs have a `when` condition, those conditions are evaluated. The `when` arcs are disabled if the `when` expression evaluates to false. Note that the `when` statement cannot enable an arc that has been disabled by case analysis.

Figure 10-6 shows how arcs of an XOR gate are disabled when using the `when` statement in the library description. When $A = 0$, all arcs connected to A , and negative_unate arcs from B to Z are disabled. Note, there might be several arcs of the same sense between the two pins.

Figure 10-6 Disabled Arcs Using the when Statement

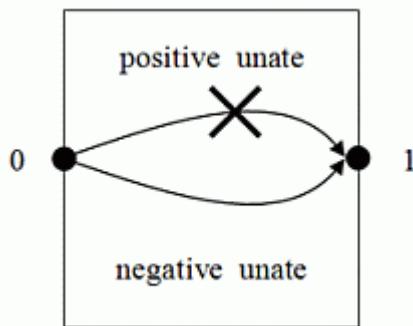


The only enabled arcs are positive_unate arcs from **B** to **Z**.
 (**X** identifies disabled arcs)

- Case analysis on three-state cells allows for disabling timing arcs across a three-state cell. In this context, case analysis uses the associated three-state logic function in the cell library definition for justifying three-state arcs. The three-state logic function can be commonplace in pad cells, so it is desirable to be able to disable the data path ($A \rightarrow Z$) arc when the three-state function turns this path off due to tristating the output.
- Physically invalid arcs

When case analysis propagates logic values from the inputs of a cell to its outputs, it disables arcs that are physically invalid. Arcs are considered physically invalid if the logic value on the input and output do not align with the sense of that arc. For example, the cell below has a logic value 0 on its input and a logic value 1 on its output. The sense of this combination of logic values is negative unate. Therefore, the positive unate arc between the pins is physically invalid and is disabled by case analysis as shown in [Figure 10-7](#).

Figure 10-7 Physically Invalid Arc



Case Analysis of Integrated Clock-Gating Cells

As defined by the Library Compiler `clock_gating_integrated_cell` attribute, there are effectively three broad categories of integrated clock-gating cells, distinguished by the internal memory element they use. These integrated clock-gating cells are:

- Latch memory element
- Flip-flop memory element
- No memory element

PrimeTime currently supports latched integrated clock-gating cells. These cells are sequential in nature. To enable the propagation of logic values from the input pins of integrated clock-gating cells to their fanout, set the value of the `case_analysis_propagate_through_icg` variable to `true`. To disable the propagation logic values and disable simulation, set the variable back to `false` (the default). Integrated clock-gating cells that have no memory element are not sequential; therefore, they are considered combinational and always propagate.

Two possible situations can arise, regardless of whether the enable pin is active or not:

- If a logic value reaches the input clock pin of the integrated clock-gating cell, it does not propagate to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin does not propagate to the gated clock output pin and then into the fanout of the cell.
- If no logic value reaches the input clock pin of the integrated clock-gating cell, no logic value is propagated to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin is propagated to the gated clock output pin and then into the fanout of the cell.

When the `case_analysis_propagate_through_icg` variable is set to `true`, integrated clock-gating cells are fully simulated.

The following situations can arise:

- When the enable pin is active (logic 1)
 1. If a logic value reaches the input clock pin of the integrated clock-gating cell, it propagates to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin does not propagate to the gated clock output pin and then into the fanout of the cell.
 2. If no logic value reaches the input clock pin of the integrated clock-gating cell, there is no logic value propagated to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin is propagated to the gated clock output pin and then into the fanout of the cell.

- When the enable pin is inactive (logic 0)
 1. If a logic value reaches the input clock pin of the integrated clock-gating cell, it does not propagate to the gated clock output pin and then into the fanout of the cell. The appropriate logic value is propagated from the gated clock output pin into the fanout of the cell. Any clock propagating up to the input clock pin is not propagated to the gated clock output pin and then into the fanout of the cell.
 2. If no logic value reaches the input clock pin of the integrated clock-gating cell, there is no logic value propagated to the gated clock output pin. The appropriate logic value is propagated from the gated clock output pin into the fanout of the cell. Any clock propagating up to the input clock pin is not propagated to the gated clock output pin and then into the fanout of the cell.
- When the enable pin is inactive (logic 0) and the cell is disabled, the appropriate logic value propagated in the fanout of the cell depends on the integrated clock-gating cell type. For example, a `latch_posedge` integrated clock-gating cell propagates a logic 0 in its fanout, whereas a `latch_negedge` integrated clock-gating cell propagates a logic 1 in its fanout.

Setting Case Analysis Values

The `set_case_analysis` command sets the logic values in the pins. It specifies that a port or pin is at a constant logic value 1 or 0, or is considered with a rising or falling transition.

When case analysis is specified as a constant value, this value is propagated through the network if the constant value is a controlling value for the traversed logic. For example, if you specify that one of the inputs of a NAND gate is a constant value 0, it is propagated to the NAND output, which is considered at a logic constant 1. This propagated constant value is then propagated to all cells driven by this signal.

Note:

When a logic value is propagated onto a net, the associated design rule checks are disable; however, you can perform maximum capacitance design rule checks on driver pins for constant nets by setting the `timing_enable_max_capacitance_set_case_analysis` variable to `true`.

To perform timing analysis for system mode by setting the case analysis for the test port to constant logic 0, enter

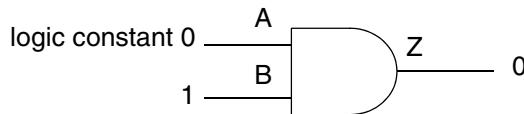
```
pt_shell> set_case_analysis 0 Test
```

You can use case analysis in combination with mode analysis (see [Mode Analysis](#)). For example, use case analysis to specify that certain internal logic is a constant value because the RAM block is in read mode:

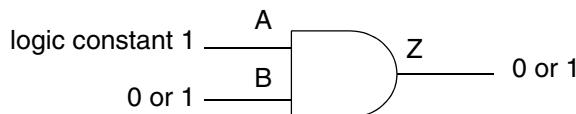
```
pt_shell> set_case_analysis 1 U1/U2/select_read
```

As shown in the following examples, if there are logic constants and case analysis on inputs of a cell, and the logic constants are set to any controlling logic value of that cell, then the logic constant is propagated from the output; otherwise case analysis is propagated.

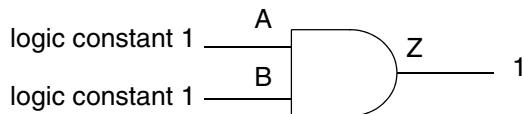
If you have an AND gate and input pin A has logic constant 0 (controlling the logic value of the AND gate), the logic constant 0 is propagated from the output pin, irrespective of the value on B. For example:



If you have the same AND gate, but pin A has logic constant 1, it is not a controlling logic value; the logic constant is not propagated to the output pin. For example:



If you have the same AND gate and both pins A and B have logic constant 1 (noncontrolling value), then a logic constant of 1 is propagated from the output of the gate. For example:



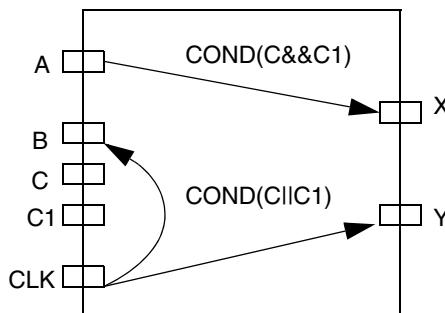
Evaluating Conditional Arcs Using Case Analysis

You can enable or disable conditional arcs from Liberty models by performing case analysis. Liberty is a modeling language that describes the static timing characteristics of large blocks for which there is no corresponding gate-level netlist. If an arc has a condition associated with it, PrimeTime evaluates the condition. If the design does not meet the condition for the arc, PrimeTime disables the conditional arc. Note that conditional testing cannot enable an arc that has been disabled by case analysis.

For example, in [Figure 10-8](#), a delay arc exists from A to X with condition COND(C&&C1). PrimeTime disables the arc when the Boolean expression (C&&C1) evaluates to false. A

setup arc also exists from CLK to B with condition $\text{COND}(\text{C1}\&\&\text{C1})$. PrimeTime disables the setup arc when the Boolean expression ($\text{C1}\&\&\text{C1}$) evaluates to false. The arc from CLK to Y is a nonconditional arc.

Figure 10-8 Conditional and Nonconditional Arcs



The following commands cause PrimeTime to disable the delay arc from A to X because it evaluates the condition to be false:

```
pt_shell> set_case_analysis 0 C
pt_shell> set_case_analysis 1 C1
```

The following command enables the delay arc from A to X and the setup arc from CLK to B:

```
pt_shell> set_case_analysis 1 {C C1}
```

For more information about specifying conditional arcs in the Liberty modeling language, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

Reporting Case Analysis Values

To report the case analysis values, use the `report_case_analysis` command. For example:

```
pt_shell> report_case_analysis
*****
Report : case_analysis
Design : false_path
*****
Pin name           Case analysis value
-----
test_port          0
U1/U2/core/WR     1
```

To show both the logic constants and case analysis set on the design, use the `report_case_analysis` command with the `-all` option. For example:

```
pt_shell> report_case_analysis -all
*****
Report : case_analysis
      -all
Design : TOP
...
*****
Pin name          Logic constant value
-----
U3/Logic0/output    0

Pin name          User case analysis value
-----
HRS                  1
ALARM                1
MINS                 1
```

In addition, you can run a more detailed case analysis by invoking PrimeTime GCA

- In the PrimeTime shell with the `report_case_analysis` command:
`report_case_analysis [-from] [-to] pin_or_port_list`
See [Example 10-1](#).
- In the PrimeTime GUI by choosing Constraints > Case Debugging; the results are displayed in a separate PrimeTime GCA window, as shown in [Figure 10-9](#).
- As a standalone tool; for detailed information, see the *PrimeTime GCA User Guide*.

Example 10-1 Case Propagation Details in pt_shell

```
pt_shell> report_case_analysis -to y/z
...
*****
Report : report_case_analysis
...
*****
Properties      Value   Pin/Port
-----
from user case  1       y/z

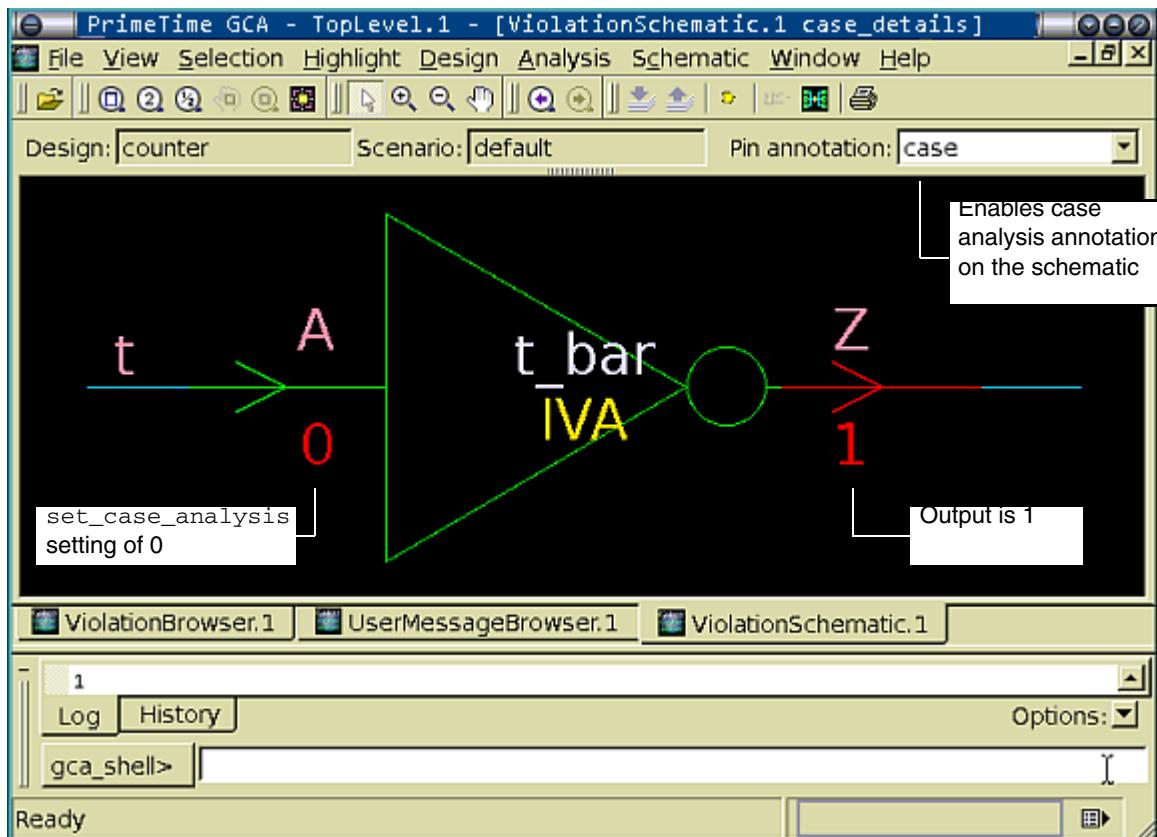
Case fanin report:
Verbose Source Trace for pin/port y/Z:
Path number: 1
Path Ref #  Value  Properties          Pin/Port
-----
1           F()=A B & C &             y/Z
2           1       y/A
3           1       y/B
4           1       y/C

Path number: 2
Path Ref #  Value  Properties          Pin/Port
-----
1           y/A
1           user case          P

Path number: 3
Path Ref #  Value  Properties          Pin/Port
-----
1           y/B
1           user case          T

Path number: 4
Path Ref #  Value  Properties          Pin/Port
-----
1           y/C
1           user case          L
```

Figure 10-9 Case Propagation Details in PrimeTime GCA



Removing Case Analysis Values

To remove case analysis values, use the `remove_case_analysis` command. The following example removes case analysis values from the `test_port` design:

```
pt_shell> remove_case_analysis test_port
```

To suppress propagating logic constants (including those set with case analysis and pins tied to logic high or low), set the `disable_case_analysis` variable to `true`. You can use this feature when you want to write scripts for Design Compiler that can be synthesized correctly.

Constant Propagation Log File

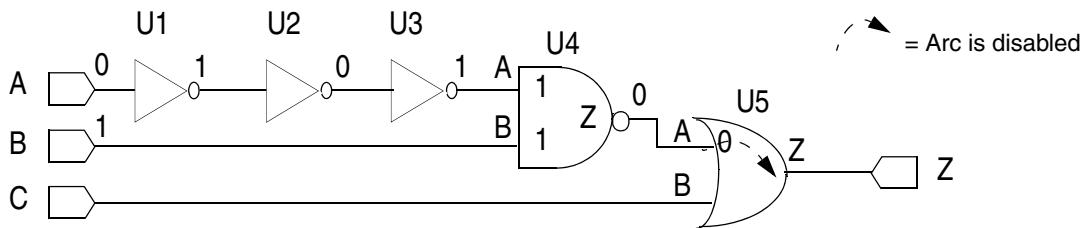
When PrimeTime performs constant propagation, it disables timing arcs at the boundary of the domain where a constant propagates. For this reason, when PrimeTime disables a timing arc, knowing the exact origin of the constant value propagated on a given pin of the design might be difficult.

To help determine the cause of a timing arc disabled because of constant propagation, you can set the `case_analysis_log_file` variable to a file name. The constant propagation process is logged to the named file. You can also view the timing arcs disabled after constant propagation with the `report_disable_timing` command.

The following example sets case analysis and reports the disabled timing arcs to the log file `my_design_cnst.log` for the circuit in [Figure 10-10](#):

```
pt_shell> set_case_analysis 0 {get_ports "A"}
pt_shell> set_case_analysis 1 {get_ports "B"}
pt_shell> set case_analysis_log_file my_design_cnst.log
```

Figure 10-10 Circuit Example



In the example, the only arc that PrimeTime needs to disable is the arc in U5 from U5/A to U5/Z. PrimeTime needs to disable this arc because U5/A is at constant value 0 and no constant is propagated to U5/Z. The constant propagation log file `my_design_cnst.log` looks like this:

```
*****
Report : case_analysis propagation
Design : my_design
Version: ...
*****

1.1 Forward propagation on NET pins (level 1)
-----
Propagating logic constant '0' from pin 'A' on net 'A':
> pin 'U1/A' is at logic '0'
Propagation of logic constant '1' from pin 'B' on net 'B':
> pin 'U4/B' is at logic '1'

1.2 Forward propagation through CELLS (level 1)
-----
Cell 'U1' (libcell 'IV') :
  input pin U1/A is at logic '0'
  > output pin 'U1/Z' is at logic '1'
```

```
2.1 Forward propagation on NET pins (level 2)
-----
    Propagating logic constant '1' from pin 'U1/Z' on net
    'w1':
        > pin 'U2/A' is at logic '1'

2.2 Forward propagation through CELLS (level 2)
-----
    Cell 'U2' (libcell 'IV') :
        input pin U2/A is at logic '1'
        > output pin 'U2/Z' is at logic '0'

3.1 Forward propagation on NET pins (level 3)
-----
    Propagating logic constant '0' from pin 'U2/Z' on net
    'w2':
        > pin 'U3/A' is at logic '0'

3.2 Forward propagation through CELLS (level 3)
-----
    Cell 'U3' (libcell 'IV') :
        input pin U3/A is at logic '0'
        > output pin 'U3/Z' is at logic '1'

4.1 Forward propagation on NET pins (level 4)
-----
    Propagating logic constant '1' from pin 'U3/Z' on net
    'w3':
        > pin 'U4/A' is at logic '1'

4.2 Forward propagation through CELLS (level 4)
-----
    Cell 'U4' (libcell 'ND2') :
        input pin U4/A is at logic '1'
        input pin U4/B is at logic '1'
        > output pin 'U4/Z' is at logic '0'

5.1 Forward propagation on NET pins (level 5)
-----
    Propagating logic constant '0' from pin 'U4/Z' on net
    'w4':
        > pin 'U5/A' is at logic '0'

5.2 Forward propagation through CELLS (level 5)
-----
6. End of Logic Constant Propagation
```

The result of the `report_disable_timing` command looks similar to the following:

Cell or Port	From	To	Flag	Reason
U5	A	Z	C	A = 0

Case Analysis and Maximum Capacitance Checking

By default, maximum capacitance checking is not performed on pins in the path of constant propagation. To check the maximum capacitance on the constant propagation path, set the `timing_enable_max_capacitance_set_case_analysis` variable to `true`.

Mode Analysis

Timing models and library cells can have defined operating modes, such as read and write modes for a RAM cell. Each mode has an associated set of timing arcs that PrimeTime analyzes while that mode is active.

To learn more about mode analysis, see

- [Mode Analysis Overview](#)
- [Setting Modes Using Case Analysis](#)
- [Setting Modes Directly on Cells](#)
- [Defining and Setting Design Modes](#)
- [Reporting Modes](#)

Mode Analysis Overview

Library cells and timing models can have operating modes defined in them, such as read and write modes for a RAM cell. Each mode has an associated set of timing arcs that PrimeTime analyzes while that mode is active. The timing arcs associated with inactive modes are not analyzed. In the absence of any mode settings, all modes are active and all timing arcs are analyzed.

There are two kinds of modes that can be set: cell modes and design modes. Cell modes are defined in a timing model or library cell, such as the read and write modes for a RAM cell. Design modes are user-defined modes that exist at the design level, such as normal and test modes.

Each design mode can be mapped to a set of cell modes in cell instances or to a set of paths. When a design mode is active, all cell modes mapped to that design mode are

activated and all paths mapped to that design modes are enabled. When a design mode is inactive (due to selection of a different design mode) then all cell modes mapped to that design mode are made inactive and all paths mapped to that design mode are set to false paths.

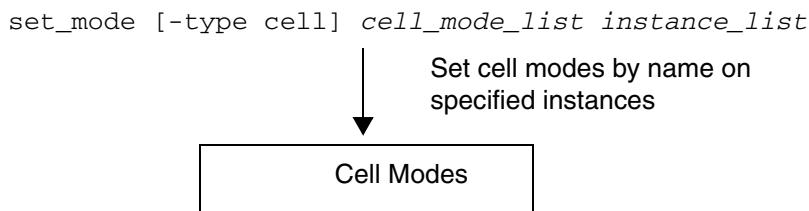
There are different methods for specifying cell operating modes in a design. In order of priority, from highest to lowest, these methods are:

1. Set cell modes directly on cell instances with the `set_mode -type cell` command. You can optionally omit `-type cell` from the command because it is the default mode type. For example, `set_mode READ {U1}` sets the READ mode on cell instance U1.
2. Set cell modes indirectly using design modes with the `set_mode -type design` command. To create design modes, use `define_design_mode_group`. Mapping can be done by specifying a list of instances or a list of paths. To map design modes into cell modes applied to instances or into paths that are enabled or disabled, use `map_design_mode`.
3. Place cells into modes using case analysis. For example, if a cell U1 has a cell mode called READ which is defined to be active when input RW is 0, setting or propagating a case analysis value of 0 on the U1/RW input activates the READ mode and deactivates all other modes for that cell. The `set_case_analysis 0 [get_pins U1/RW]` command sets a logic 0 directly on the RW pin of U1.

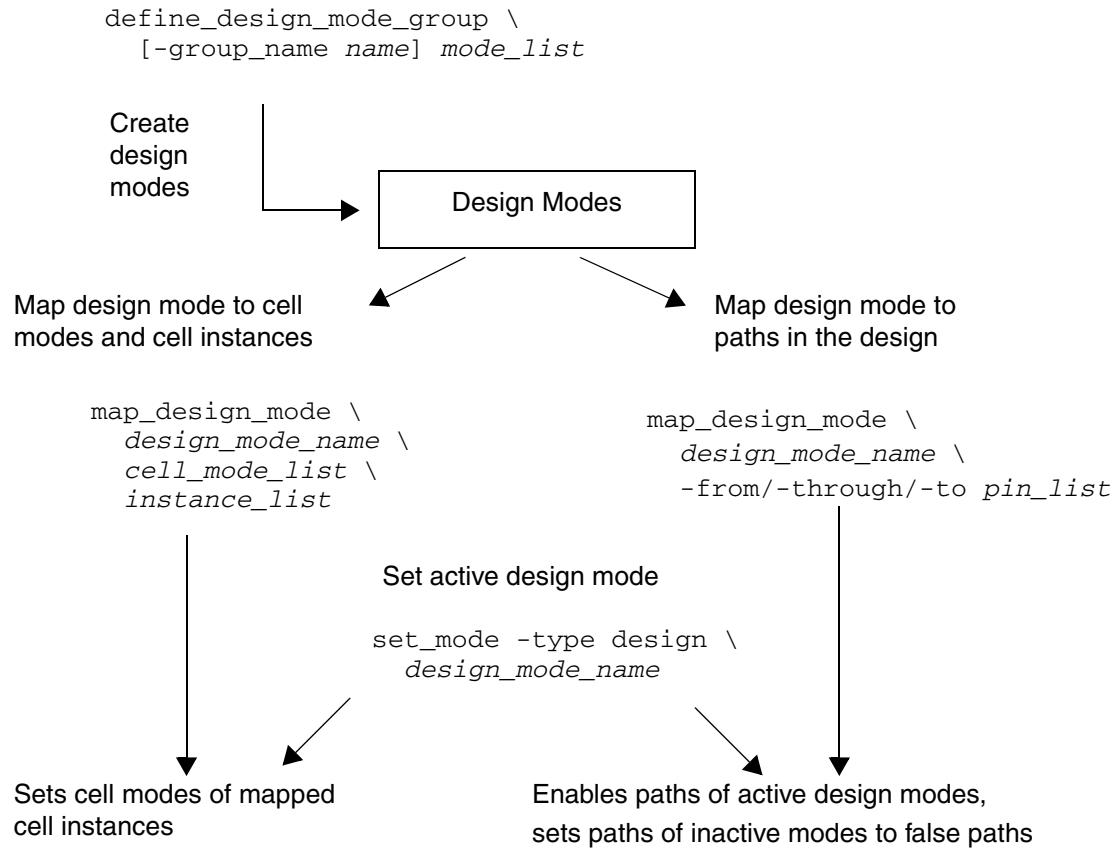
The following figures show how to select modes:

- [Figure 10-11](#) shows how to select modes in specific cell instances by name. This mode selection method has the highest priority.

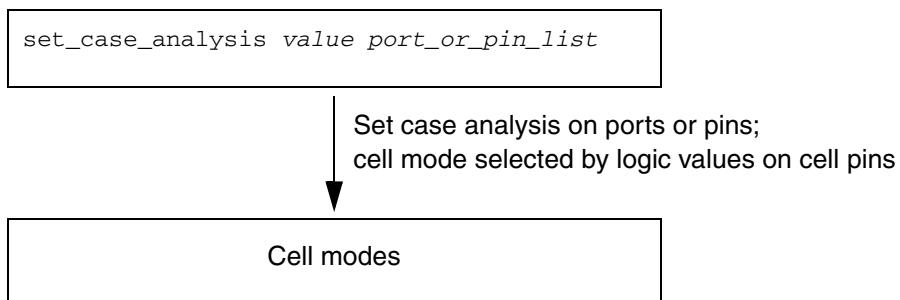
Figure 10-11 Selecting Modes Directly on Cell Instances



- [Figure 10-12](#) shows how to create, map, and select design modes. This mode selection method has medium priority.

Figure 10-12 Selecting Modes Using Design Modes

- [Figure 10-13](#) shows how to select modes by using case analysis, taking advantage of the conditional modes defined in the cell. This mode selection method has the lowest priority.

Figure 10-13 Selecting Modes Using Case Analysis

How Cell Modes Are Defined

Library cells and timing models created with the Liberty modeling language can contain operating modes. To define modes for library cells, use the mode specification features of Library Compiler. See the Library Compiler documentation for more information. To find out about the available cell modes and design modes in the current design, use the `report_mode -type cell` or `report_mode -type design` command.

Mode Groups

Modes are organized into groups. Within each group, there are two possible states: all mode enabled (the default), or one mode enabled and all other modes disabled. Often there is only one mode group.

A library cell with modes can have one or more mode groups defined. Each cell mode group has a number of cell modes. Each cell mode is mapped to a number of timing arcs in the library cell. Every instance of that library cell has these cell mode groups together with the cell modes. For example, a typical RAM block can have read, write, latched, and transparent modes. You can group the read and write modes, then group the latched and transparent modes in a different mode group. The advantage of grouping modes is that when you set a cell mode, PrimeTime makes the corresponding mutually exclusive modes inactive.

For example, specifying the RAM block for the read mode implicitly specifies that the write mode is inactive, irrespective of any setting for the transparent and latched modes. Similarly, specifying the RAM block for the latched mode implies that transparent mode is inactive, irrespective of the read and write mode setting. The two mode groups (read/write and transparent/latched) are independent.

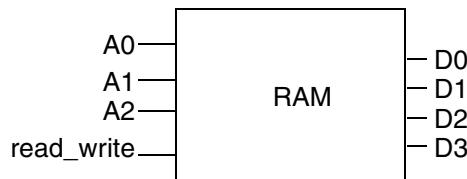
By default, all cell modes are enabled in each cell instance. Using the `set_mode -type cell` command, you can set each cell mode group to have one mode enabled and all other modes disabled. When a mode is disabled, all timing arcs in that cell mapped to that mode are disabled (not analyzed for timing).

Setting Modes Using Case Analysis

Some library cells and timing models have conditional modes defined in them. This means that a mode selection is invoked by the logical conditions at the cell inputs. For example, the read mode of a RAM cell could be invoked by a logic 0 on the read/write input pin.

When you set the controlling logic value on the input pin using case analysis (or when case analysis values are propagated to that pin), the cell is implicitly placed into the appropriate analysis mode.

In the following RAM example, the address input pins are A0, A1, and A2; and the data I/O pins are D0, D1, D2, and D3. The RAM Liberty cell can be operated in read and write modes.



The `read_write` pin of the Liberty cell controls the read/write mode of the RAM. If the RAM is modeled in the Liberty modeling language with the following mode and condition association, you can perform the read and write mode analysis by setting logic values on the `read_write` input pin:

```

cell(RAM) {
    mode_definition(read_write) {
        mode_value(read) {
            when : "read_write";
            sdf_cond : "read_write == 0";
        }
        mode_value(write) {
            when : "read_write";
            sdf_cond : "read_write == 1";
        }
    }
    ...
}

```

To enable the read mode in PrimeTime, enter

```
pt_shell> set_case_analysis 0 [get_ports read_write]
```

To enable the write mode in PrimeTime, enter

```
pt_shell> set_case_analysis 1 [get_ports read_write]
```

Setting or propagating a logic value to the `read_write` pin implicitly selects the corresponding mode and disables the other mode. Only the timing arcs associated with the active mode are used in the timing analysis.

When no modes in a mode group are selected by case analysis or other mode selection methods, all of the modes are enabled. The default mode is enabled if no mode is selected.

Setting Modes Directly on Cells

To specify the active cell modes directly for cell instances, use this command:

```
pt_shell> set_mode -type cell cell_mode_list instance_list
```

cell_mode_list is a list of modes to be made active, no more than one mode per mode group. If *cell* has only one mode group, you can list only one mode to be made active.

instance_list is a list of instances to be placed into the specified mode. You can optionally omit *-type cell* because *cell* is the default mode type. (You must specify *-type design* to set a design mode). For example, to set the U1/U2/core cell to read mode, enter

```
pt_shell> set_mode read U1/U2/core
```

This makes the read mode active and all other modes in the mode group inactive for the U1/U2/core cell.

To cancel the mode selection and make all modes active for the U1/U2/core cell, enter

```
pt_shell> reset_mode U1/U2/core
```

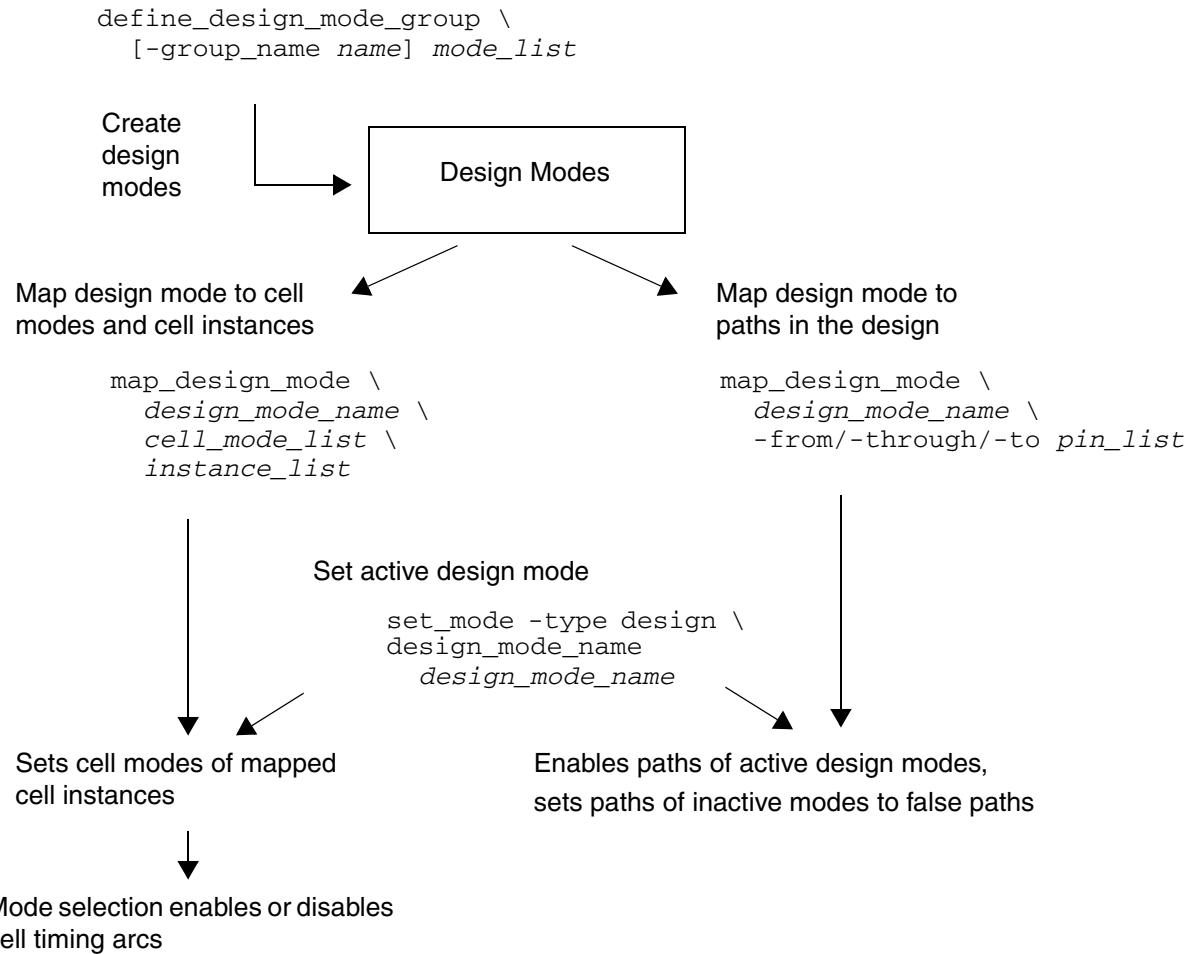
Defining and Setting Design Modes

Design modes are user-defined modes that exist at the design level, such as normal and test modes for a chip design. You create design modes in PrimeTime at the design level and map them to cell instance design modes or to paths in the design. When you set a design mode, the associated cell instances are set to the mapped cell modes, or the associated paths are enabled and paths associated with other design modes are set to false paths.

To define, map, and set a design mode,

1. Specify a set of modes for the current design using the `define_design_mode_group` command.
2. For each design mode, specify the associated cell modes or paths with the `map_design_mode` command.
3. Set the current design mode using the `set_mode -type design` command.

This process is summarized in [Figure 10-14](#).

Figure 10-14 Defining, Mapping, and Setting Design Modes

Defining Design Modes

To create a new set of design modes, use the `define_design_mode_group` command. In the command, specify the list of design modes. This is the command syntax:

```
pt_shell> define_design_mode_group [-group_name name] mode_list
```

If you are planning to have more than one design mode group, assign a unique group name for the set of design modes, and use another `define_design_mode_group` command for each design mode group. Otherwise, you can omit the `-group_name name` option and just specify the list of design modes in a single command. In that case, the design modes are assigned to a group called default. For example, to create design modes called read and write, use this command:

```
pt_shell> define_design_mode_group {read write}
```

To remove one or more design modes defined previously, use the `remove_design_mode mode_list` command.

Mapping Design Modes

After you create design modes with the `define_design_mode_group` command, you need to specify what each design mode does. This process is called mapping. You can map a design mode either to a set of cell modes and cell instances or to a set of paths.

Mapping a Design Mode to Cell Modes and Instances

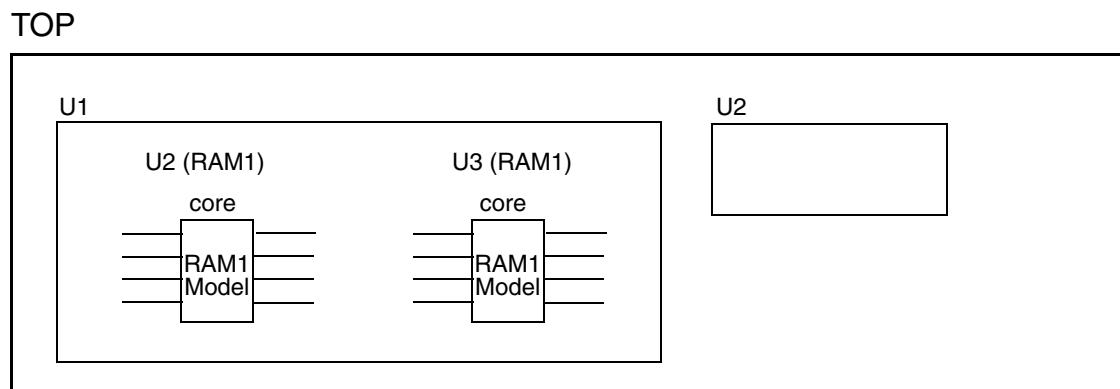
When you map a design mode to a cell mode and cell instances, activating that design mode applies the specified cell mode to the cell instances. To map a design mode in this manner, use the following command syntax:

```
map_design_mode design_mode_name \
    cell_mode_list instance_list
```

Specify the design mode name, the cell mode to be applied, and the list of instances on which to apply the cell mode. If the cell has more than one mode group, you can specify up to one cell mode per mode group in the `cell_mode_list`. Otherwise, specify just one cell mode to be applied to the cell instances. Use a separate `map_design_mode` command to map each design mode. For example, consider the design shown in [Figure 10-15](#). To map the `execute` design mode to invoke the `read_ram` cell mode in the cell instance U1/U2/core, use this command:

```
pt_shell> map_design_mode execute read_ram U1/U2/core
```

Figure 10-15 Mapping a Design Mode to a Cell Mode and Cell Instance



After this mapping, setting the design mode called “execute” makes that design mode active, causing the cell mode `read` to be applied to the U1/U2/core instance. This also deactivates the other design modes in the same design mode group as `execute`, causing those design modes to return to their default state.

Mapping a Design Mode to a Set of Paths

When you map a design mode to a set of paths, activating that design mode activates those paths. Paths that are mapped to the other design modes in the same design mode group are set to false paths. These false paths are, therefore, removed from the timing analysis.

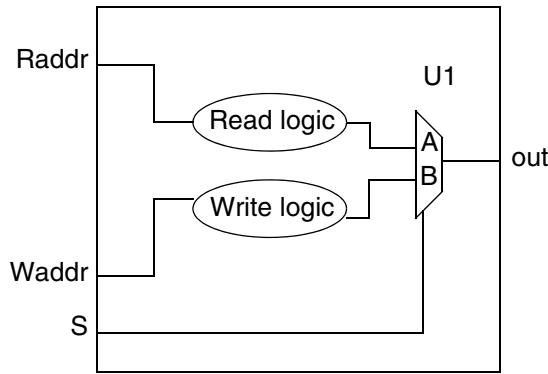
To map a design mode to a set of paths:

```
map_design_mode design_mode_name \
    [-from pin_list] [-through pin_list] [-to pin_list]
```

Specify the design mode name and the set of paths associated with that design mode. Use a separate `map_design_mode` command to map each design mode. For example, consider the design shown in [Figure 10-16](#). To map the design mode read to the set of paths through the read logic, and the design mode write to the set of paths through the write logic, enter,

```
pt_shell> map_design_mode read -from Raddr -through U1/A
pt_shell> map_design_mode write -from Waddr -through U1/B
```

Figure 10-16 Extracting Modes Using the -through Option



After this mapping, setting the design mode called `read` enables the paths through the read logic. At the same time, it deactivates the write design mode (because `read` and `write` belong to the same mode group), causing the associated paths through the write logic to become false paths. Setting the `write` design mode has the opposite effect.

Unmapping a Design Mode

If you want to cancel or change the mapping of a design mode, you can do so with the `remove_design_mode` command. To remove one or more cell instances previously mapped to a design mode, use the following syntax:

```
remove_design_mode design_mode_name cell_mode_name instance_list
```

To remove one or more paths previously mapped to a design mode, use the `remove_design_mode` command.

Removing the mapping of a design mode reverses the effects of any cell mode settings or false paths applied as a result of that mapping. To remove one or more design modes entirely, including all their mapping, use the following syntax:

```
remove_design_mode design_mode_list
```

Removing a design mode reverses the effects of any cell mode settings or false paths applied as a result of activating or deactivating that design mode.

Setting Design Modes

After you create design modes with the `define_design_mode_group` command and map all these modes with the `map_design_mode` command, you can set the active mode with the `set_mode -type design` command. Setting a design mode makes that mode active and makes the other design modes in the same design mode group inactive.

To set the current design mode, use the following syntax:

```
set_mode -type design design_mode_name
```

If there are multiple design mode groups, you can specify a list of design mode names, one per mode group. Otherwise, just specify a single design mode.

For a design mode mapped to cell modes and cell instances, the cell modes are applied to the associated instances. Other design modes in the same design mode group are made inactive, causing the associated cell instances to be returned to their default state.

For a design mode mapped to paths, the associated paths are enabled. Other design modes in the same design mode group are made inactive, causing the associated paths to become false paths that are not analyzed for timing.

To cancel the effects of a `set_mode -type design` command, use the following command syntax:

```
reset_mode -type design -group group_name
```

or

```
reset_mode -type design design_mode_name
```

If there are multiple design mode groups, you can list multiple group names or multiple design mode names, one per group. Otherwise, specify just a single group name or design mode name.

Resetting the design mode to the default state causes no mode to be selected, which means that no cell modes are applied and no paths are set to be false paths.

Reporting Modes

The `report_mode` command reports modes that have been defined or set. To report the cell modes, use the `report_mode` command with or without the `-type cell` option. To report the design modes, use `report_mode -type design`. For example:

```
pt_shell> report_mode
```

Cell	Mode (Group)	Status	Condition	Reason
Core (LIB_CORE)	read(rw)	disabled	(A==1)	cond
	oen-on (rw)	ENABLED	(A==0)	cond
	write(rw)	disabled	(B==1)	cond
Core2 (LIB_CORE)	read(rw)	disabled	(A==1)	cell
	oen-on (rw)	ENABLED	(A==0)	cell
	write(rw)	disabled	(B==1)	cell
Core3 (LIB_CORE)	read(rw)	disabled	(A==1)	design - dm1
	oen-on (rw)	ENABLED	(A==0)	design - dm1
	write(rw)	disabled	(B==1)	design - dm1
Core4 (LIB_CORE)	read(rw)	disabled	(A==1)	default
	oen-on (rw)	ENABLED	(A==0)	default
	write(rw)	disabled	(B==1)	default

The report shows the name of each cell instance with a mode setting, the possible mode settings and group names for the cell, the current status of each mode (enabled or disabled), the condition causing the current status, and the reason for the mode setting. The possible reasons are:

- cond – The mode is set conditionally by case analysis.
- cell – The mode is set directly by `set_mode -type_cell`.
- design – The mode is set by the indicated design mode.
- default – The default mode setting is in effect.

The following example shows a design mode report:

```
pt_shell> map_design_mode {read,latching} {U2/core,U1/core} read
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF3/D}
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF4/D}
pt_shell> map_design_mode {write,transparent} {U2/core,U1/core} write
pt_shell> report_mode -type design
...

```

```

-----
Design Mode Group : 'default'
  Design Mode : 'read' (is current design mode)
-----
  Cell           Mode (Group)
-----
  U2/core        latching(output_latch)
                  read(rw)
-----
  U1/core        latching(output_latch)
                  read(rw)
  From Pin
-----
INFF1/CLK
INFF1/CLK
-----
  To Pin
-----
INFF3/D
INFF4/D
-----

-----
Design Mode Group : 'default'
  Design Mode : 'write'
-----
  Cell           Mode (Config)
-----
  U2/core        transparent(output_latch)
                  write(rw)
-----
  U1/core        transparent(output_latch)
                  write(rw)
-----
```

Mode Merging for Scenario Reduction

Complex designs typically require the timing analysis of many scenarios. You can reduce the number of scenarios, which is the number of modes multiplied by the number of corners, by using the mode merging capabilities in PrimeTime. During mode merging, the tool

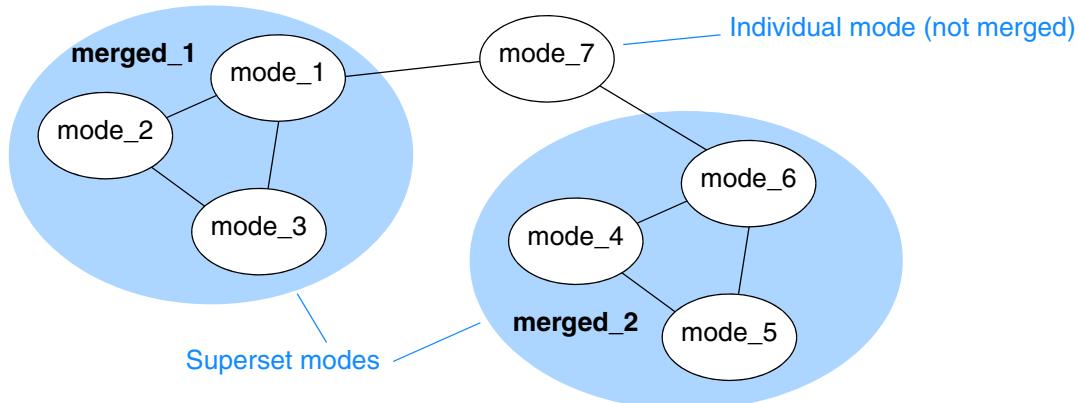
- Determines the input modes to be merged
- Merges the timing constraints from multiple input modes
- Generates a superset output mode

To run mode merging, you need the following licenses:

- PrimeTime
- PrimeTime GCA

[Figure 10-17](#) shows an example of mode merging. The tool automatically determines which input modes can be merged together and creates the output modes listed in the table.

Figure 10-17 Mode Merging Example



Input modes	Output mode
mode_1, mode_2, and mode_3	merged_1
mode_4, mode_5, and mode_6	merged_2
mode_7	mode_7 (not merged)

Running Mode Merging

To run mode merging,

1. Invoke PrimeTime in multi-scenario mode:

```
% pt_shell -multi_scenario
```

2. (Optional) Specify the number of cores, processes, or hosts for distributed mode merging:

```
set_host_options -max_cores numCores
-num_processes numProcesses
-submit_command submitCommand [host_name]
```

3. Create the scenarios for the specified modes and corners:

```
create_scenario -mode mode_name -corner corner_name
```

4. Execute mode merging:

```
create_merged_modes
```

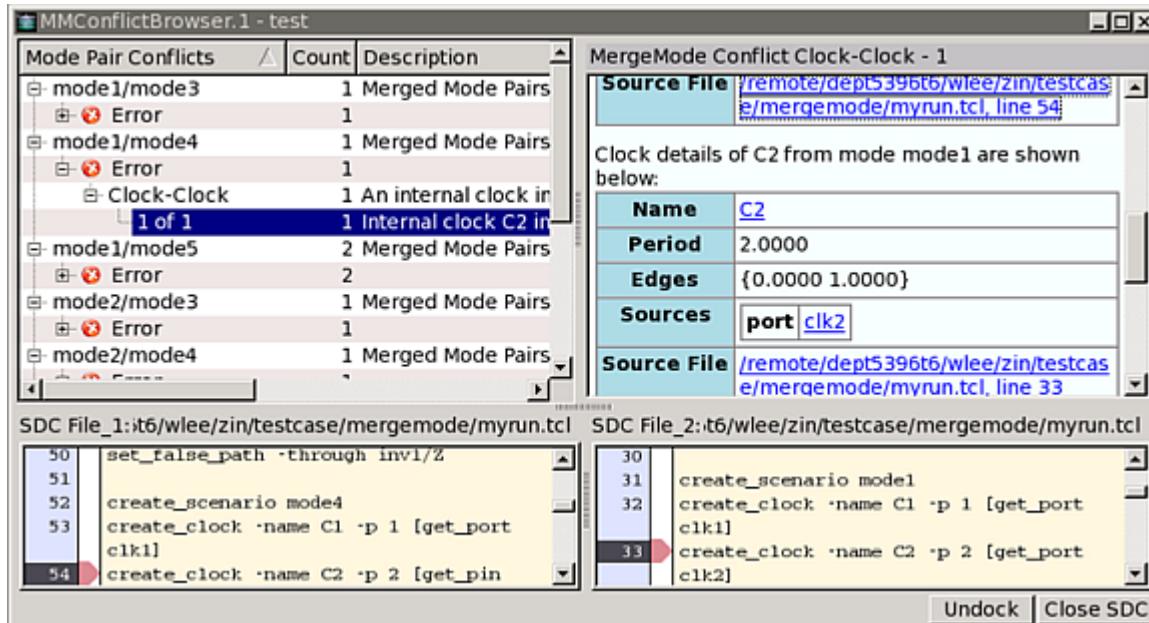
5. Check the merged constraints and mode merging analysis report in the merged_constraints directory.

Mode merging is corner-aware. The merged constraints are valid only for corners if its corresponding input modes are valid for those corners. The valid corners are specified by the `snp_s_corner_name` DMSA host variable.

Debugging Mode Merging Conflicts

To visually debug mode merging conflicts in a GUI, use the `create_merged_modes` command with the `-interactive` option. This runs mode merging with test-only analysis and does not actually merge the constraints. After the mode merging test-only analysis completes successfully, the tool opens the MergeModeConflict Browser shown in [Figure 10-18](#).

Figure 10-18 MergeModeConflict Browser



The GUI shows details, schematics, debugging help, and fixing suggestions for the following types of conflicts:

- Clock-clock
- Clock-data
- Exception
- Group path
- Reconvergence
- Value

For more information about mode merging, see [SolvNet article 036121, “PrimeTime Mode Merging Application Note.”](#)

11

Back-Annotation

Back-annotation is the process of reading delay, resistance, and capacitance values from an external file into the tool for timing analysis. Using back-annotation, you can more accurately analyze the circuit timing in the tool after each phase of physical design.

To learn about back-annotation, see

- [Overview of SDF Back-Annotation](#)
- [Reading SDF Files](#)
- [Setting Annotations From the Command Line](#)
- [Reporting Delay Back-Annotation Status](#)
- [Writing an SDF File](#)
- [Setting Lumped Parasitic Resistances and Capacitances](#)
- [Reduced and Detailed Parasitics](#)
- [Reading Parasitic Files](#)
- [Incomplete Annotated Parasitics](#)
- [Reporting Annotated Parasitics](#)
- [Removing Annotated Delays, Checks, Transitions, and Parasitics](#)
- [Back-Annotation Order of Precedence](#)

Overview of SDF Back-Annotation

For initial static timing analysis, PrimeTime estimates net delays based on a wire load model. Actual delays depend on the physical placement and routing of the cells and nets.

A floor planner or router can provide more detailed and accurate delay information, which you can provide to PrimeTime for a more accurate analysis. This process is called delay back-annotation. Back-annotated information is often provided in an SDF file.

You can read SDF back-annotated delay information in these ways:

- Read the delays and timing checks from an SDF file.
- Annotate delays, timing checks, and transition times from the command line without using the SDF format.

PrimeTime supports SDF v1.0 through 2.1 and a subset of v3.0 features. In general, it supports all SDF constructs except for the following:

- PATHPULSE, GLOBALPATHPULSE
- NETDELAY, CORRELATION
- PATHCONSTRAINT, SUM, DIFF, SKEWCONSTRAINTS

It also supports the following subset of SDF v3.0 constructs:

- RETAIN
- RECREM
- REMOVAL
- CONDELSE

If you do not have an SDF file, you can specify delays in an analyzer script file containing capacitance and resistance parasitics annotation commands.

Reading SDF Files

The `read_sdf` command reads instance-specific pin-to-pin leaf cell and net timing information from an SDF version 1.0, 1.1, 2.0, 2.1, or 3.0 file, and uses the information to annotate the current design.

Instance names in the design must match instance names in the timing file. For example, if the timing file was created from a design using VHDL naming conventions, the design you specify must use VHDL naming conventions.

After reading an SDF file, PrimeTime reports the following:

- Number of errors found while reading the SDF file (for example, pins not found in the design)
- Number of annotated delays and timing checks
- Unsupported SDF constructs found in the SDF file, with the number of occurrences of each SDF construct
- Process, temperature, and voltage values found in the SDF file
- Annotated delays and timing checks of the design (with the `report_annotation_delay` and `report_annotation_check` commands)

The following command reads from disk the SDF format file `adder.sdf`, which contains load delays included in the cell delays and uses its information to annotate the timing on the current design.

```
pt_shell> read_sdf -load_delay cell adder.sdf
```

The following commands read the timing information of instance `u1` of design `MULT16` from the disk file `mult16_u1.sdf`, and annotates the timing on the design `MY_DESIGN`. The load delay is included in the net delays.

```
pt_shell> current_design MY_DESIGN
pt_shell> read_sdf -load_delay net -path u1 mult16_u1.sdf
```

The following command reads timing information and annotates the current design with the worst timing when the timing file has different timing conditions for the same pin pair. The load delay is assumed to be included in the cell delay.

```
pt_shell> read_sdf -cond_use max boo.sdf
```

The following command reads minimum and maximum timing information and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delay, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delays, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type on_chip_variation boo.sdf
```

The following command reads minimum and maximum timing information from two separate SDF files and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delays, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delays, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type on_chip_variation \
-min_file boo_bc.sdf -max_file boo_wc.sdf
```

Annotating Timing From a Subdesign Timing File

When you specify the `-path` option, the `read_sdf` command annotates the current design with information from a timing file created from a subdesign of the current design. When you specify a subdesign, you cannot use the net delays to the ports of the subdesign to annotate the current design.

Annotating Load Delay

The load delay, also known as extra source gate delay, is the portion of the cell delay caused by the capacitive load of the driven net. Some delay calculators consider the load delay part of the net delay; other delay calculators consider the load delay part of the cell delay. By default, the `read_sdf` command assumes the load delay is included in the cell delay in the timing file being read. If your timing file includes the load delay in the net delay instead of in the cell delay, use the `-load_delay` option with the `read_sdf` command.

Annotating Conditional Delays From SDF

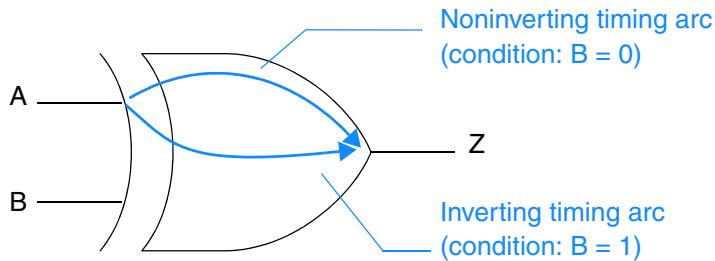
Delays and timing checks specified in an SDF file can be conditional. The SDF condition is usually an expression based on the value of some inputs of the annotated cell. The way PrimeTime annotates these conditional delays depends on whether the Synopsys library specifies conditional delays.

- If the Synopsys library contains conditional arcs, all conditional delays specified in SDF file are annotated. The condition string specified in the Synopsys library with the `sdf_cond` construct must exactly match the condition string in the SDF file.
- If the Synopsys library does not contain conditional arcs (there is no `sdf_cond` construct in the Synopsys library), the maximum or minimum delays of all conditional delays from SDF are annotated. To specify whether to annotate the minimum or maximum delays, use the `-cond_use max` or `-cond_use min` option of the `read_sdf` command.

If your library contains state-dependent delays, using a Synopsys library containing conditional arcs enables more accurate annotation from the SDF.

PrimeTime uses the condition specified in the SDF only to annotate the delays and timing checks to the appropriate timing arc specified in the Synopsys library. If you have conditional timing arcs in your SDF, and your library is defined correctly to support conditional arcs, you can use case analysis (or constant propagation) to enable the desired conditional arc values. Consider the example 2-input XOR gate in [Figure 11-1](#).

Figure 11-1 Example of State-Dependent Timing Arcs

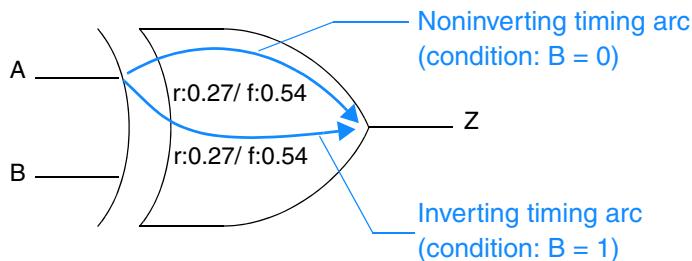


The SDF for the delay from A to Z looks like this:

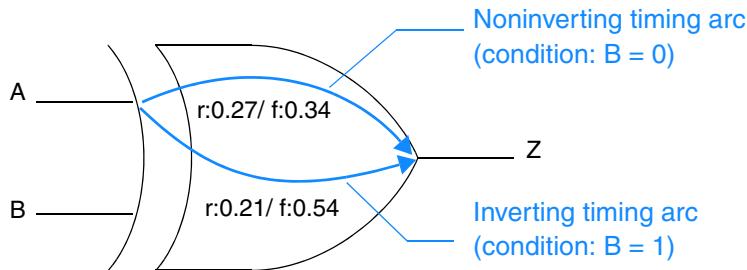
```
(INSTANCE U1)
(DELAY
  (ABSOLUTE
    (COND B (IOPATH A Z (0.21) (0.54) ) )
    (COND ~B (IOPATH A Z (0.27) (0.34) ) )
  )
)
```

If the Synopsys library contains no conditions, the annotation from SDF uses the worst-case delay for all timing arcs from A to Z. Mapping from the library the timing arc that corresponds to a given condition is not possible. In this case, the annotated delays are as shown in [Figure 11-2](#).

Figure 11-2 Annotated Delays When the Synopsys Library Contains No Conditions



If the Synopsys library contains conditions, the annotation can identify which timing arc corresponds to the SDF condition. In this case, the annotated delays are as shown in [Figure 11-3](#).

Figure 11-3 Annotated Delays When the Synopsys Library Contains Conditions**Note:**

An `IOPATH` statement in the SDF file annotates all arcs between two pins. However, even if an `IOPATH` statement follows a `COND IOPATH` statement, the `COND IOPATH` statement takes precedence over the `IOPATH` statement.

The `IOPATH` delay for the A to Z arc varies depending on whether the B input is a 1 or 0. To select the B = 0 delays, set a case analysis on the B input pin. For example, enter

```
pt_shell> set_case_analysis 0 [get_pins U1/B]
```

Note:

It is possible for a constant to propagate to a pin that selects a conditional arc. This can occur through normal logic constant propagation from a tie-high or tie-low condition, or through a `set_case_analysis` that was set on another pin that propagates a constant to the pin.

Annotating Timing Checks

When the SDF file contains the timing checks in [Table 11-1](#), they are used to annotate the current design.

Table 11-1 SDF Constructs for Timing Checks

For check	SDF construct is
Setup and hold	<code>SETUP</code> , <code>HOLD</code> , and <code>SETUPHOLD</code>
Recovery	<code>RECOVERY</code>
Removal	<code>REMOVAL</code>
Minimum pulse width	<code>WIDTH</code>
Minimum period	<code>PERIOD</code>
Maximum skew	<code>SKEW</code>

Table 11-1 SDF Constructs for Timing Checks (Continued)

For check	SDF construct is
No change	NOCHANGE

Setting Annotations From the Command Line

You can annotate delays, timing checks, and transition times from the command line to make a limited number of changes for debugging. To learn about manually set annotations, see

- [Annotating Delays](#)
- [Annotating Timing Checks](#)
- [Annotating Transition Times](#)

Annotating Delays

The `set_annotated_delay` command sets cell or net delays from the command line. To list annotated delay values, use the `report_annotated_delay` command. To set a cell delay, you specify the delay from a cell input to an output of the same cell. To set a net delay, you specify the delay from a cell output to a cell input.

To remove the annotated cell or net delay values from a design, use the `remove_annotated_delay` or `reset_design` command.

This example annotates a cell delay of 20 units between input pin A of cell instance U1/U2/U3 and output pin Z of the same cell instance. The delay value of 20 includes the load delay.

```
pt_shell> set_annotated_delay -cell -load_delay cell 20 \
-from U1/U2/U3/A -to U1/U2/U3/Z
```

This example annotates a rise net delay of 1.4 units between output pin U1/Z and input pin U2/A. The delay value for this net does not include load delay.

```
pt_shell> set_annotated_delay -net -rise 1.4 -load_delay \
cell -from U1/Z -to U2/A
```

This example annotates a rise net delay of 12.3 units between the same output pins. In this case the net delay value does include load delay.

```
pt_shell> set_annotated_delay -net -rise 12.3 -load_delay \
net -from U1/Z -to U2/A
```

This example annotates a fall cell delay of 21.2 units on the enable arc of the three-state cell instance U8.

```
pt_shell> set_annotated_delay -cell -fall -of_objects \
           [get_timing_arcs -from U8/EN -to U8/Z \
           -filter sense==enable_low] 21.2
```

Annotating Timing Checks

The `set_annotated_check` command sets the setup, hold, recovery, removal, or no-change timing check value between two pins.

To list annotated timing check values, use the `report_annotated_check` command. To remove annotated timing check values from a design, use the `remove_annotated_check` or `reset_design` command. To see the effect of `set_annotated_check` for a specific instance, use the `report_timing` command.

This example annotates a setup time of 2.1 units between clock pin CP of cell instance u1/ff12 and data pin D of the same cell instance.

```
pt_shell> set_annotated_check -setup 2.1 -from u1/ff12/CP \
           -to u1/ff12/D
```

Annotating Transition Times

The `set_annotated_transition` command sets the transition time on any pin of a design. Transition time (also known as slew) is the amount of time it takes for a signal to change from low to high or from high to low.

This example annotates a rising transition time of 0.5 units and a falling transition time of 0.7 units on input pin A of cell instance U1/U2/U3.

```
pt_shell> set_annotated_transition -rise 0.5 [get_pins U1/U2/U3/A
pt_shell> set_annotated_transition -fall 0.7 [get_pins U1/U2/U3/A
```

Generating Timing Constraints for Place and Route

To write path timing constraints in SDF versions 1.0 and 2.1 formats to a constraints file, use the `write_sdf_constraints` command. Use the constraint file to constrain layout tools to meet critical timing goals.

To write constraints for the current design, arrival totals and slacks must be available throughout the design, not just at endpoints. The optimal flow (in terms of CPU usage) to use this command within PrimeTime is as follows:

1. Store slacks at all pins by setting the `timing_save_pin_arrival_and_slack` variable to `true`.
2. Update timing by using the `update_timing` command.

Note:

If you intend to use `write_sdf_constraints` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update.

3. Write constraints for the current design by using the `write_sdf_constraints` command.

Keep the following limitations in mind:

- The tool does not generate capacitance constraint files.
- The tool does not support some Design Compiler `write_constraints` command options, including the `-cover_nets` and `-load_delay net | cell` options.

[Example 11-1](#) shows a timing constraint file in SDF version 2.1 format.

Example 11-1 Timing Constraint File in SDF Version 2.1 Format

```
(DELAYFILE
(SDFVERSION "OVI 2.1")
(DESIGN "counter")
(DATE "Mon Dec 08 08:18:28 2014")
(VENDOR "abc_10k")
(PROGRAM "Synopsys PrimeTime")
(VERSION "2014.12")
(DIVIDER /)
(VOLTAGE 5.00:5.00:5.00)
(PROCESS "NOMINAL")
(TEMPERATURE 25.00:25.00:25.00)
(TIMESCALE 1ns)
(CELL
  (CELLTYPE "counter")
  (INSTANCE)
  (TIMINGCHECK
    (PATHCONSTRAINT ffb/CP ffb/QN w/B w/Z q/A q/Z j/C j/Z
      ffd/D
      (9.100:9.100:9.100) (9.100:9.100:9.100) )
    (PATHCONSTRAINT ffb/CP ffb/QN v/A v/Z r/A r/Z g/C g/Z
      ffc/D
      (9.100:9.100:9.100) (9.100:9.100:9.100) )
    (PATHCONSTRAINT ffa/CP ffa/QN u/A u/Z s/A s/Z d/C d/Z
      ffb/D
      (9.100:9.100:9.100) (9.100:9.100:9.100) )
```

```
(PATHCONSTRAINT ffa/CP ffa/QN t/B t/Z a/C a/Z ffa/D
  (9.150:9.150:9.150) (9.150:9.150:9.150) )
)
)
```

The following example shows the syntax to write a timing constraint file in SDF v2.1 format for the most critical path in each path group in the current design:

```
pt_shell> write_sdf_constraints cstr.sdf
```

The following example shows the syntax to write a timing constraint file in SDF v1.0 format for the 10 most critical paths in each path group in the current design (named counter):

```
pt_shell> write_sdf_constraints -max_paths 10 \
-version 1.0 counter_cstr.sdf
```

Providing Constraint Coverage for the Entire Design

To constrain a design fully for placement, every net or every net arc (connection between two cells) generally requires a constraint. You can ensure that a design is fully constrained by forcing the `write_sdf_constraints` command to generate constraints for all paths. Generation of the resulting constraint file requires a large amount of CPU time and disk space.

Some place and route tools require only the worst path through any point (other data is not used). In these cases, generating the worst path through every driver-load pin pair is sufficient to provide full constraint coverage. In that case, the total number of paths for the design is less than or equal to the total number of leaf cell input pins plus the number of primary outputs (often, points are covered by multiple paths).

The `write_sdf_constraints` command's `-cover_design` option generates just enough unique paths to provide constraint coverage for the entire design. The overall runtime with the `-cover_design` option is greater, but memory and disk space requirements are significantly less.

Note:

The `-cover_design` option ensures that a constraint is placed on every driver-load pin pair, but does not fully constrain the design. The coverage is minimal; it contains less information than a constraint file with all paths enumerated. Theoretically, a place and route tool can meet all the given constraints and still have timing violations.

The `-cover_design` option is most useful for large designs in which generating data for all paths is not reasonable, or when the targeted place and route tool uses only the worst path through any point. A point can be a net or an edge, where an edge is any connection between two leaf-level cells.

For large designs in which generating data for all paths is not reasonable, using the `-cover_design` option might be the only way to ensure that every net is constrained at least one time. When the targeted place and route tool uses only the worst path through any point, constraints other than the worst are ignored by the place and route tool.

To write a timing constraint file in SDF v2.1 format with just enough paths to cover the worst path through every driver-load pin pair, enter

```
pt_shell> write_sdf_constraints -cover_design \
    counter_cstr.sdf
```

Reporting Delay Back-Annotation Status

Because SDF files are usually large (about 100 MB or larger) and contain many delays, it is a good idea to verify that all nets are back-annotated with delays (and timing checks, where applicable).

Reporting Annotated or Nonannotated Delays

You can check and identify nets that have not been back-annotated with delays in SDF. PrimeTime checks and reports on cell delays and net delays independently. When reporting net delays, PrimeTime considers three types of nets:

- Nets connected to primary input ports
- Nets connected to primary output ports
- Internal nets that are not connected to primary ports

This distinction is important because according to the SDF standard, only internal nets are annotated in SDF.

The `report_annotated_delay` command reports the number of annotated and nonannotated delay arcs. To produce a report of annotated delays, use the `report_annotated_delay` command.

PrimeTime displays a report similar to the following:

Delay type	Total	Annotated	NOT Annotated
cell arcs	16	14	2
cell arcs (unconnected)	5	5	0
internal net arcs	3	3	0
net arcs from primary inputs	11	11	0
net arcs to primary outputs	4	4	0
	39	37	2

Reporting Annotated or Nonannotated Timing Checks

You can create a report showing the number of annotated timing checks of all cell timing arcs within your design.

To produce an annotated timing check report, use the `report_annotated_check` command.

PrimeTime displays a report similar to the following:

```
*****
report: annotated_check
Design: my_design
*****
```

	Total	Annotated	NOT Annotated
cell setup arcs	2368	2368	0
cell hold arcs	2368	2368	0
cell recovery arcs	676	676	0
cell removal arcs	0	0	0
cell min pulse width arc	135	135	0
cell min period arcs	822	822	0
cell max skew arcs	716	716	0
cell nochange arcs	0	0	0
	7085	7085	0

Faster Timing Updates in SDF Flows

For each point in a path, PrimeTime calculates slew from the input slew and capacitance and propagates the calculated slew forward from that point in the path. The `timing_use_zero_slew_for_annotated_arcs` variable enables you to use zero slew for arcs annotated with SDF delays, thereby reducing runtime when analyzing designs with all or almost all arcs annotated with SDF.

By default, this variable is set to `auto`. In a design that uses SDF-annotated delays on all arcs or almost all arcs, such as 95% or more, you can forego the higher accuracy of slew calculation in favor of faster runtime with the `auto` setting. In this case, for each arc that is fully annotated by either the `read_sdf` or `set_annotated_delay` command, PrimeTime skips the delay and slew calculation and sets a slew of zero on the load pin of the annotated arc. As a result, timing updates can be completed in significantly less time.

For any arcs that are not annotated, PrimeTime estimates the delay and output slew using the best available input slew. For a block of arcs that are not annotated, PrimeTime propagates the worst slew throughout the block.

When you use this feature, it is recommended that you disable prelayout slew scaling by setting the `timing_prelayout_scaling` variable to `false`.

Writing an SDF File

You might want to write out the back-annotated delay information to use for gate-level simulation or another purpose. To write the delay information in SDF version 1.0, 2.1, or 3.0 format, use the `write_sdf` command. The default output format is version 2.1. For example, to write an SDF file, enter

```
pt_shell> write_sdf -version 2.1 -input_port_nets mydesign.sdf
```

Note:

If you use a utility other than the `write_sdf` command to write out the SDF file, ensure that the annotations are explicitly specified where the SDF version permits.

To learn about the SDF written by PrimeTime, see

- [SDF Constructs](#)
 - [SDF Delay Triplets](#)
 - [SDF Conditions and Edge Identifiers](#)
 - [Reducing SDF for Clock Mesh/Spine Networks](#)
 - [Writing VITAL Compliant SDF Files](#)
-

SDF Constructs

The SDF written by PrimeTime uses the following SDF constructs:

- DELAYFILE, SDFVERSION, DESIGN, DATE, VENDOR, PROGRAM, VERSION, DIVIDER, VOLTAGE, PROCESS, TEMPERATURE, TIMESCALE
- CELL, CELLCYPE, INSTANCE
- ABSOLUTE, COND, CONDELSCE, COSETUP, DELAY, HOLD, INTERCONNECT, IOPATH, NOCHANGE, PERIOD, RECOVERY, RECREM, RETAIN, SETUP, SETUPHOLD, SKEW, TIMINGCHECK, WIDTH

Note:

The following constructs are supported in SDF version 3.0 only: CONDELSCE, RETAIN, RECREM, REMOVAL.

- Posedge and negedge identifiers

The SDF written by the `write_sdf` command does not use the following SDF constructs: INCREMENT, CORRELATION, PATHPULSE, GLOBALPATHPULSE, PORT, DEVICE, SUM, DIFF, SKEWCONSTRAINT, PATHCONSTRAINT. However, the `write_sdf_constraints` command supports the PATHCONSTRAINT construct.

SDF Delay Triplets

The SDF delay triplet values depend on whether your design uses a single operating condition or minimum and maximum operating conditions. For a single operating condition, the SDF delay triplet has three identical values, for example: (1.0:1.0:1.0).

For minimum and maximum operating conditions, the SDF triplet contains only two delays for the minimum operating condition and the maximum operating condition, respectively: (1.0::2.0). The typical delay of the SDF triplet is not used. The SDF delays written by PrimeTime specify the following transitions: 0 to 1, 1 to 0, 0 to Z, Z to 1, 1 to Z, and Z to 0.

SDF Conditions and Edge Identifiers

PrimeTime takes advantage of the edge identifiers as well as conditions if edge identifiers are specified in the library with `sdf_cond`. PrimeTime uses edge identifiers for timing checks and cell delays.

PrimeTime writes an edge identifier (`POSEDGE` or `NEGEDGE`) for a combinational cell delay arc when the positive edge delay differs from the negative edge delay, and the input net transition differs between rise and fall on the input pin of the delay arc. As a result, two timing arc `IOPATH` delays can be generated for a given timing arc. For example,

```
(CELL
  (CELLTYPE "XOR")
  (INSTANCE U1)
  (DELAY
    (ABSOLUTE
      (IOPATH (posedge A) Z (0.936:0.936:0.936)
(1.125:1.125:1.125))
      (IOPATH (negedge A) Z (1.936:1.936:1.936)
(2.125:2.125:2.125))
      (IOPATH (posedge B) Z (0.936:0.936:0.936)
(1.125:1.125:1.125))
      (IOPATH (negedge B) Z (1.936:1.936:1.936)
(2.125:2.125:2.125))
    )
  )
)
```

This is common for library cells such as multiplexers and exclusive OR cells. Other SDF writers might only generate one set of delay triplets for the positive and negative edges. PrimeTime writes both for the highest accuracy. However, some logic simulators do not support edge identifiers on combinational and sequential timing arcs and expect to see only one timing arc. To write an SDF that is compatible with these simulators, use the `-no_edge` option with the `write_sdf` command. For example,

```
pt_shell> write_sdf -no_edge mydesign.sdf
```

With the `-no_edge` option, PrimeTime generates only one timing arc, with the worst-case delay triplets for the positive and negative transitions. Note that the simulation timing delays might be pessimistic as a result of using the `-no_edge` option.

Reducing SDF for Clock Mesh/Spine Networks

A design with a large clock mesh or spine network can produce an unreasonably large SDF file because of the extremely large number of nets in the clock network. In these cases, you can choose to have PrimeTime reduce the SDF file by combining SDF values that differ by a negligible amount, and using the SDF 3.0 PORT construct to represent the combined nets.

The SDF reduction features operate under the control of four variables:

- `sdf_enable_port_construct` – Boolean variable, when set to `true`, enables the PORT construct to be used for writing SDF. The default setting is `false`.
- `sdf_enable_port_construct_threshold` – Floating-point value that specifies the absolute delay difference, in picoseconds, below which the PORT construct is used. The default setting is 1 ps.
- `sdf_align_multi_drive_cell_arcs` – Boolean variable, when set to `true`, causes PrimeTime to unify the small differences in driver cell outputs to networks that constitute a mesh or spine, which can cause simulation failure. The default setting is `false`.
- `sdf_align_multi_drive_cell_arcs_threshold` – Floating-point value that specifies the absolute delay difference, in picoseconds, below which multidrive arcs are aligned. The default setting is 1 ps.

See Also

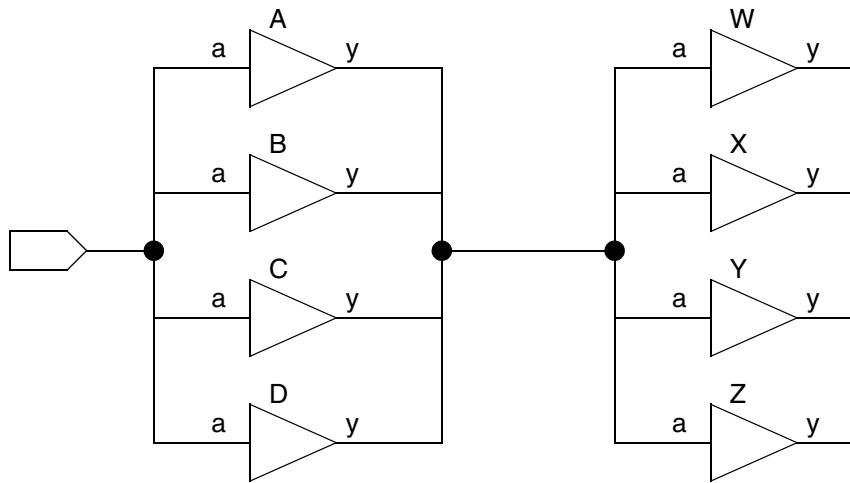
- [Fast Multidrive Delay Analysis](#)
- [Parallel Driver Reduction](#)

PORT Construct

The `sdf_enable_port_construct` variable determines whether the PORT statement is used to replace multiple INTERCONNECT statements. The PORT statement is used in all parallel nets (within specified threshold) except those parallel nets which are driven by three-state buffers. If the PORT statement is used, the

`sdf_enable_port_construct_threshold` variable determines the maximum allowable absolute difference in delay arc values for interconnections to be combined. For example, consider the clock network shown in [Figure 11-4](#).

Figure 11-4 Parallel Buffers Driving Parallel Buffers



If using the PORT construct is disabled (the default), the `write_sdf` command writes out this clock network using SDF syntax similar to the following:

```
(INSTANCE top)
(DELAY
  (ABSOLUTE
    (INTERCONNECT A/y W/a (0.01 ::0.03))
    (INTERCONNECT A/y X/a (0.02 ::0.04))
    (INTERCONNECT A/y Y/a (0.01 ::0.04))
    (INTERCONNECT A/y Z/a (0.02 ::0.03))
    (INTERCONNECT B/y W/a (0.01 ::0.03))
    (INTERCONNECT B/y X/a (0.03 ::0.05))
    ...
    (INTERCONNECT D/y Y/a (0.02 ::0.05))
    (INTERCONNECT D/y Z/a (0.01 ::0.03)))
  )
)
```

There are 16 interconnection combinations listed.

If the PORT construct is enabled, and if the variation in delay values is within the specified threshold, the `write_sdf` command reduces the SDF as follows:

```
(INSTANCE top)
(DELAY
  (ABSOLUTE
    (PORT W/a (0.02 ::0.04))
    (PORT X/a (0.03 ::0.05))
    (PORT Y/a (0.02 ::0.06))
    (PORT Z/a (0.03 ::0.07)))
  )
)
```

Four PORT statements replace 16 INTERCONNECT statements.

Normalizing Multidriven Arcs for Simulation

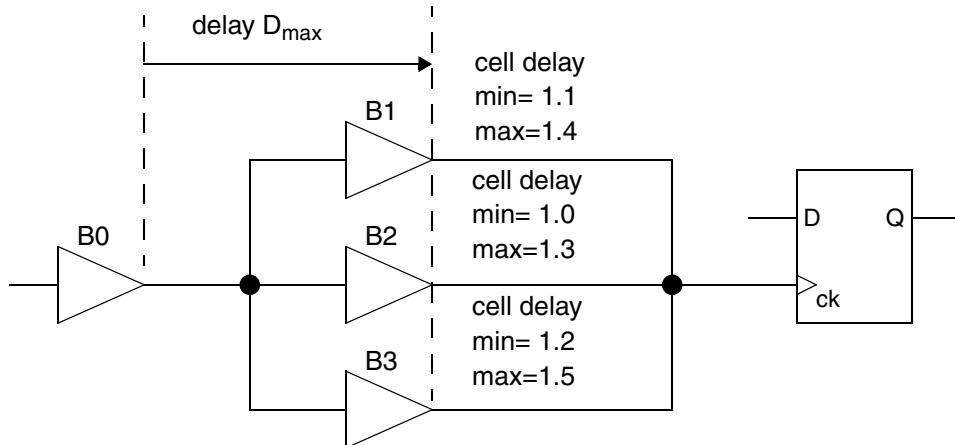
When the `sdf_align_multi_drive_cell_arcs` and `sdf_enable_port_construct` variables are enabled, PrimeTime aligns similar delay values of cell timing arcs where multiple drivers drive a common net.

Note:

If the common net is driven by three-state buffers, PrimeTime does not align cell delays.

The `sdf_align_multi_drive_cell_arcs_threshold` variable then determines the maximum allowable absolute difference in delay arc values for normalizing to occur. Normalizing means using a single worst-case delay arc value to represent multiple drivers. This can prevent errors from occurring when the SDF file is used for circuit simulation. For example, consider the cell delays in the parallel driver network shown in [Figure 11-5](#).

Figure 11-5 Cell Delays in a Parallel Driver Network



PrimeTime aligns the cell arcs if the `sdf_align_multi_drive_cell_arcs` variable is set to true, the delay arc values are within the `sdf_align_multi_drive_cell_arcs_threshold` variable setting, the cells in the clock network are combinational (not sequential), and each cell has one input and one output.

The normalizing process adjusts both the net delays and cell delays in the clock network as needed to ensure that the complete path delays are the same. Net delays are adjusted only if the `sdf_enable_port_construct` variable is set to true and the delays are within the threshold variable setting, in which case the PORT statement is used to represent the net arcs connected to the ck load pin in the example. In the case of the nets connecting B_0 to cells B_1 through B_3 , no changes are made to the delay values because each of the three cells has only one fanin net.

In the case of the cell delays, the worst delay from B0 to the output pin of cells B1 through B3 is calculated to be Dmax. The cell arcs of B1 through B3 are adjusted to make the delay of each path from B0 to the output pins of the cells to be equal to Dmax. In this example, the cells are given the same delays because the net arcs from B0 to the cells B1 through B3 all have the same delays. Taking the worst delay means using the minimum of min delays and maximum of max delays, the most pessimistic values. In the case of cells with multiple arcs, the worst cell arc is used to represent the cell delay.

To generate an SDF file for simulation, set the `sdf_align_multi_drive_cell_arcs` variable to `true`. Avoid reading the generated SDF back into PrimeTime for timing analysis, as the data is pessimistic.

If multidrive normalizing is disabled, the `write_sdf` command writes the following description of the example network.

Net delays:

```
(INTERCONNECT B1/z FF1/ck (0.01 ::0.04))
(INTERCONNECT B2/z FF1/ck (0.03 ::0.05))
(INTERCONNECT B3/z FF1/ck (0.02 ::0.04))
```

Cell delays:

```
(CELLTYPE "buf")
(INSTANCE B1)
(DELAY
  (ABSOLUTE
    (DEVICE (1::8) (4::7))
  )
)
(CELLTYPE "buf")
(INSTANCE B2)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (3::11))
  )
)
(CELLTYPE "buf")
(INSTANCE B3)
(DELAY
  (ABSOLUTE
    (DEVICE (1::5) (5::8))
  )
)
```

If multidrive normalizing is enabled, and if the delay differences are under the specified threshold, the `write_sdf` command writes the following description of the example network.

Net delays:

```
(PORT FF1/ck (0.03 ::0.05))
```

Cell delays:

```
(CELLTYPE "buf" )
(INSTANCE B1)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11)))
  )
)
(CELLTYPE "buf" )
(INSTANCE B2)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11)))
  )
)
(CELLTYPE "buf" )
(INSTANCE B3)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11)))
  )
```

Writing VITAL Compliant SDF Files

To write out SDF that is VITAL compliant, use this command:

```
pt_shell> write_sdf -no_edge_merging -exclude {"no_condense"} \
file.sdf
```

If your simulator cannot handle negative delays, use the `-no_negative_values` option with the `timing_checks`, `cell_delays`, or `net_delays` values. To zero out all negative timing check values—such as setup, hold, recovery or removal SDF statements, use the `-no_negative_values` `timing_checks` option. To zero out all negative cell delay values—such as IOPATH and port SDF statements, use the `-no_negative_values` `cell_delays` option. To zero out all negative net delays—such as interconnect SDF statements, use the `-no_negative_values` `net_delays` option. For example, to zero out timing checks and net delays, use this command:

```
pt_shell> write_sdf -no_edge_merging -no_negative_values \
{timing_checks net_delays} -exclude {"no_condense"} file.sdf
```

Writing a Mapped SDF File

The Standard Delay Format (SDF) mapping feature of PrimeTime lets you specify the output format of the SDF file. You start by creating an SDF map file, which specifies the syntax and timing arcs written to the SDF file for cells in the library. For example, if you want to change the SDF output for a flip-flop in the library, you define a map for that cell. You apply the mapping by using the `-map` option of the `write_sdf` command.

To learn about the SDF mapping feature, see

- [Specifying Timing Labels in the Library](#)
- [Specifying the min_pulse_width Constraint](#)
- [Using SDF Mapping](#)
- [Supported SDF Mapping Functions](#)
- [SDF Mapping Assumptions](#)
- [Labeling Bus Arcs](#)

Specifying Timing Labels in the Library

To reference delays of timing arcs in the SDF mapping mechanism, you need to specify a label to the timing arcs in the library. The `timing_label` attribute is written in the timing group of the library file to label a timing arc. This labels arcs for supplying values for the following SDF constructs: IOPATH, SETUP, HOLD, SETUPHOLD, RECOVERY, REMOVAL, RECREM, NOCHANGE, and WIDTH.

The following example shows how to label a timing arc A_Z:

```
cell(IV_I) {
    area : 1;
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A'";
        timing() {
            related_pin : "A";
            timing_label : "A_Z";
            rise_propagation(onebyone) {
                values("0.380000");
            }
            rise_transition(tran) {
                values("0.03, 0.04, 0.05");
            }
            fall_propagation(prop) {
                values("0.15, 0.17, 0.20");
            }
            fall_transition(tran) {
                values("0.02, 0.04, 0.06");
            }
        }
    }
}
```

Specifying the min_pulse_width Constraint

To specify the minimum pulse width arcs, use one of the following methods:

- Method 1 – Specify a `min_pulse_width` group within the pin group (in the library file).
 1. Place the `timing_label_mpw_high` attribute in the `min_pulse_width` group to reference the `constraint_high` attribute.
 2. Place the `timing_label_mpw_low` attribute in the `min_pulse_width` group to reference the `constraint_low` attribute.
- Method 2 – Specify the constraints in the library by using the `min_pulse_width_high` and `min_pulse_width_low` attributes in the pin group.
 1. Place the `timing_label_mpw_high` attribute in the pin group to reference the `min_pulse_width_high` attribute.
 2. Place the `timing_label_mpw_low` attribute in the pin group to reference the `min_pulse_width_low` attribute.

Accessing the min_pulse_width Constraint

To access the `min_pulse_width` values, use these functions:

- `min_rise_delay(label_high)` – Returns the value of the `min_pulse_width_high` attribute.
- `max_rise_delay(label_high)` – Same as `min_rise_delay(label_high)`.
- `min_fall_delay(label_low)` – Returns the value of the `min_pulse_width_low` attribute.
- `max_fall_delay(label_low)` – Same as `min_fall_delay(label_low)`.

For information about functions that support SDF mapping, see [Table 11-3](#).

Specifying the min_period Constraint on a Pin

Specify the `min_period` constraint on a pin as an attribute in the pin section. Use the `min_period(pin_name)` function to take the pin name as argument and to return the value of the `min_period` constraint.

Using SDF Mapping

To specify the SDF mapping file when writing the SDF file for a design, use this command:

```
write_sdf [-context verilog | vhdl] [-map map_file] output_sdf_file
```

Note:

Only names that are specified using the `bus()` function are affected. For more information, see [Table 11-3](#).

For example:

```
pt_shell> write_sdf -context verilog -map example.map example.sdf
Output
File: example.sdf
(DELAYFILE
 (SDFVERSION "OVI V2.1" )
 (DESIGN      "ADDER"  )
 (DATE        "Mon Jun 09 08:18:28 2014"  )
 (VENDOR      "nyaa"   )
 (PROGRAM     " "      )
 (VERSION     "3.24"   )
 (DIVIDER    /)
 (VOLTAGE    0:0:0)
 (PROCESS    " ")
 (TEMPERATURE 0:0:0 )
 (TIMESCALE   1 ns)
```

```
(CELL
  (CELLTYPE "ADDER")
  (INSTANCE )
  (DELAY
    (ABSOLUTE
      (INTERCONNECT ...))
  )
)
(CELL
  (CELLTYPE "EXMP")
  (INSTANCE U15)
  (DELAY
    (ABSOLUTE
      (IOPATH A Z (1.8::4.3) (1.8::4.0))
      (IOPATH IN[0] Z (3.0::1.2) (3.0::1.0)))
  )
)
(CELL
  (CELLTYPE "FD02")
  (INSTANCE B1/U12/DFF)
  (DELAY
    (ABSOLUTE
      (IOPATH (posedge CP) Q(1.2::1.2) (1.5::1.5))
      (IOPATH (posedge CP) QN(1.2::1.2) (1.5::1.5))
      (IOPATH (negedge CD) Q (:) (2.1::2.4))
      (IOPATH (negedge CD) QN (2.1::2.4) (:)))
    )
    (TIMINGCHECK
      (SETUP D (posedge CP)) (2.1::2.5))
      (HOLD D (posedge CP)) (1.1::1.4))
      (WIDTH (negedge CP) (5.0))
      (RECOVERY (posedge CD) (posedge CP)) (2.5))
      (PERIOD (posedge CP) (5.0)))
    )
  )
)
)
```

When SDF for cells are not present in the SDF mapping file, they are written with the default `write_sdf` format.

SDF Mapping Notation

The notation used in presenting the syntax of the SDF mapping language follows. In the following list, `item` is a symbol for a syntax construct item.

- `item?` – Item is optional in the definition (it can appear one time or not at all).
- `item*` – Item can appear zero or any number of times.

- **item+** – Item can appear one or more times (but cannot be omitted).
- **KEYWORD** – Keywords are shown in uppercase bold for easy identification but are case-insensitive.
- **VARIABLE** – Is a symbol for a variable. Variable symbols are shown in uppercase for easy identification, but are case-insensitive. Some variables are defined as one of a number of discrete choices; other variables represent user data such as names and numbers.

SDF Mapping Comments

When adding comment lines in the SDF mapping file, use this format:

```
[white_space]* [#] [any char except new-line char]*
```

If the line has a pound sign (#) as the first nonwhitespace character, it is treated as a comment line and is ignored.

Quoted strings can contain new-line characters in them, and can span multiple lines. Comments cannot be embedded inside a quoted string. For definitions of SDF mapping functions, see [Table 11-3](#).

SDF Mapping Variables

[Table 11-2](#) lists the variables used in the SDF mapping language.

Table 11-2 SDF Mapping Variables

Variable	Description
SDFMAP_DNUMBER	A nonnegative integer number, for example, +12, 23, 0
SDFMAP_FUNCTION	A string that denotes the function name. Supported functions are given in Table 11-3 .
SDFMAP_IDENTIFIER	The name of an object. It is case-sensitive. The following characters are allowed: alphanumeric characters, the underscore character (_), and the escape character (\). If you want to use a nonalphanumeric character as a part of an identifier, you must prefix it with the escape character. White spaces are not allowed with the hierarchy divider character (/).
SDFMAP_MAP	A positive integer value preceded with the dollar sign (\$), for example, \$10, \$3, \$98. Used to specify a placeholder for a value in the mapping file.
SDFMAP_NUMBER	A nonnegative (zero or positive) real number, for example, 0, 1, 3.4, .7, 0.5, 2.4e-2, 3.4e2

Table 11-2 SDF Mapping Variables (Continued)

Variable	Description
SDFMAP_QSTRING	A string of any legal SDF characters and spaces, including tabs and new-line characters, optionally enclosed by double quotation marks.
SDFMAP_RNUMBER	A positive, zero, or negative real number, for example, 0, 1, 0.0, -3.4, .7, -0.3, 2.4e-2, 3.4e4
SDFMAP_TSVALUE	A real number followed by a unit.

Supported SDF Mapping Functions

[Table 11-3](#) lists the functions supported when writing mapped SDF files.

Table 11-3 SDF Mapping Functions

Function	Return value	Comment
in(<i>float</i>)	Float	Natural logarithm.
exp(<i>float</i>)	Float	Exponentiation.
sqrt(<i>float</i>)	Float	Square root.
max(<i>float, float</i>)	Float	Two-argument maximum function.
min(<i>float, float</i>)	Float	Two-argument minimum function.
bus(<i>string</i>)	String	Returns a transformed string by replacing the bus delimiter characters in the string with the appropriate bus delimiter characters as specified in the write_sdf command. The -context verilog option causes the string to be transformed to the %s[%d] format. The -context vhdl option causes the string to be transformed to the %s(%d) format.
pin(<i>string</i>)	String	Returns the string argument passed to it.
max_fall_delay(<i>label</i>)	Float	Maximum fall delay value associated with that timing label.

Table 11-3 SDF Mapping Functions (Continued)

Function	Return value	Comment
max_fall_delay_bus (<i>label, from_pin, to_pin</i>)	Float	Maximum fall delay value to be used when the timing label is associated with a bus arc.
max_fall_retain_delay (<i>label, from_pin, to_pin</i>)	Float	Maximum fall delay value to be used when the timing label is associated with a retain arc.
max_rise_delay(<i>label</i>)	Float	Maximum rise delay value associated with that timing label.
max_rise_delay_bus (<i>label, from_pin, to_pin</i>)	Float	Maximum rise delay value is used when the timing label is associated with a bus arc.
max_rise_retain_delay (<i>label, from_pin, to_pin</i>)	Float	Maximum rise delay value to be used when the timing label is associated with a retain arc.
min_fall_delay(<i>label</i>)	Float	Minimum fall delay value associated with that timing label.
min_fall_delay_bus (<i>label, from_pin, to_pin</i>)	Float	Minimum fall delay function to be used when the timing label is associated with a bus arc.
min_fall_retain_delay (<i>label, from_pin, to_pin</i>)	Float	Minimum fall delay value to be used when the timing label is associated with a retain arc.
min_period(<i>pin</i>)	Float	Returns the minimum period value associated with the pin name “ <i>pin</i> ” of the current cell.
min_rise_delay(<i>label</i>)	Float	Minimum rise delay value associated with that timing label.
min_rise_delay_bus (<i>label, from_pin, to_pin</i>)	Float	Minimum rise delay value to be used when the timing label is associated with a bus arc.
min_rise_retain_delay (<i>label, from_pin, to_pin</i>)	Float	Minimum rise delay value to be used when the timing label is associated with a retain arc.

SDF Mapping File Syntax

Example 11-2 shows the complete syntax of a mapped SDF file.

Example 11-2 Syntax of Mapped SDF File

```

mapping_file      ::= map_header cell_map+
map_header       ::= sdf_version sdf_map_name? hierarchy_divider?
                           bus_delimiter?

sdf_map_name     ::= $SDF_MAP_NAME SDFMAP_QSTRING
sdf_version      ::= $SDF_VERSION SDFMAP_QSTRING
hierarchy_divider ::= $SDF_HIERARCHY_DIVIDER SDFMAP_HCHAR
bus_delimiter    ::= $SDF_BUSBIT SDFMAP_QSTRING

cell_map          ::= $SDF_CELL cell_name format_string var_map
                           $SDF_CELL_END
cell_name         ::= SDFMAP_QSTRING
format_string     ::= SDFMAP_QSTRING

var_map           ::= var_map_line+
var_map_line      ::= SDFMAP_MAP expression

expression         ::= expression binary_operator expression
                           |= unary_operator expression
                           |= expression ? expression : expression
                           |= ( expression )
                           |= function_call
                           |= SDFMAP_NUMBER

function_call     ::= SDFMAP_FUNCTION ( func_args? )
func_args          ::= func_arg
                           |= func_args , func_args
func_arg           ::= SDFMAP_IDENTIFIER
                           |= expression

binary_operator   ::= +
                           |= -
                           |= *
                           |= /
                           |= &
                           |= |
                           |= >
                           |= <
                           |= =
binary_operator   ::= !
                           |= -

```

SDF Mapping Assumptions

The SDF mapping file reader assumes that the SDF format string read from the SDF mapping file has valid SDF syntax. For instance, it assumes that when the proper substitutions are made for the placeholders, the resulting SDF is syntactically (and also semantically) correct. The SDF map parser does not attempt to parse the format string to check for possible user errors. It performs the placeholder substitutions and prints the resulting string to the SDF output file.

To check the validity of the generated mapped SDF file, read the mapped file by using the `read_sdf` command. For example,

```
pt_shell> read_sdf -syntax_only mapped.sdf
```

Bus Naming Conventions

When you specify mapping names with the `bus(string)` function, the SDF writer replaces the bus bit delimiting characters (specified by the construct `$SDF_BUSBIT QSTRING`) by using the appropriate delimiting characters.

For example, suppose the following lines are in the mapping file:

```
$SDF_BUSBIT "<>"  
...  
$23 bus(output_bus<5>)
```

When you specify the `write_sdf` command with the `-context verilog` option, the SDF writer prints the name as `output_bus[5]`. For more information, see [Using SDF Mapping](#). If more than one set of matching bus delimiters is found in a name, the SDF writer replaces only the matching set of delimiters at the end of the name string. If you do not specify bus delimiters, the names are printed unchanged.

Header Consistency Check for SDF Mapping Files

Each SDF mapping file defines SDF version and SDF bus delimiter characters in its header. When PrimeTime reads multiple SDF mapping files during one `write_sdf` command, it assumes all the headers have the same SDF version and bus delimiter characters.

Labeling Bus Arcs

A pin in the library file can be a bus. In the following example, the timing group on the pin defines multiple arcs.

Example 11-3 Timing Group on the Pin Defines Multiple Arcs

```
bus(A) {  
    bus_type : bus_11;  
    direction : input ;  
    timing() {  
        related_pin : "CK" ;  
        timing_type : setup_rising ;  
        timing_label : "tas";  
        intrinsic_rise : 1.12000 ;  
        intrinsic_fall : 1.12000 ;
```

In [Example 11-3](#) (assuming A is an 11-bit bus), 11 setup arcs are defined between bus A and CK. When you label such an arc, since only one timing label is present, PrimeTime attaches the same label (tas) to all arcs defined by the statement. When you access individual bit arcs, use the bus versions of the max/min rise/fall delay functions. For example,

to reference the arc corresponding to A[5], which is from CK to A[5], use the mapping functions shown in [Example 11-4](#) and [Example 11-5](#).

Example 11-4 Mapping Functions

```
max_rise_delay_bus(tas, CK, A[5]) #To get the max rise delay
min_rise_delay_bus(tas, CK, A[5]) #To get the min rise delay
max_fall_delay_bus(tas, CK, A[5]) #To get the max fall delay
min_fall_delay_bus(tas, CK, A[5]) #To get the min fall delay
```

Example 11-5 Mapping Functions

```
bus (In) {
    bus_type : "bus8";
    direction : input;
    capacitance : 1.46;
    fanout_load : 1.46;
}
bus (Q) {
    bus_type : "bus8";
    direction : output;
    timing () {
        timing_label : "In_to_Q"
        timing_sense : non_unate;
        intrinsic_rise : 2.251622;
        rise_resistance : 0.020878;
        intrinsic_fall : 2.571993;
        fall_resistance : 0.017073;
        related_pin : "In";
    }
}
```

To reference the arc from In[6] to Q[6] and print the following SDF line:

```
(IOPATH In[6] Q[6] (1.8::4.3) (1.8::4.0))
```

This is the SDF mapping file format string:

```
(IOPATH $1 $2 ($3::$4) ($5::$6))
```

The resulting SDF function mapping is as follows:

```
$1 bus(I1[6])
$2 bus(Q[6])
$3 min_rise_delay_bus(In_to_Q, In[6], Q[6])
$4 max_rise_delay_bus(In_to_Q, In[6], Q[6])
$5 min_fall_delay_bus(In_to_Q, In[6], Q[6])
$6 max_fall_delay_bus(In_to_Q, In[6], Q[6])
```

Note:

Avoid using a bus arc in nonbus max/min rise/fall delay functions.

SDF Mapping Limitations

The SDF mapping file syntax has the following limitations:

- You cannot use wildcard characters.
- You cannot specify instance-specific mapping format. An SDF mapping format must be specified for a library cell. The format is applied when writing SDF for every cell that is an instance of that library cell.
- If the SDF mapping file does not provide a format for every cell present in the library, PrimeTime prints these cells using the default format of the `write_sdf` command.

Mapped SDF File Examples

The following examples show different types of mapped SDF files.

Library File for Cells EXMP and FF1

A complete SDF mapping example for cells EXMP and FF1 is shown in [Example 11-7](#). The `min_pulse_width` arcs are labeled by using Style 2.

[Example 11-6](#) shows a library file, `example.lib`.

Example 11-6 Library File example.lib

```
library(example) {define("timing_label_mpw_low", "pin",
"string");
define("timing_label", "timing", "string");
define("timing_label_mpw_low", "pin", "string");

/* If using style #1 to specify min_pulse_width info
 */
/* define("timing_label_mpw_low", "min_pulse_width",
"string");
 */
/* define("timing_label_mpw_high", "min_pulse_width",
"string"); */

type(two_bit) {
    base_type : array;
    data_type : bit;
    bit_width : 2;
    bit_from : 1;
    bit_to : 0;
    downto : true;
}
```

```
cell (EXMP) {
    version : 1.00;
    area : 1.000000;
    cell_footprint : "EXMP";
    bus(IN) {
        bus_type : two_bit;
        direction : input;
        pin(IN[1:0]) {
            capacitance : 1;
        }
    }

    pin (A) {
        direction : input;
        capacitance : 0.49;
        fanout_load : 0.49;
        max_transition : 4.500000;
    }
    pin (B) {
        direction : input;
        capacitance : 0.51;
        fanout_load : 0.51;
        max_transition : 4.500000;
    }
    pin(Y){
        direction : output;
        capacitance : 0.00;
        function : "(A & B & IN[0] & IN[1])";
        timing () {
            related_pin : "A";
            timing_label : "A_Z";
            rise_transition (transitionDelay){ ... }
            fall_transition (transitionDelay){...}
            rise_propagation (propDelay){...}
            fall_propagation (propDelay){...}
        }
        timing () {
            related_pin : "B";
            timing_label : "B_Z";
            rise_transition (transitionDelay){...}
            fall_transition (transitionDelay){...}
            rise_propagation (propDelay){...}
            fall_propagation (propDelay) { ... }
        }
        timing () {
            related_pin : "IN[0]";
            timing_label : "IN[0]_Z";
            rise_transition (transitionDelay){...}
            fall_transition (transitionDelay){...}
            rise_propagation (propDelay){...}
            fall_propagation (propDelay){...}
        }
    }
}
```

```
timing () {
    related_pin : "IN[1]";
    timing_label : "IN[1]_Z";
    rise_transition (transitionDelay){...}
    fall_transition (transitionDelay){...}
    rise_propagation (propDelay){...}
    fall_propagation (propDelay){...}
}
...
}

cell(FD2) {
    area : 9;
    pin(D) {
        direction : input;
        capacitance : 1;
        timing() {
            timing_label : "setup_D";
            timing_type : setup_rising;
            intrinsic_rise : 0.85;
            intrinsic_fall : 0.85;
            related_pin : "CP";
        }
        timing() {
            timing_label : "hold_D";
            timing_type : hold_rising;
            intrinsic_rise : 0.4;
            intrinsic_fall : 0.4;
            related_pin : "CP";
        }
    }
    pin(CP) {
        direction : input;
        capacitance : 1;
        min_period : 5.0
        min_pulse_width_high: 1.5
        min_pulse_width_low: 1.5
        timing_label_mpw_low: "min_pulse_low_CP"
        timing_label_mpw_high: "min_pulse_high_CP"
    }
    pin(CD) {
        direction : input;
        capacitance : 2;
        timing() {
            timing_label : "recovery_rise_CD";
            timing_type : recovery_rising;
            intrinsic_rise : 0.5;
            related_pin : "CP";
        }
    }
}
...
```

```

pin(Q) {
    direction : output;
    function : "IQ";
    internal_node : "Q";
    timing() {
        timing_label : "CP_Q";
        timing_type : rising_edge;
        intrinsic_rise : 1.19;
        intrinsic_fall : 1.37;
        rise_resistance : 0.1458;
        fall_resistance : 0.0523;
        related_pin : "CP";
    }
    timing() {
        timing_label : "clear_Q";
        timing_type : clear;
        timing_sense : positive_unate;
        intrinsic_fall : 0.77; /* CP -> Q intrinsic - 0.6 ns */
        fall_resistance : 0.0523;
        related_pin : "CD";
    }
}
pin(QN) {
    direction : output;
    function : "IQN";
    internal_node : "QN";
    timing() {
        timing_label : "CP_QN";
        timing_type : rising_edge;
        intrinsic_rise : 1.47;
        intrinsic_fall : 1.67;
        rise_resistance : 0.1523;
        fall_resistance : 0.0523;
        related_pin : "CP";
    }
    timing() {
        timing_label : "preset_QN";
        timing_type : preset;
        timing_sense : negative_unate;
        intrinsic_rise : 0.87; /* CP -> QN intrinsic - 0.6 ns */
        rise_resistance : 0.1523;
        related_pin : "CD";
    }
}
}
}

```

Note:

The `min_pulse_width_low/high` constraints have been defined as attributes inside the pin group. The timing label attributes `timing_label_mpw_low` and `timing_label_mpw_high` are used to reference these constraints.

Mapped SDF File

[Example 11-7](#) shows a mapped SDF file, example.map.

Example 11-7 Mapped SDF File, example.map

```

# This is a comment
$SDF_VERSION "OVI 2.1"
$SDF_BUSBIT "[ ]"
$SDF_CELL EXMP
"(DELAY
  (ABSOLUTE
    (IOPATH $1 $2 ($3:$4) ($5:$6))
    (IOPATH $7 $2 ($8:$9) ($10:$11))
  )
)
$3 min_rise_delay(A_Z)
$4 max_rise_delay(A_Z)
$5 min_fall_delay(A_Z)
$6 max_fall_delay(A_Z)
$1 pin(A)
$2 pin(Z)
$7 bus(IN[0])
$8 min_rise_delay(IN[0]_Z)
$9 max_rise_delay(IN[0]_Z)
$10 min_fall_delay(IN[0]_Z)
$11 max_fall_delay(IN[0]_Z)
$SDF_CELL_END

$SDF_CELL DFF
"(DELAY
  (ABSOLUTE
    (IOPATH (posedge $1) $2 ($3:$4) ($5:$6))
    (IOPATH (posedge $7) $8 ($9:$10) ($11:$12))
    (IOPATH (negedge $13) $2 (:)) ($16:$17))
    (IOPATH (negedge $13) $8($18:$19) (:))
  )
)
(TIMINGCHECK
  (SETUP $20(posedge $1)) ($30:$31))
  (HOLD $20 (posedge $1)) ($32:$33))
  (WIDTH (negedge $1) ($34))
  (RECOVERY (posedge $13) (posedge $1)) ($36))
  (PERIOD (posedge $1) ($37)))
)
$1 pin(CP)
$2 pin(Q)
$3 min_rise_delay(CP_Q)
$4 max_rise_delay(CP_Q)
$5 min_fall_delay(CP_Q)
$6 max_fall_delay(CP_Q)
$7 pin(CP)
$8 pin(QN)
$9 min_rise_delay(CP_QN)

```

```

$10 max_rise_delay(CP_QN)
$11 min_fall_delay(CP_QN)
$12 max_fall_delay(CP_QN)
$13 pin(CD)
$16 min_fall_delay(clear_Q)
$17 max_fall_delay(clear_Q)
$18 min_rise_delay(preset_QN)
$19 max_rise_delay(preset_QN)
$20 pin(D)
$30 min_rise_delay(setup_D)
$31 max_rise_delay(setup_D)
$32 min_rise_delay(hold_D)
$33 max_rise_delay(hold_D)
$34 min_fall_delay(min_pulse_low_CP)
$36 max_rise_delay(recovery_rise_CD)
$37 min_period(CP)

```

Three-State Buffers

For an example of files for a three-state noninverting buffer, see

- Library file – [Example 11-8](#)
- Mapped SDF file – [Example 11-9](#)

Example 11-8 Library File for Three-State Noninverting Buffer

```

/
*-----
Internal Tristate Non-Inverting Buffer, Positive Enable, 1x
Drive
-----*
/
define("timing_label", "timing", "string");
define("timing_label_mpw_low", "pin", "string");
define("timing_label_mpw_low", "pin", "string");

cell(BTS) {
    area : 63 ;
    pin(Z) {
        direction : output ;
        function : "A";
        max_capacitance : 0.14000 ;
        capacitance : 0.00420 ;
        three_state : "E'";
        timing() {
            related_pin : "A" ;
            timing_sense : positive_unate ;
            timing_label : "A_Z";
            cell_rise(table_1) {
                values ( ... ) ;
            }
        }
    }
}

```

```
    rise_transition(table_1) {
        values ( ... ) ;
    }
    cell_fall(table_1) {
        values ( ... ) ;
    }
    fall_transition(table_1) {
        values ( ... ) ;
    }
    intrinsic_rise : 0.31166 ;
    intrinsic_fall : 0.40353 ;
}
timing() {
    related_pin : "E" ;
    timing_label : "E_Z_3s_enable";
    cell_rise(table_1) {
        values ( ... ) ;
    }
    rise_transition(table_1) {
        values ( ... ) ;
    }
    cell_fall(table_1) {
        values ( ... ) ;
    }
    fall_transition(table_1) {
        values ( ... ) ;
    }
    intrinsic_rise : 0.22159 ;
    intrinsic_fall : 0.27933 ;
}
timing() {
    related_pin : "E" ;
    timing_type : three_state_disable ;
    timing_label : "E_Z_3s_disable";
    cell_rise(table_10) {
        values ( ... ) ;
    }
    rise_transition(table_10) {
        values ( ... ) ;
    }
    cell_fall(table_10) {
        values ( ... ) ;
    }
    fall_transition(table_10) {
        values ( ... ) ;
    }
    intrinsic_rise : 0.30693 ;
    intrinsic_fall : 0.19860 ;
}
}
```

```

pin(A) {
    direction : input ;
    capacitance : 0.00423 ;
}
pin(E) {
    direction : input ;
    capacitance : 0.01035 ;
}
}

```

Example 11-9 Mapped SDF File for Three-State Noninverting Buffer

```

$SDF_VERSION "OVI 2.1"
$SDF_BUSBIT "[]"
$SDF_CELL BTS
" (DELAY
  (ABSOLUTE
    (IOPATH $1 $3 ($4:::$5) ($6:::$7))
    (IOPATH $2 $3 ($8:::$9) ($10:::$11) ($12:::$13) ($8:::$9)
($14:::$15) ($10:::$11))
  )
#####
$1 pin(A)
$2 pin(E)
$3 pin(Z)
#####
$4 min_rise_delay(A_Z)
$5 max_rise_delay(A_Z)
$6 min_fall_delay(A_Z)
$7 max_fall_delay(A_Z)
#####
$8 min_rise_delay(E_Z_3s_enable)
$9 max_rise_delay(E_Z_3s_enable)
$10 min_fall_delay(E_Z_3s_enable)
$11 max_fall_delay(E_Z_3s_enable)
$12 min_rise_delay(E_Z_3s_disable)
$13 max_rise_delay(E_Z_3s_disable)
$14 min_fall_delay(E_Z_3s_disable)
$15 max_fall_delay(E_Z_3s_disable)
#####

```

Managing SDF Large Files

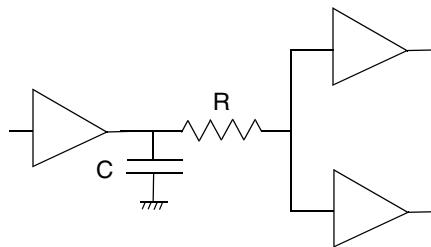
To compress large SDF files to a UNIX gzip file, use the `write_sdf` command with the `-compress` option:

```
pt_shell> write_sdf -compress gzip 1.sdf.gz
```

Setting Lumped Parasitic Resistances and Capacitances

You can annotate resistance and capacitance (RC) on nets, as shown in [Figure 11-6](#). Even if you annotated all the delays of the design with SDF, you might want to annotate parasitics to perform certain design rule checks, such as maximum transition or maximum capacitance.

Figure 11-6 Lumped RC



Setting lumped resistance or capacitance with the `set_resistance` or `set_load` command temporarily overrides the wire load model or detailed parasitics for a net. Removing lumped resistance or capacitance from a net with the `remove_resistance` or `remove_capacitance` command causes the net to revert back to its previous form of parasitic data.

You can set lumped resistance and capacitance separately. For example, you can read in detailed parasitics for a net, and then override just the capacitance for that net with the `set_load` command. In that case, PrimeTime still uses the detailed parasitic information to calculate the net resistance.

Setting Net Capacitance

The `set_load` command sets the net capacitance values on ports and nets. If the current design is hierarchical, you must link it with the `link` command.

By default, the total capacitance on a net is the sum of all of pin, port, and wire capacitance values associated with the net. A capacitance value you specify overrides the internally estimated net capacitance.

Use the `set_load` command for nets at lower levels of the design hierarchy. Specify these nets as `BLOCK1/BLOCK2/NET_NAME`.

If you use the `-wire_load` option, the capacitance value is set as a wire capacitance on the specified port and the value is counted as part of the total wire capacitance (not as part of the pin or port capacitance).

To view capacitance values on ports, use `report_port`. To view capacitance values on nets, use the `report_net` command.

Setting Net Resistance

The `set_resistance` command sets net resistance values. The specified resistance value overrides the internally estimated net resistance value.

You can also use the `set_resistance` command for nets at lower levels of the design hierarchy. You can specify these nets as `BLOCK1/BLOCK2/NET_NAME`.

To view resistance values, use the `report_net` command. To remove resistance values annotated on specified nets, use the `remove_resistance` command. To remove resistance annotated on the entire design, use the `reset_design` command.

Reduced and Detailed Parasitics

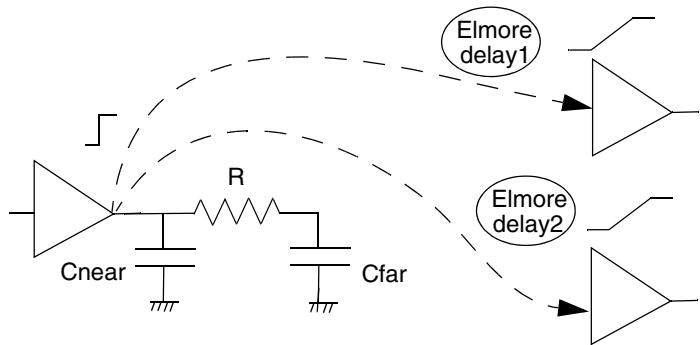
PrimeTime provides features that enable you to annotate reduced parasitics and detailed parasitics.

Reduced Parasitics

Reduced resistance and capacitance represents the RC within a design from a driver standpoint: two capacitors and one resistor, as shown in [Figure 11-7](#). The net delays are typically annotated with an Elmore delay, which is the most commonly used representation. For this model the following rules apply:

- RC is represented in terms of a pi model (two C and one R).
- Net delay from the driver to the net fanout provided is an Elmore delay.
- The RC pi model is used to compute the slew at the driver pin.
- Slew degradation (transition time) from the driver pin to each net load pin is computed if the Elmore delay is significant (more than 20 percent of driver transition time). The slew degradation is based on a published paper by E. G. Friedman and J. H. Mulligan, Jr., titled “Ramp Input Response of RC Tree Network,” in Analog Integrated Circuits and Signal Processing, Volume 14, No. 1/2, pp. 53-58, September 1997.

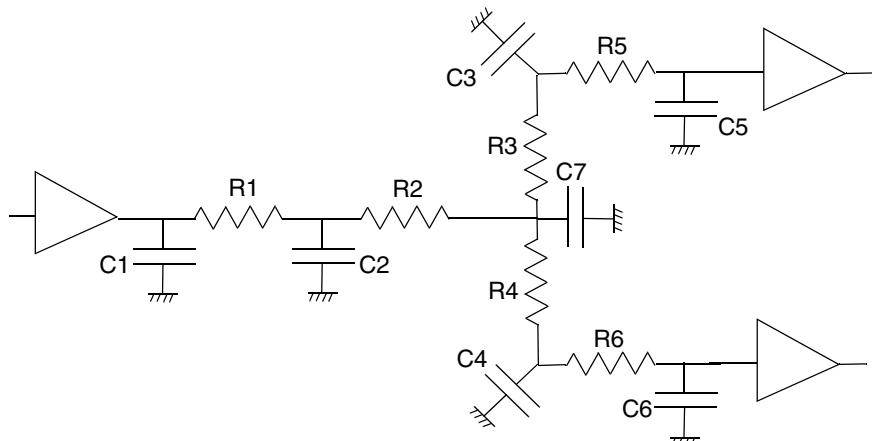
The model in [Figure 11-7](#) calculates the cell delay and the driver output slew more accurately than the lumped RC model. Reduced RC is annotated with RSPEF (Reduced Standard Parasitic Format) or SPEF (Standard Parasitic Exchange Format).

Figure 11-7 Reduced RC

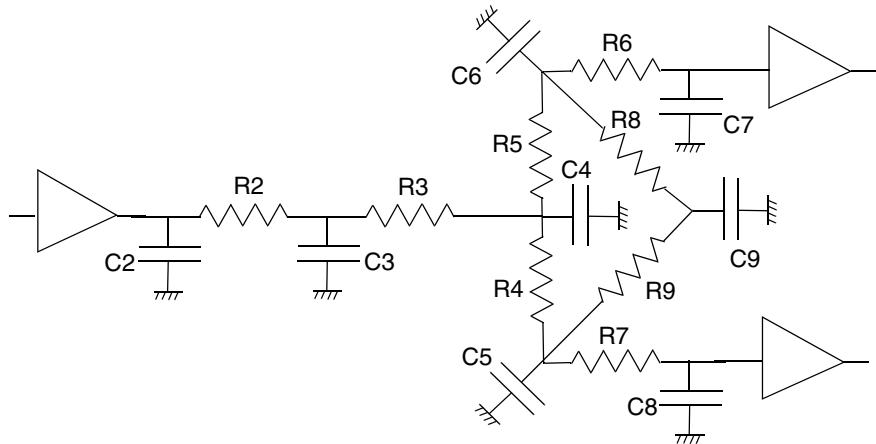
Detailed Parasitics

You can annotate detailed parasitics into PrimeTime and annotate each physical segment of the routed netlist in the form of resistance and capacitance (see [Figure 11-8](#)). Annotating detailed parasitics is very accurate but more time-consuming than annotating lumped parasitics. Because of the potential complexity of the RC network, PrimeTime takes longer to calculate the pin-to-pin delays in the netlist.

This RC network is used to compute effective capacitance ($C_{effective}$), slew, and delays at each subnode of the net. PrimeTime can read detailed RC in SPEF format.

Figure 11-8 Detailed RC

Use this model for netlists that have critical timing delays, such as clock trees. This model can produce more accurate results, especially in deep submicron designs where net delays are more significant compared to cell delays. The detailed RC network supports meshes, as shown in [Figure 11-9](#).

Figure 11-9 Meshed RC

Supported File Formats for Parasitic Annotation

PrimeTime supports the open Verilog International (OVI) SPEF format for parasitic annotation.

The following limitations apply to RSPF and DSPF constructs:

- SPICE inductors (Lxxx) and lines (Txxx) are allowed in the DSPF file, but they are ignored.
- Physical coordinates of pins and instances are ignored.
- The `BUSBIT` construct is not supported.
- Instances listed in the RSPF and DSPF files in the SPICE section are ignored. They are not checked to determine whether they match the current design loaded in PrimeTime.
- Resistors cannot be connected to ground. PrimeTime ignores such resistors and displays a warning about them.
- Capacitors must be connected to ground. PrimeTime ignores node-to-node coupling capacitors and displays a warning about them.

The following limitations apply to SPEF constructs:

- Inductors are ignored.
- Poles and residue descriptions on reduced nets are not supported.
- Resistors cannot be connected to ground. PrimeTime ignores resistors connected to ground and displays a warning.

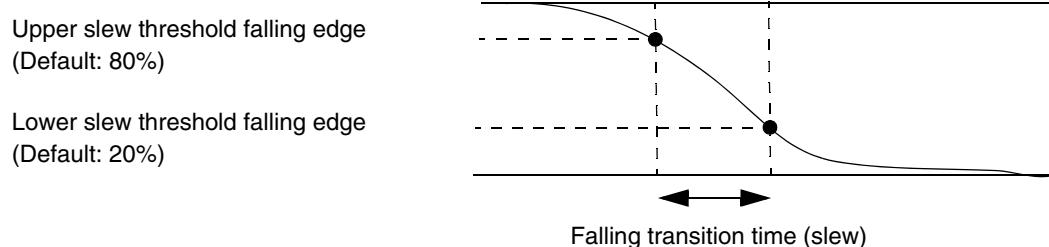
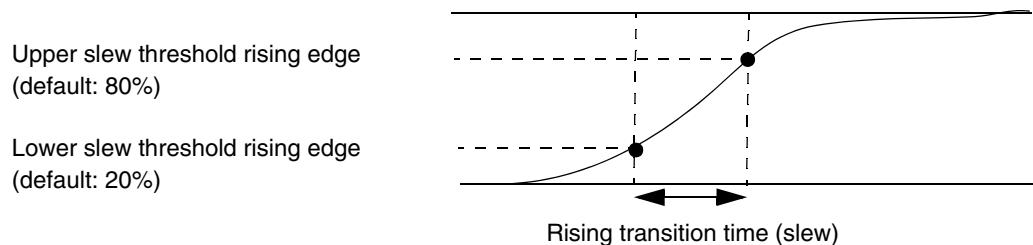
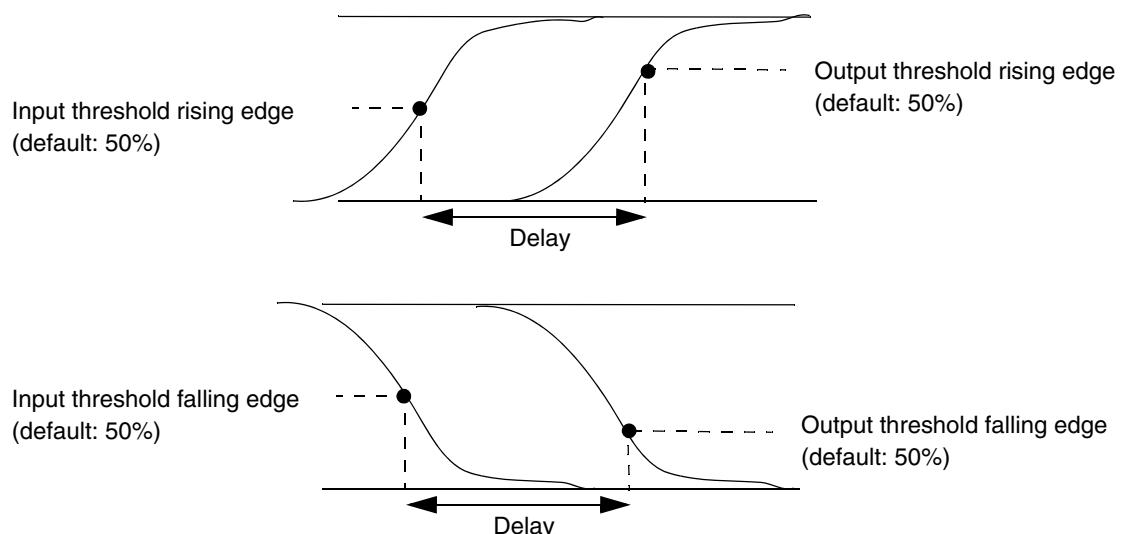
- Cross-coupling capacitors are split to ground unless you are using PrimeTime SI and crosstalk analysis is enabled.

Characterization Trip Points

The characterization trip points of a design (waveform measurement thresholds) affect the calculation of delays and transition times by PrimeTime. PrimeTime establishes the trip points as follows:

1. If pin-level thresholds are defined, the application tools use those. Pin-level thresholds override library-level thresholds with the same name.
2. If pin-level thresholds are not defined, but library-level thresholds are, the application tools use the library-level thresholds.
3. If neither pin-level nor library-level thresholds are defined, the application tools use the thresholds defined in the main library (the first library in the link path).
4. If none of the previous thresholds are defined, the application tools use default trip points at 20 percent and 80 percent of the rail voltage for transition time calculations and 50 percent of the rail voltage for delay calculations.
5. If the trip points defined by any of these methods are not valid (for instance, 0 percent and 100 percent), the trip points are set to 5 percent and 95 percent of the rail voltage for transition time calculations and 50 percent of the rail voltage for delay calculations.

[Figure 11-10](#) and [Figure 11-11](#) show some signal waveforms and the trip points used to calculate transition times (slews) and delays. The default trip point values define slew as the time that a signal takes to change from 20 to 80 percent or from 80 to 20 percent of the rail voltage, and they define cell delay as the amount of time from the input signal reaching 50 percent of the rail voltage to the output signal reaching 50 percent of the rail voltage.

Figure 11-10 Slew Transition Points**Figure 11-11 Input/Output Transition Points**

To check the threshold levels defined in a library, use the `report_lib` command. To check the threshold settings used for a particular delay calculation arc, use the `report_delay_calculation -thresholds` command.

Reading Parasitic Files

The `read_parasitics` command reads a parasitic data file in SPEF, RSPF (version IEEE 1481-1999), or SBPF (Synopsys Binary Parasitic Format) format, and it annotates the current design with the parasitic information. An SPEF or RSPF file is an ASCII file that can be compressed with gzip. Specifying the format in the command is optional because the reader can automatically determine the file type.

Net and instance pin names in the design must match the names in the parasitics file. For example, if you create the parasitics file from a design using VHDL naming conventions, the design name must use VHDL naming conventions.

When reading parasitic files, by default, PrimeTime assumes that capacitance values specified in the SPEF files do not include the pin capacitance. PrimeTime uses the pin capacitance values specified in the Synopsys design libraries; any pin capacitance values specified in SPEF are ignored. You must ensure that the coupling capacitance in the SPEF file are symmetric. To verify that the coupling capacitance contains only symmetric coupling, read in the design and then use both the `-syntax_only` and `-keep_capacitive_coupling` options of the `read_parasitics` command. PrimeTime checks for asymmetric coupling and issues warning messages when this type of issue is identified. Ensure the SPEF files contain valid coupling capacitance before proceeding.

The reduced and detailed RC networks specified in SPEF files are used to compute effective capacitance dynamically during delay calculation. The capacitance value reported by most report commands, such as the `report_timing` and `report_net` command, is the lumped capacitance, also known as Ctotal. Ctotal is the sum of all capacitance values of a net as specified in the SPEF, to which pin capacitance is also added.

Parasitic data files are often very large and time-consuming for PrimeTime to read. To minimize the read time, make sure that the data file is on a local disk that can be accessed directly, not across a network. To avoid using disk swap space, having enough memory is also helpful. Compressing an SPEF file using gzip can improve overall processing time because the file is much smaller.

For more information, see the following topics:

- [Applying Location Transformations to Parasitic Data](#)
- [Multivalued SPEF Files](#)
- [Converting a SPEF Parasitics File to SBPF](#)
- [Reading Multiple Parasitic Files](#)
- [Checking the Annotated Nets](#)
- [Scaling Parasitic Values](#)
- [Reading Parasitics for Incremental Timing Analysis](#)

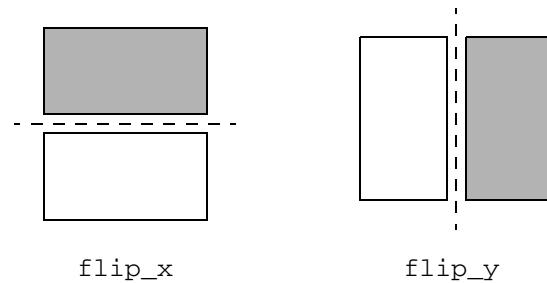
Applying Location Transformations to Parasitic Data

You can apply location transformations, such as offsets, rotations, and flips, to the parasitic data. To do this, use the `read_parasitics` command with the following options:

```
read_parasitics
-x_offset x_offset_in_nm
-y_offset y_offset_in_nm
-rotation rotate_none | rotate_90 | rotate_180 | rotate_270
-axis_flip flip_x | flip_y | flip_both | flip_none
```

[Figure 11-12](#) shows the transformations specified by the `-axis_flip flip_x` and `-axis_flip flip_y` options.

Figure 11-12 Transformations Specified by the -axis_flip flip_x and -axis_flip flip_y Options



To apply location transformations to multiple instances, specify the options in a list. Each argument in the list applies a transformation to the corresponding instance specified by the `-path` option. For example:

```
read_parasitics \
-path { n1 n2 n3 } \
-x_offset { 1000000 2000000 3000000 } \
-y_offset { 2000000 3000000 1000000 } \
-rotation { rotate_90 rotate_none rotate_270 } \
-axis_flip { flip_y flip_x flip_none }
```

In this example, instance n1 has an x-offset of 1,000,000 nm, a y-offset of 2,000,000 nm, a rotation of 90 degrees, and a flip across the y-axis. Instance n2 has an x-offset of 2,000,000 nm, a y-offset of 3,000,000 nm, a rotation of 0 degrees, and a flip across the x-axis.

Multivalued SPEF Files

Double-patterning technology for 20-nm processes results in additional variation in the coupling capacitances between wires. A multivalued SPEF file models this coupling capacitance variation with minimum, typical, and maximum values; the grounded capacitances and resistances are represented by single values as in a regular SPEF file.

PrimeTime SI can use the minimum and maximum values in multivalued SPEF files for setup and hold analysis, as shown in [Figure 11-13](#) and [Table 11-4](#).

Figure 11-13 Setup and Hold Analysis in PrimeTime

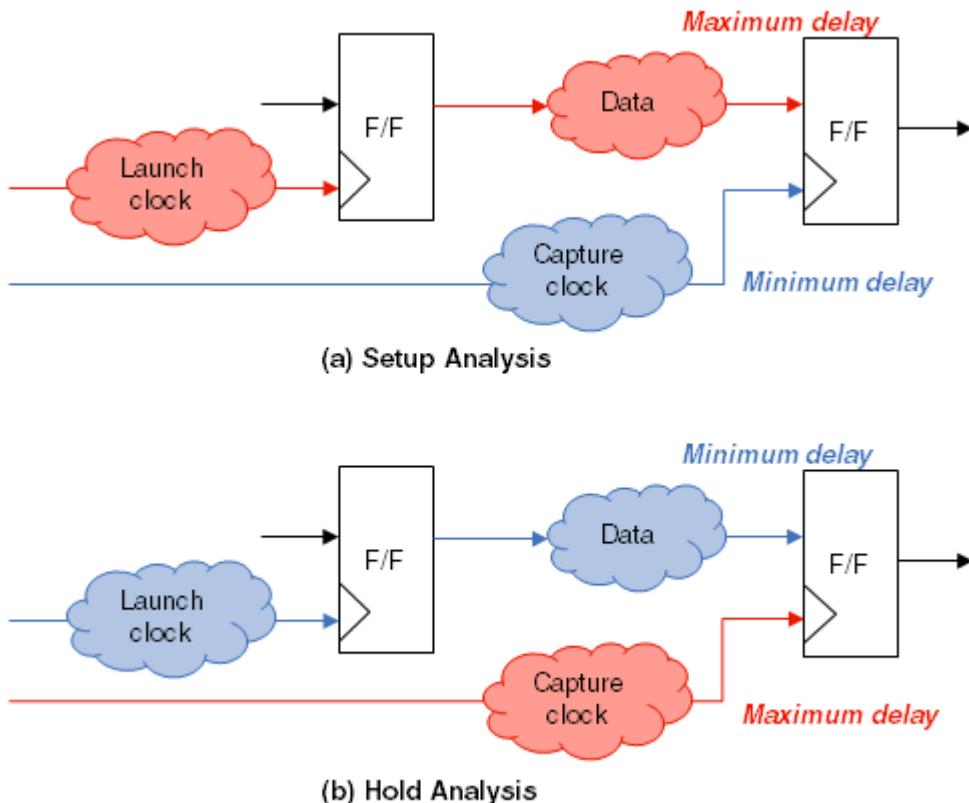


Table 11-4 Use of Minimum or Maximum Values from Multivalued SPEF File During Setup and Hold Analysis

Analysis type		Launch clock and data path delays are calculated with	Capture clock path delays are calculated with
Uncoupled delay analysis	Setup	Maximum coupling capacitance values	Minimum coupling capacitance values
	Hold	Minimum coupling capacitance values	Maximum coupling capacitance values

Table 11-4 Use of Minimum or Maximum Values from Multivalued SPEF File During Setup and Hold Analysis (Continued)

Analysis type		Launch clock and data path delays are calculated with	Capture clock path delays are calculated with
Crosstalk delay analysis	Setup	Maximum coupling capacitance values	Maximum coupling capacitance values
	Hold	Maximum coupling capacitance values	Maximum coupling capacitance values

Requirements for Multivalued SPEF Files

To analyze double patterning effects with PrimeTime SI, you should use a multivalued SPEF file that meets the following requirements:

- The coupling capacitances must be represented as triplets of minimum, typical, and maximum values. Values in the triplet must be listed in increasing numerical order.
- The grounded capacitances and resistances can be represented as either single values or triplets. If they are represented as triplets, all three values must be equal.

If the multivalued SPEF file does not meet the preceding requirements, PrimeTime SI can still read and annotate the parasitic file successfully, but the timing analysis results might not bound the worst case of the shifted masks.

Reading Multivalued SPEF Files

To read multivalued SPEF files, perform the following steps with PrimeTime SI:

1. Set the following variable:

```
set_app_var si_enable_multi_valued_coupling_capacitance true
```

2. Enable signal integrity analysis:

```
set_app_var si_enable_analysis true
```

3. Read the parasitic information with the `-keep_capacitive_coupling` option:

```
read_parasitics -format SPEF -keep_capacitive_coupling file_names
```

Limitations of Multivalued SPEF

The tool uses only the maximum values for

- Ground capacitances and resistances
- Attributes and reporting
- The `create_ilm`, `extract_model`, `write_parasitics`, `write_spice_deck`, and HyperScale commands

Reading a Single Corner From Multicorner Parasitic Data

You can annotate your design with the parasitic data from a single corner of a multicorner SPEF file. To do this,

1. Specify the corner name:

```
set_app_var parasitic_corner_name corner_name
```

2. Read the parasitic data with the `-keep_capacitive_coupling` option:

```
read_parasitics -format SPEF  
-keep_capacitive_coupling multicorner_SPEF_file
```

The tool reports the corner name in the `read_parasitics` report:

```
*****  
Report : read_parasitics multi_corner_file.spef  
-keep_capacitive_coupling  
...  
*****  
0 error(s)  
Format is SPEF  
Annotated corner name      :  name  
Annotated nets              :  ...  
Annotated capacitances     :  ...  
Annotated resistances      :  ...  
Annotated coupling capacitances :  ...  
Annotated PI models        :  ...  
Annotated Elmore delays    :  ...
```

Converting a SPEF Parasitics File to SBPF

The Synopsys Binary Parasitic Format (SBPF) is the most efficient format for parasitic data because it occupies less disk space and can be read much faster than the same data stored in the other formats. Consider using SBPF if your tools support it.

To convert parasitic data from SPEF to SBPF,

1. Read the parasitic data from a SPEF file by using the `read_parasitics` command:

```
pt_shell> read_parasitics -keep_capacitive_coupling spef_file
```

Note:

Ensure that the coupling capacitances in the SPEF file contain only symmetric couplings.

2. Make any desired modifications to the parasitic data.
3. Write the data to a SBPF file by using the `write_parasitics` command:

```
pt_shell> write_parasitics -format SBPF file_name
```

4. You can quickly read back in the SBPF file by using the `read_parasitics` command:

```
pt_shell> read_parasitics -format SBPF file_name
```

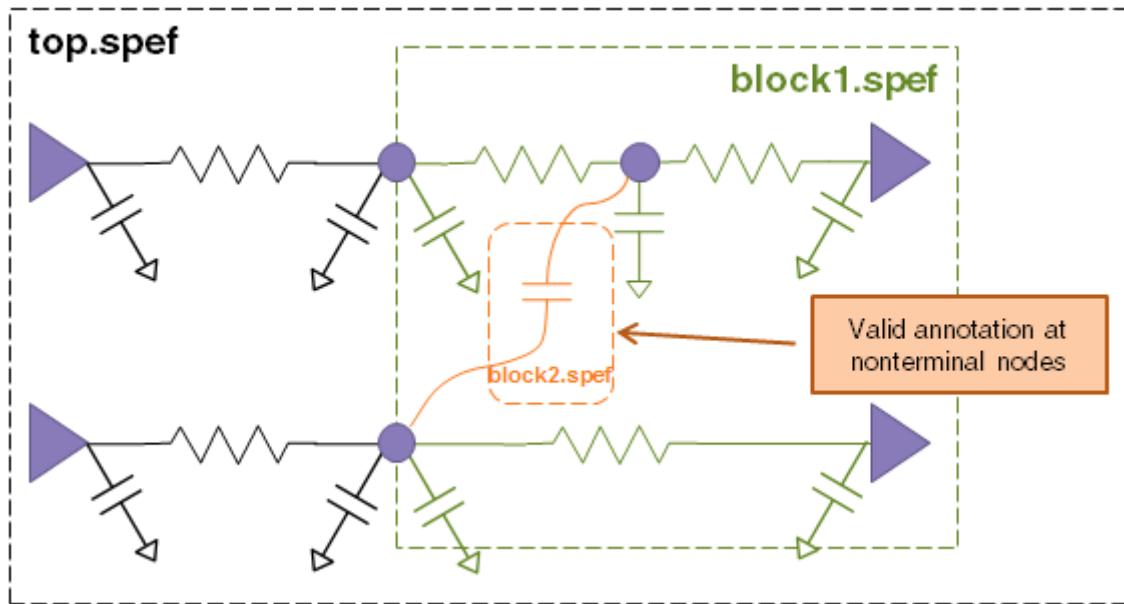
Reading Multiple Parasitic Files

You can read multiple parasitic files. For example, you might have separate parasitic files for your subblocks and a separate file for your top-level interconnect. Parasitics for chip-level timing are often read incrementally. Hierarchical parasitics for each instance are read separately and stitched with top-level parasitics. Use the following recommended flow for reading multiple hierarchical incremental parasitic files:

```
read_parasitics A.sbpf -path [all_instances -hierarchy BLKA]  
read_parasitics B.sbpf  
read_parasitics C.sbpf  
read_parasitics D.sbpf  
read_parasitics chip_file_name  
report_annotated_parasitics -check
```

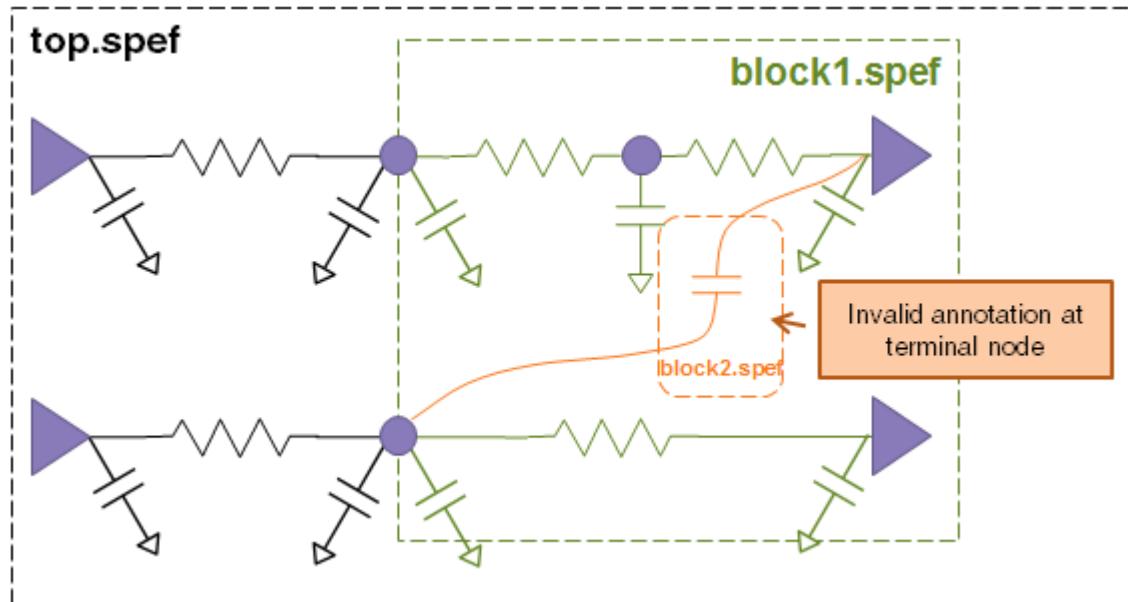
When reading multiple parasitic files, PrimeTime stitches subsequent parasitic annotations that occur only on nonterminal nodes. For example, in [Figure 11-14](#), the annotation specified by the block2.spef file is valid because it does not occur at a terminal node.

Figure 11-14 Second Annotation is Valid



However, in [Figure 11-15](#), the annotation specified by **block2.spef** occurs at a terminal node and is therefore invalid. PrimeTime ignores this annotation and issues a PARA-114 error message.

Figure 11-15 Second Annotation is Invalid



Checking the Annotated Nets

By default, the `read_parasitics` command does a check for incomplete annotated nets on the nets it annotates. It then executes the `report_annotated_parasitics -check` command. Use the `-path` option to take a list of instances so that multiple instantiations of the same block-level design can all be annotated together. This can significantly reduce both the runtime and memory required for reading parasitics in PrimeTime. By executing the commands in the recommended flow and explicitly using the `report_annotated_parasitics -check` command, all of the parasitic files are read in, then the entire design is checked to confirm and report the successful annotation of all nets. The report contains all nets, not just the nets that are annotated for the last command.

Some messages issued during `read_parasitics`, `report_annotated_parasitics -check`, `read_sdf`, and an implicit or explicit `update_timing` command have a default limit each time the command is invoked. A summary of the messages affected and other useful information is stored in the log file and also printed at the end of the PrimeTime session. The `sh_message_limit` variable controls the default message limit and the `sh_limited_messages` variable controls the messages affected by this feature.

Scaling Parasitic Values

You can scale parasitic values that have been read into the design. There are three separate scaling factors for resistors, ground capacitors, and (for PrimeTime SI users) coupling capacitors. To scale the parasitics, you use the `scale_parasitics` command.

Each factor is multiplied against all back-annotated values of the specified type throughout the design. A factor greater than 1.0 increases the parasitic component values, whereas a factor less than 1.0 decreases the parasitic component values. Factors must be greater than or equal to 0.0.

The `scale_parasitics` command operates immediately to modify the parasitics in memory. If you use the `scale_parasitics` command more than one time in a session, the factor is applied to the scaled values, not the original values read from the parasitic data file. If you write the design parasitics with the `write_parasitics` command, the scaled values (not original values) are written.

Net-Specific Parasitic Scaling

By using the `scale_parasitics` command with a list of nets, you can apply different scaling factors to individual nets instead of using a global scaling factor. You can scale resistance, ground capacitance, and coupling capacitance values.

The `report_scale_parasitics` command allows you to see a report of the scale parasitics. The `reset_scale_parasitics` command allows you to reset the parasitics to

the global value. You can inspect the changes to the scaled parasitics using the `report_annotated_parasitics -list_annotated` command.

Note:

Net-specific parasitic scaling does not incur a full timing update, only an incremental update.

Ground-Capacitance and Resistance Scaling

You can scale all ground capacitances on a net by a given scaling factor. A scaling factor is a positive floating point number greater than zero. For example,

```
pt_shell> scale_parasitics [get_net n1] -ground_capacitance_factor 1.3
```

You can scale all resistances of a net by another scaling factor:

```
pt_shell> scale_parasitics [get_net n1] -resistance_factor 1.22
```

If you scale the net multiple times or you scale globally, followed by net-specific scaling, the last command issued takes effect with respect to the original (not the previous value). For example, if the following sequence is executed, the ground capacitance of net n1 is scaled by 1.2, *not* 1.3 multiplied by 1.2:

```
pt_shell> scale_parasitics -ground_capacitance_factor 1.3
pt_shell> scale_parasitics [get_net n1] -ground_capacitance_factor 1.2
```

Coupling-Capacitance Scaling

You can scale all the coupling capacitances of a net by a scaling factor.

```
pt_shell> scale_parasitics -coupling_capacitance_factor 1.3 [get_net n1]
```

If you scale the two coupled nets by different factors, then the later command takes effect. For example,

```
pt_shell> scale_parasitics -coupling_capacitance_factor 1.3 [get_net n1]
pt_shell> scale_parasitics -coupling_capacitance_factor 1.2 [get_net n2]
```

The net effect is that the all the coupling of net n1 is scaled by 1.3 except the coupling to n2, and all the coupling of net n2 is scaled by 1.2 including the coupling to n1.

Resetting Scale Parasitics

Use the `reset_scale_parasitics` command to reset the scaled nets to their original values. You can only issue this command on nets that have been previously scaled.

```
pt_shell> reset_scale_parasitics [get_nets n1]
```

The `reset_scale_parasitics` command returns the values to the original values, not necessarily the previous values. For example,

```
pt_shell> scale_parasitics -ground_capacitance_factor 1.3 #scale all nets
pt_shell> scale_parasitics [get_net n1] \
           -ground_capacitance_factor 1.2 ##scale n1
pt_shell> reset_scale_parasitics [get_net n1] ## reset scale n1
```

The net effect is that there is no scaling on net n1.

You can also use the `reset_scale_parasitics` command to regain original coupling values:

```
pt_shell> scale_parasitics -coupling_capacitance_factor 1.3 \
           [get_net n1]
pt_shell> scale_parasitics -coupling_capacitance_factor 1.2 [get_net n2]
pt_shell> reset_scale_parasitics [get_net n1]
```

The net effect of this sequence of commands is that the coupling of n1 is unchanged, and all coupling of n2 except the coupling between n1 and n2 is scaled by 1.2.

Reporting Scale Parasitics

Use the `report_scale_parasitics` command to look at the current scale factors of all scaled nets or a list of scaled nets:

```
pt_shell> report_scale_parasitics [get_nets n1]
```

Net name	Ground cap factor	Resistance factor	Coupling factor
n1	--	1.2	0.98

This means that net n1 is not scaled for ground capacitance: its resistance is scaled by a factor of 1.2, and its coupling capacitance is scaled by a factor of 0.98.

Note:

Be aware of the following possible scenario:

```
pt_shell> scale_parasitics -coupling_capacitance_factor 1.3 \
           [get_net n1]
pt_shell> scale_parasitics -coupling_capacitance_factor 1.2 \
           [get_net n2]
pt_shell> report_scale_parasitics [get_net n1]
```

This shows a coupling scale factor of 1.3 even though the coupling between net n1 and net n2 is scaled by a different factor, 1.2.

Net-Specific Parasitic Scaling Examples

The following examples show net-specific parasitic scaling:

- [Example 1: Single Net Scaling](#)
- [Example 2: Physical Block \(Net\) Scaling](#)
- [Example 3: Scaling Due to Net Separation](#)

Example 1: Single Net Scaling

This example shows scaling the resistance and ground capacitance of a selected net.

[Example 11-10](#) shows the output of the `report_annotated_parasitics` command without any scaling.

Example 11-10 Report of Annotated Parasitics Without Scaling

```
*****
Report : annotated_parasitics
    -internal_nets
    -boundary_nets
    -list_annotated
    -max_nets 10
...
*****
1. ADDR[1] (driver: port ADDR[1])

      C in pF          Node      Pin/Port
      -----          -----
0.00842337        1    ADDR[1] (driver) [+pin_cap=0pF]
0.0170247         2        --
0.00531228        3        --
0.00702817        4        --
0.00487712        5        --
0.00397756        6        --
0.00590386        7        --
0.0122464         8        --
0.00170559        9        --
0.00611351        10       --
0.00304156        11       --
0.00395301        12       --
0.00607125        13       --
0.00227354        14       --
0.0119327         15       --
0.00564987        16       --
0.00749174        17       --
0.00585252        18       --
0.0083778         19       --
0.0020378         20       --
0.00111427        21    U527/A1 (load) [+pin_cap=0.00103pF]
0.00159791        22    U459/B1 (load) [+pin_cap=0.00112pF]
```

1e-06	23	U2053/A1 (load) [+pin_cap=0.00075pF]
1e-06	24	U483/A1 (load) [+pin_cap=0.00107pF]
1e-06	25	U534/A1 (load) [+pin_cap=0.00191pF]
1e-06	26	U503/A1 (load) [+pin_cap=0.00189pF]

0.13201	Total
---------	-------

CC in pF	Local Node	Other Node
0.00197175	13	n4250:13
0.00139652	17	n4250:6
0.00218098	3	n2124:2
0.00141374	1 ADDR[1] (driver)	REG_DATA[22]:2
0.000183647	16	n8376:2
0.000175112	6	n13565:2
0.000433937	6	n13599:4
0.000633856	8	n4750ASTipoNet3769:1
0.000346092	8	n1946:1
0.00274598	3	n3117:1
0.00033236	17	n7632:1
0.00067407	12	n11321:1
0.000256377	2	BW0_30_:2
0.000994872	3	n225:2
0.000264488	15	n1785:2
0.00157215	9	n5870ASThfnNet347:5
0.00130062	2	REG_DATA[26]:2
0.00170201	4	n2248ASThfnNet432:10
0.00157756	14	n2248ASThfnNet432:13

0.0201561	Total
-----------	-------

R in Kohm	Left node	Right Node
0.198293	1 ADDR[1] (driver)	2
0.0190159	2	8
0.00350119	3	11
0.027464	3	5
0.00235238	4	14
0.00688478	4	11
0.00561526	5	20
0.000852706	6	16
0.0239033	6	21 U527/A1 (load)
0.0380269	6	22 U459/B1 (load)
0.0397274	6	12
0.0257463	7	9
0.0313717	7	12
0.00521104	8	19
0.00360633	9	10
0.0152421	10	13
0.0923591	12	21 U527/A1 (load)
0.146931	12	22 U459/B1 (load)
0.00413988	13	17
0.00264083	14	17

0.0060443	15	18				
0.00321024	15	19				
0.0329064	16	23 U2053/A1 (load)				
0.0285095	16	24 U483/A1 (load)				
0.0187162	16	25 U534/A1 (load)				
0.0200831	16	26 U503/A1 (load)				
0.00884356	18	20				
0.0230755	26 U503/A1 (load)	25 U534/A1 (load)				
0.0667172	21 U527/A1 (load)	22 U459/B1 (load)				
0.00850544	23 U2053/A1 (load)	24 U483/A1 (load)				
<hr/>						
0.909497	Total					
<hr/>						
Net Type	Total	Lumped	RC pi	RC network	Coupled network	Not Annotated
Internal nets	0	0	0	0	0	0
-Driverless nets		0	0	0	0	0
<hr/>						
Boundary/port nets	1	0	0	0	1	0
-Driverless nets		0	0	0	0	0
<hr/>						
	1	0	0	0	1	0

After the parasitics are scaled for resistance and ground capacitance, the `report_annotated_parasitics` command generates the report shown in [Example 11-11](#).

Example 11-11 Report of Annotated Parasitics With Scaling

```
*****
Report : annotated_parasitics
  -internal_nets
  -boundary_nets
  -list_annotated
  -max_nets 10
...
*****
```

1. ADDR[1] (driver: port ADDR[1])

C in pF	Node	Pin/Port
0.0168467	1	ADDR[1] (driver) [+pin_cap=0pF]
0.0340494	2	--
0.0106246	3	--
0.0140563	4	--
0.00975424	5	--
0.00795512	6	--
0.0118077	7	--
0.0244928	8	--
0.00341118	9	--
0.012227	10	--
0.00608312	11	--
0.00790602	12	--

0.0121425	13	--
0.00454708	14	--
0.0238654	15	--
0.0112997	16	--
0.0149835	17	--
0.011705	18	--
0.0167556	19	--
0.0040756	20	--
0.002222854	21	U527/A1 (load) [+pin_cap=0.00103pF]
0.00319582	22	U459/B1 (load) [+pin_cap=0.00112pF]
2e-06	23	U2053/A1 (load) [+pin_cap=0.00075pF]
2e-06	24	U483/A1 (load) [+pin_cap=0.00107pF]
2e-06	25	U534/A1 (load) [+pin_cap=0.00191pF]
2e-06	26	U503/A1 (load) [+pin_cap=0.00189pF]

0.264021	Total
----------	-------

CC in pF	Local Node	Other Node
0.00197175	13	n4250:13
0.00139652	17	n4250:6
0.00218098	3	n2124:2
0.00141374	1 ADDR[1] (driver)	REG_DATA[22]:2
0.000183647	16	n8376:2
0.000175112	6	n13565:2
0.000433937	6	n13599:4
0.000633856	8	n4750ASTipoNet3769:1
0.000346092	8	n1946:1
0.00274598	3	n3117:1
0.00033236	17	n7632:1
0.00067407	12	n11321:1
0.000256377	2	BW0_30_:2
0.000994872	3	n225:2
0.000264488	15	n1785:2
0.00157215	9	n5870ASThfnNet347:5
0.00130062	2	REG_DATA[26]:2
0.00170201	4	n2248ASThfnNet432:10
0.00157756	14	n2248ASThfnNet432:13

0.0201561	Total
-----------	-------

R in Kohm	Left node	Right Node
0.396586	1 ADDR[1] (driver)	2
0.0380318	2	8
0.00700238	3	11
0.054928	3	5
0.00470476	4	14
0.0137696	4	11
0.0112305	5	20
0.00170541	6	16
0.0478066	6	21 U527/A1 (load)
0.0760538	6	22 U459/B1 (load)

0.0794548	6	12
0.0514926	7	9
0.0627434	7	12
0.0104221	8	19
0.00721266	9	10
0.0304842	10	13
0.184718	12	21 U527/A1 (load)
0.293862	12	22 U459/B1 (load)
0.00827976	13	17
0.00528166	14	17
0.0120886	15	18
0.00642048	15	19
0.0658128	16	23 U2053/A1 (load)
0.057019	16	24 U483/A1 (load)
0.0374324	16	25 U534/A1 (load)
0.0401662	16	26 U503/A1 (load)
0.0176871	18	20
0.046151	26 U503/A1 (load)	25 U534/A1 (load)
0.133434	21 U527/A1 (load)	22 U459/B1 (load)
0.0170109	23 U2053/A1 (load)	24 U483/A1 (load)
<hr/>		
1.81899	Total	

Note that only the resistance and the ground capacitance have been scaled by 2. All delay calculation and PrimeTime SI analysis are updated by the new values of these parasitics.

Example 2: Physical Block (Net) Scaling

This example shows how to scale each net in a physical block in your design. Foundries can provide different scaling values for different conditions of a block (process, temperature, or voltage), and the use of parasitics scaling can reflect this variation.

To use this methodology, link the top level of the design (Chip, in this example). You can then switch to the physical block of interest by using the following:

```
current_instance A

set gnets [get_net *] # This produces a collection of
                      # nets for block A.

scale_parasitics -ground_capacitance_factor 1.1 \
-resistance_factor 1.1 -coupling_capacitance_factor 1.1 $gnets

current_instance Chip

...
# Proceed with the analysis
```

Each net in block A, is scaled as specified by the `scale_parasitics` command. Keep in mind that nets that cross hierarchies are also scaled. You can inspect the changes to the parasitics using the `report_annotated_parasitics -list_annotated` command:

Example 3: Scaling Due to Net Separation

PrimeTime has the capability to scale the coupling capacitance due to net separation. By spacing nets, you can reduce the coupling capacitance between the nets. This can be reflected using the `scale_parasitics` command. For example,

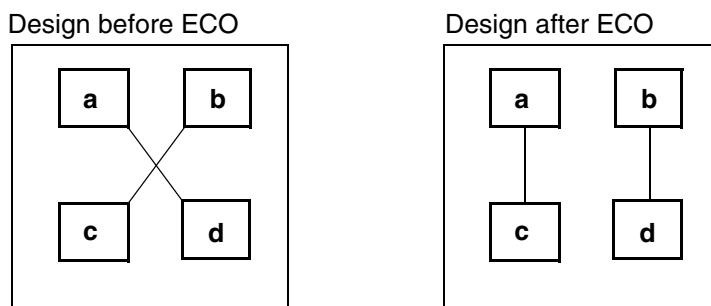
```
pt_shell> scale_parasitics -coupling_capacitance_factor 0.75 \
[get_net Net1]
```

Each of the coupling capacitances for Net1 are scaled. Therefore, in this case both capacitors, from Net1 to Net2, and from Net2 to Net1, are scaled.

Reading Parasitics for Incremental Timing Analysis

PrimeTime can perform an incremental update on a design that is already analyzed when only a small number of nets are affected. The maximum number of nets that can be affected without requiring a full timing update depends upon the total design size. For example, if you have a full-chip annotated file and an engineering change order (ECO) annotated file, and you want to analyze the effects of your ECO, you can run an analysis with the `report_timing` command on a full-chip annotated file, and then repeat the analysis with just the ECO changes applied. See [Figure 11-16](#).

Figure 11-16 Incremental Update Before and After Engineering Change



To set the annotation on most or all nets, enter

```
pt_shell> read_parasitics full_chip.spf
pt_shell> report_timing
```

To override the annotations on a small number of nets, enter

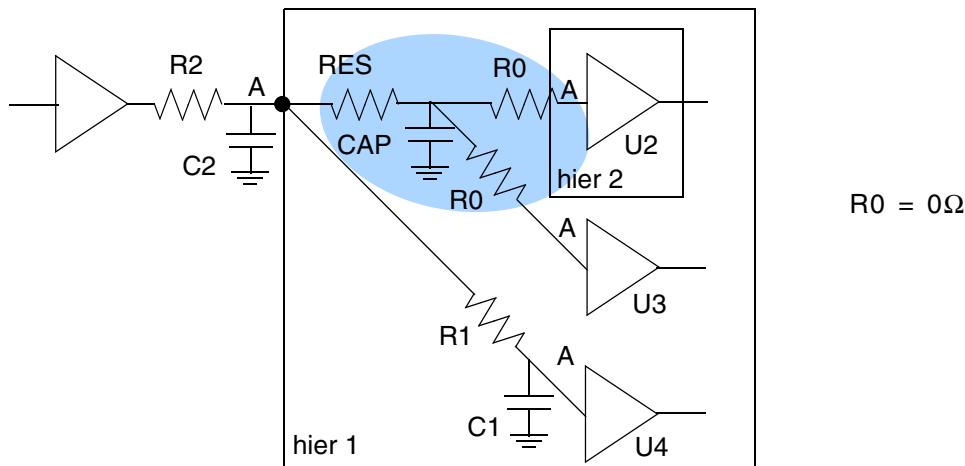
```
pt_shell> read_parasitics eco.spf
pt_shell> report_timing
```

Incomplete Annotated Parasitics

PrimeTime can complete parasitics on nets that have incomplete parasitics. The following conditions apply to RC network completion:

- The nets that have an RC network must be back-annotated from an SPEF file.
- PrimeTime cannot complete effective capacitance calculation for a net that has incomplete parasitics. Instead, PrimeTime reverts to using wire load models for delay calculation on these nets.
- Partial parasitics can be completed on a net only if all the missing segments are between two pins (either boundary pins or leaf pins). As shown in [Figure 11-17](#), the missing segment is completed with a fanout of two.

Figure 11-17 Missing Segment Completed Using the Fanout of Two



The shaded area shown in [Figure 11-17](#) is completed when you use the `complete_net_parasitics` command or the `-complete_with` option of the `read_parasitics` command.

Selecting a Wire Load Model for Incomplete Nets

After the missing segments are identified, PrimeTime selects the wire load model to complete the missing parts of the net as follows:

- The wire load mode is taken from the top-level hierarchy. If the wire load mode does not exist, the default from the main library is taken.
- If the wire load mode is ‘enclosed,’ the wire load for the missing segment is the hierarchy that encloses the missing segment.

- If the wire load mode is ‘top,’ the wire load for the top level of hierarchy is used.
- If the wire load model does not exist on the enclosing hierarchy, the wire load model of the parent hierarchy is taken. If the parent hierarchy does not exist, the wire load model of the parent of the parent is used, this process continues until the top-level hierarchy is reached.
- The default wire load model from the main library is used if the wire load model cannot be obtained from the previous steps.
- Zero resistance and capacitance are used if no wire load model could be obtained using the `-complete_with wlm` option.

Completing Missing Segments on the Net

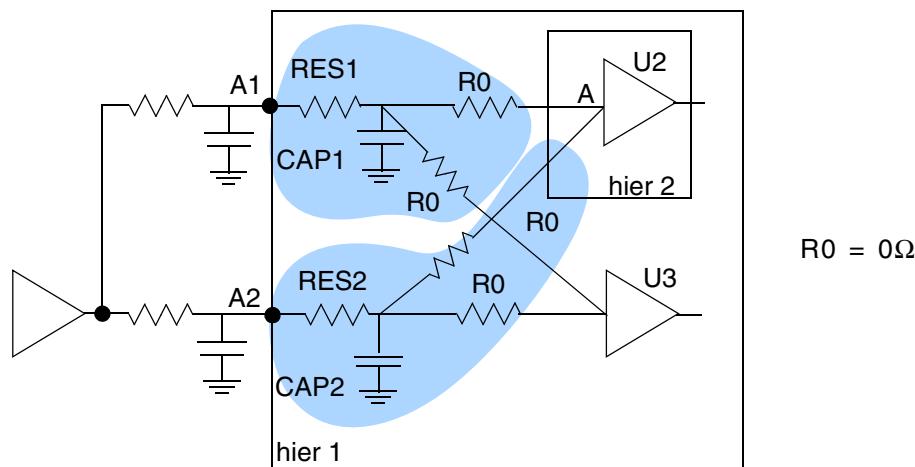
PrimeTime completes multiple pin to pin segments with a single RC pair based on the value of the option `-complete_with`, which can be `zero` or `wlm` (wire load model).

Here is a command sequence that completes the missing segments:

```
pt_shell> read_ddc design.ddc
pt_shell> read_parasitics incomplete_parasitics.spf
pt_shell> complete_net_parasitics -complete_with wlm
```

As shown in [Figure 11-18](#), if the `-complete_with wlm` option is used, the resistance (RES) and capacitance (CAP) values are estimated based on the wire load model. If `-complete_with zero` is used, the values assigned to RES and CAP are zero. In both cases, the resistance that connects the net to the loads is 0 ohms.

Figure 11-18 Multidrive Segments



You can see how networks get completed by comparing the attribute `rc_network` defined for the incomplete nets before and after completion.

After the segment is completed, it cannot be undone. For this reason, if you are using multiple `read_parasitics` commands, be sure to use the `-complete_with` option only with the very last `read_parasitics` command. Otherwise, PrimeTime completes the network before you have finished reading in all of the parasitic data files. To cancel the effects of all `read_parasitics` and `complete_net_parasitics` commands, use the `remove_annotated_parasitics` command.

Do not use the `complete_net_parasitics` command when you have parasitics with errors. However, you can manually fix the SPEF file and reread them. Use the `complete_net_parasitics` command only when the following assumption holds true: The relevant portion of a net's delay is already annotated with detailed parasitics, and you want PrimeTime to complete the remaining, less significant portion of the net with zero or wire load model based resistance and capacitance.

Reporting Annotated Parasitics

Parasitic data files can be large (100 MB or larger) and can contain many parasitics. Use the `report_annotated_parasitics` command after reading an SPEF or RSPP file to verify that the tool back-annotated all cell drivers.

To create a report for annotated parasitic data files and verify that all RC networks are complete, use the `report_annotated_parasitics -check` command, as shown in the following example:

```
*****
Report : annotated_parasitics
      -check
...
*****

```

Pin type	Total	RC pi	RC network	Not Annotated
internal net drive	23649	0	23649	0
design input port	34	0	34	0
	23683	0	23683	0

Removing Annotated Delays, Checks, Transitions, and Parasitics

To remove annotations, use the commands in [Table 11-5](#).

Table 11-5 Commands for Removing Annotations

To remove this type of annotation	Use this command
Delays	<code>remove_annotated_delay</code>
Checks	<code>remove_annotated_check</code>
Transitions	<code>remove_annotated_transitions</code>
Parasitics	<code>remove_annotated_parasitics</code>
All annotations	<code>reset_design</code>

Back-Annotation Order of Precedence

In case of conflict between different types of annotated information, PrimeTime uses the data on each net in the following order of highest to lowest precedence:

1. Annotated delays specified with an SDF file or the `set_annotated_delay` command.
2. Lumped resistance and capacitance values annotated with the `set_resistance` and `set_load` commands.
3. Detailed parasitics annotated with the `read_parasitics` command.
4. Wire load models obtained from the logic library or specified with the `set_wire_load_model` command.

12

Variation

The standard on-chip variation analysis applies different operating conditions to minimum paths and maximum paths. To improve the accuracy of your timing analysis, you can apply the following variation methods:

- [Advanced On-Chip Variation](#)
- [Parametric On-Chip Variation](#)

Advanced On-Chip Variation

Advanced on-chip variation (AOCV) analysis reduces unnecessary pessimism by taking the design methodology and fabrication process variation into account. AOCV determines derating factors based on metrics of path logic depth and the physical distance traversed by a particular path. A longer path that has more gates tends to have less total variation because the random variations from gate to gate tend to cancel each other out. A path that spans a larger physical distance across the chip tends to have larger systematic variations. AOCV is less pessimistic than a traditional OCV analysis, which relies on constant derating factors that do not take path-specific metrics into account.

The AOCV analysis determines path-depth and location-based bounding box metrics to calculate a context-specific AOCV derating factor to apply to a path, replacing the use of a constant derating factor.

AOCV analysis works with all other PrimeTime features and affects all reporting commands. This solution works in both the Standard Delay Format (SDF)-based and the delay calculation based flows. It is compatible with distributed multi-scenario analysis and multicore analysis.

To learn about using AOCV, see

- [AOCV Flow](#)
 - [Specifying the Scope of the AOCV Analysis](#)
 - [Importing AOCV Information](#)
-

AOCV Flow

To enable AOCV, set the `timing_aocvm_enable_analysis` variable to `true`. You also need to provide tables of derating factors that apply to different ranges of path depths and physical path distances. You read these tables into PrimeTime with the `read_aocvm` command. For example, the following table specifies the values used for derating the early delays of cells in paths when the logic depths and physical distances of the paths fall within specified ranges:

```
version:          1.0
object_type:      design
rf_type:         rise fall
delay_type:       cell
derate_type:      early
object_spec:     top
voltage:          1.2
depth:            0 1 2 3
distance:        100 200
table:           0.88 0.94 0.96 0.97 \
                  0.84 0.86 0.88 0.91
```

After you enable AOCV analysis and read in the derating tables, you perform timing analysis in the usual manner using commands such as `update_timing` and `report_timing`. The analysis results reflect the application of varying derating factors based on the logic depths and physical spans of the paths.

The AOCV feature in PrimeTime is integrated into the graph-based and path-based analysis capabilities, which enables a successive refinement strategy like that used in ordinary non-AOCV flows. In AOCV flows, sign-off can be performed using graph-based AOCV, with path-based AOCV available as an optional capability to refine the analysis of any remaining violating paths.

Graph-Based AOCV Analysis

Graph-based AOCV analysis is a fast, design-wide analysis performed during the `update_timing` command. It allows designers to exploit reduced derating pessimism across the entire design to reduce silicon area and improve design performance.

By construction, graph-based analysis always provides a conservative analysis compared to path-based analysis to prevent an optimistic calculation. With graph-based AOCV analysis, PrimeTime computes conservative values of depth and distance metrics. To perform graph-based AOCV analysis, conservative values of path-depth and location-based bounding box are chosen to bound the worst-case path through a cell. For example, in graph-based AOCV analysis, the depth count for a cell does not exceed that of the path of minimum depth that traverses that cell. Otherwise, the analysis might be optimistic for the minimum-depth path.

Graph-based AOCV is a prerequisite for further margin reduction using path-based AOCV analysis on selected critical paths. For most designs, graph-based AOCV is sufficient for sign-off.

Path-Based AOCV Analysis

If violations still remain after completion of graph-based AOCV analysis, you can reduce pessimism and improve the accuracy of results by invoking path-based AOCV analysis, which analyzes each path in isolation from other paths. To do so, use the `report_timing` or `get_timing_paths` command with the `-pba_mode` option set to `path`. For example,

```
pt_shell> report_timing -pba_mode path
```

In graph-based AOCV analysis, PrimeTime calculates the depth and distance for all paths through each timing arc and applies the worst values found for all such paths, whereas in path-based mode, it calculates depth and distance precisely for each individual timing path.

By default, when you select path-based analysis with AOCV, PrimeTime performs both normal path-based analysis (without AOCV) and AOCV path-based analysis. The analysis without AOCV performs path-based slew propagation recalculation, and the AOCV analysis reduces pessimism by reducing the derating for deeper paths and paths that span shorter

physical distances. To save runtime, you can optionally perform only the AOCV portion of path-based analysis and use the worst-slew propagation instead of the recalculated slew propagation for path-based analysis. To do so, set the `pba_derate_only_mode` variable to true.

Specifying the Scope of the AOCV Analysis

To configure AOCV analysis, set the `timing_aocvm_analysis_mode` variable to the one or more of following analysis modes:

- `clock_network_only` – Restricts AOCV derating factors to arcs in the clock network.
 - `combined_launch_capture_depth` – Considers the launch and capture paths together and calculates the combined depth for the entire path. You cannot use this mode with `separate_data_and_clock_metrics` mode.
 - `separate_data_and_clock_metrics` – Separately computes the depths of data and clock paths. You cannot use this mode with `combined_launch_capture_depth` mode.
 - `single_path_metrics` – Calculates single-path metrics for all path objects.
-

Importing AOCV Information

To perform AOCV analysis, you must specify AOCV information in derating tables. To optionally model nonprocess-related effects in an AOCV flow, use guard-band timing deratings.

To learn how to specify AOCV information, see

- [Specifying AOCV Derating Tables](#)
- [File Format for AOCV](#)
- [Specifying Depth Coefficients](#)
- [Guard-Banding in AOCV](#)
- [Incremental Timing Derating](#)
- [Using OCV Deratings in AOCV Analysis](#)

Specifying AOCV Derating Tables

Use the `read_aocvm` command to read AOCV derating tables from a disk file. Derating tables are annotated onto one or more design objects and are applied directly to timing arc delays. The allowed design object classes are hierarchical cells, library cells, and designs.

Use the `write_binary_aocvm` command to create binary encoded AOCV files from ASCII-AOCV files. This command is used to protect sensitive process-related information. The `read_aocvm` command can read binary and compressed binary AOCV files that were created using the `write_binary_aocvm` command. No additional arguments are required to read binary or compressed binary AOCV files.

Internally calculated AOCV derating factors are shown in the derate column of the `report_timing` command if you have specified the `-derate` option.

To display AOCV derating table data, use the `report_aocvm` command. An AOCV derating table imported from a binary or compressed binary file is not visible in the `report_aocvm` output. You can show design objects annotated with early, late, rise, fall, cell, or net derating tables. You can also use this command to determine cells and nets that have been annotated or not annotated with AOCV information.

In a graph-based AOCV analysis, specify a timing arc in the `object_list` of the `report_aocvm` command. The graph-based depth and distance metrics and AOCV derating factors on timing arc objects are displayed.

```
pt_shell> report_aocvm [get_timing_arcs -of_objects [get_cells U1]]
```

If you specify a timing path in the `object_list`, path metrics (distance, launch depth, and capture depth) for that path are displayed.

```
pt_shell> report_aocvm [get_timing_paths -path_type full_clock_expanded \
-pba_mode path]
```

File Format for AOCV

The AOCV file format allows you to specify multiple AOCV derating tables. It supports the following table types:

- One-dimensional tables in either depth or distance
- Two-dimensional tables in both depth and distance

The file format is defined so that you can associate an AOCV table with a group of objects. The objects are selected internally within PrimeTime based using the following definitions:

- `object_type` `design | lib_cell | cell`
- `object_spec` `[patterns]`

In the `object_spec` definition, `patterns` is optional. It specifies the object name and an expression that you want to evaluate based on the attributes of the object.

You can use regular expression matching for the patterns value of the `object_spec` definition. It is the same as the `regexp` Tcl command that you can use for design object gathering commands, such as the `get_cells`, `get_lib_cells`, and `get_designs` commands in PrimeTime.

You can use any of the options of the related object-gathering command in the patterns value. For example, if the `object_type` is `lib_cell`, use any of the arguments of the related `get_lib_cells` command in the patterns value.

PrimeTime can annotate cell and net tables on the design using the `object_type` design. It can also annotate cell tables on library cells using the `lib_cell` object class and annotate cell and net tables on hierarchical cells using the `cell` object class.

Note:

All descriptions are required, unless otherwise stated. To add a comment in any location within the file, use double forward slashes (//).

Table 12-1 AOCV File Format Syntax

Specifier	Description
<code>version</code>	The AOCV version number.
<code>group_name</code>	A group name, which can be used to group AOCV tables together. You apply a group with the <code>set_aocvm_table_group</code> command.
<code>object_type</code>	<code>design</code> <code>lib_cell</code> <code>cell</code>
<code>rf_type</code>	<code>rise</code> <code>fall</code> <code>rise fall</code>
<code>delay_type</code>	<code>cell</code> <code>net</code> <code>cell net</code>
<code>derate_type</code>	<code>early</code> <code>late</code>
<code>path_type</code>	<code>clock</code> <code>data</code> <code>clock data</code>
<code>object_spec</code>	A pattern describing objects affected by the table.
<code>voltage</code>	The supply voltage, in volts. The table applies only to cells operating at that voltage. This is an optional parameter.
<code>depth</code>	A set of M floating-point values, each representing the successive number of logic cells in the path. If no depth values are provided, M=0.
<code>distance</code>	A set of N floating-point values, each representing the physical distance spanned by the path, in nanometers (nm). If no distance values are provided, N=0.

Table 12-1 AOCV File Format Syntax (Continued)

Specifier	Description
table	A set of N x M floating-point values representing the derating factors applied for all combinations of depth and distance. PrimeTime uses linear interpolation to obtain derating factors between the data points provided in the table. It does not extrapolate beyond the most extreme values in the table. The table size is one-dimensional in the following cases: <ul style="list-style-type: none"> • If N=0, the table has M entries. • If M=0, the table has N entries. • If M=0 and N=0, the table has one entry.

When different `object_type` entries with the same `rf_type` and `derate_type` apply to the same cell or net object, the precedence rules that are used are consistent with the `set_timing_derate` command.

Cell arc derating uses the following precedence, from highest to lowest:

1. library cell
2. hierarchical cell
3. design

Net arc derating uses the following precedence, from highest to lowest:

1. hierarchical cell
2. design

Note:

When multiple table entries with the same `object_type`, `rf_type`, and `derate_type` specifications apply to the same cell or net object, the last table entry takes precedence.

If you specify the `voltage` value, but an associated float value is missing, or if you do not specify the voltage value at all, the derating table applies to all voltages. For a multirail cell, do not specify a voltage; instead, specify the most conservative derating factors across all possible input-output voltage combinations of the multirail cell.

The following example of an AOCV file sets an early AOCV table for the whole design, which applies to all cell and nets:

```
version:          1.0
object_type:     design
rf_type:         rise fall
delay_type:      cell net
derate_type:     early
object_spec:    top
depth:           0 1 2 3
distance:        100 200
table:           0.87 0.93 0.95 0.96 \
                  0.83 0.85 0.87 0.90
```

The following example of an AOCV file sets an early AOCV table for the whole design, which applies to all cells with the voltage of 1.2 volts:

```
version:          1.0
object_type:     design
rf_type:         rise fall
delay_type:      cell
derate_type:     early
object_spec:    top
voltage:         1.2
depth:           0 1 2 3
distance:        100 200
table:           0.88 0.94 0.96 0.97 \
                  0.84 0.86 0.88 0.91
```

The following example of an AOCV file sets an early AOCV table for the whole design, which applies to all nets:

```
version          1.0
object_type:     design
rf_type:         rise fall
delay_type:      net
derate_type:     early
object_spec:    top
depth:           0 1 2 3
distance:        100 200
table:           0.88 0.94 0.96 0.97 \
                  0.84 0.86 0.88 0.91
```

The following example of an AOCV file includes the optional `path_type` statement. PrimeTime applies the table data to only the specified path types. To enable separate derating for clock paths and data paths, the `timing_aocvm_analysis_mode` variable must be set to the `separate_data_and_clock_metrics` mode. If the `path_type` statement is

omitted, the table applies to both clock paths and data paths. If this statement is used, the version number specified by the `version` statement must be set to 2.0. For example,

```
version: 2.0
object_type: lib_cell
object_spec: LIB/BUF1X
rf_type: rise fall
delay_type: cell
derate_type: late
path_type: data
depth: 1 2 3 4 5
distance: 500 1000 1500 2000
table: \
1.123 1.090 1.075 1.067 1.062 \
1.124 1.091 1.076 1.068 1.063 \
1.125 1.092 1.077 1.070 1.065 \
1.126 1.094 1.079 1.072 1.067
```

AOCV Table Groups

You can define multiple groups of AOCV tables that can be used for different blocks at different levels of hierarchy. You assign a name to each table group and apply these groups to hierarchical cells using the `set_aocvm_table_group` command. You can apply these tables independently at different levels of hierarchy, without sharing data between different blocks, and maintain the AOCV table settings when a block is used at a higher level of hierarchy.

Use the `group_name` keyword in the AOCV table to specify the group name for a table. For example,

```
version: 3.0
group_name: core_tables
object_type: lib_cell
object_spec: LIB/BUF1X
rf_type: rise fall
delay_type: cell
derate_type: late
path_type: data clock
depth: 1 2 3 4 5
distance: 500 1000
table: \
1.123 1.090 1.075 1.067 1.062 \
1.124 1.091 1.076 1.068 1.063
```

In this example, the `group_name` statement specifies that the table belongs to the `core_tables` group. Multiple tables are typically assigned to a given group. If you use the `group_name` statement, the `version` statement must be set to 3.0 as shown in the example. If no `group_name` statement is used, the table belongs to the default group of AOCV tables. The default group of tables applies to all parts of the design that have not been assigned a named AOCV table group.

To apply an AOCV table group to an instance of a hierarchical cell, use the `set_aocvm_table_group` command as shown in the following example:

```
pt_shell> set_aocvm_table_group core_tables [get_cells H1]
```

This example applies the AOCV `core_tables` table group to the hierarchical block instance named `H1`. The `core_tables` table group applies to all nets fully enclosed in cell instance `H1` as well as all cells in `H1`, including lower-level cells in the hierarchy. Note that the `core_tables` table group does not apply to a net partly inside and partly outside of `H1`.

The `report_aocvm` command reports the numbers of cells annotated with AOCV data for each table group and for the default table group. It also shows the name of each defined table group and the names of corresponding affected hierarchical cells.

To remove the application of a named table group, use the `reset_aocvm_table_group` command. Note that using the `set_aocvm_table_group`, `reset_aocvm_table_group`, or `read_aocvm` command after a timing update triggers a new full timing update.

Specifying Depth Coefficients

You can specify depth coefficients for cells to modify path depth calculations based on cell complexity. A complex cell can consist of an unusually large number of transistors, so it might be desirable to associate that cell with a logic depth count larger than 1, the default logic depth count for a cell arc. For example, a buffer is often implemented as two inverters in series. In that case, the buffer cell could be assigned a derate coefficient of 2.0 for its logic depth count.

Use the `set_aocvm_coefficient` command to set AOCV coefficients on cells, library cells, and library timing arcs. For example,

```
pt_shell> set_aocvm_coefficient 2.0 [get_lib_cells lib1/BUF2]
```

This example sets a coefficient of 2.0 on all instances of the `lib1/BUF2` library cell.

Setting AOCV coefficients is optional. The default coefficient is 1.0.

Guard-Banding in AOCV

Guard-band timing derate allows you to model nonprocess-related effects in an AOCV flow. The `-aocvm_guardband` option is available in the `set_timing_derate`, `report_timing_derate`, and `reset_timing_derate` commands.

Use the `set_timing_derate` command to specify a guard-band derating factor. The `-aocvm_guardband` option is applicable only in an AOCV context. The derating factor that is applied to an arc is a product of the guard-band derate and the AOCV derate. The guard-band derate has no effect outside the context of AOCV.

To report timing derating factors on the current design, use the `report_timing_derate` command. To report only guard-band derating factors, specify the `-aocvm_guardband` option of the `report_timing_derate` command. If you do not specify either the `-variation` or `-aocvm_guardband` option, only the deterministic derating factors are reported. These two options are mutually exclusive.

To reset guard-band derating factors, use the `-aocvm_guardband` option of the `reset_timing_derate` command.

Incremental Timing Derating

Incremental timing derating enables you to fine-tune the timing derating factors on objects such as cells or nets. To specify that the derating factor is an incremental timing derating, use the `set_timing_derate` command with the `-increment` option. You cannot incrementally apply guard-band timing deratings; therefore, the `-increment` and `-aocvm_guardband` options of the `set_timing_derate` command are mutually exclusive.

Incremental timing deratings adhere to the same precedence and override rules as regular timing deratings. For more information about the precedence rules, see [File Format for AOCV](#).

A single incremental derating factor per object, such as a cell or net, is calculated. If multiple incremental deratings are applied to the same object, the last value overrides all other values. If no timing derating factor exists, the value of 0.0 is used for incremental deratings, and 1.0 is used for regular deratings. The incremental timing derating factor is added to the regular timing derating factor. If the final resulting derating factor is less than 0.0, the negative derating factor is converted to 0.0 to avoid negative delays, and an error message is issued.

Here are some examples of how incremental timing deratings are calculated.

- In an AOCV analysis, you might have a u1/u252 cell that has a regular late derating of 1.082 and early derating of 0.924.

```
set_app_var timing_aocvm_enable_analysis true
set_timing_derate -increment -late 0.03 [get_cells u1/u252]
set_timing_derate -increment -early -0.03 [get_cells u1/u252]
```

The final late derating factor on cell u1/u252 = $1.082 + 0.03 = 1.112$, and the final early derating factor on cell u1/u252 = $0.924 + (-0.03) = 0.894$.

- In an OCV analysis, you might have the following values:

```
set_timing_derate -late 1.09
set_timing_derate -early 0.91
set_timing_derate -late 1.11 [get_cells u1/u252]
set_timing_derate -increment -late 0.04 [get_cells u1/u252]
set_timing_derate -increment -early -0.02 [get_cells u1/u252]
set_timing_derate -increment -early -0.03 [get_cells u1/u252]
```

The final late derating factor on cell u1/u252 = $1.11 + 0.04 = 1.15$, and the final early derating factor on cell u1/u252 = $0.91 + (-0.03) = 0.88$.

To reset only incremental derating factors for the specified object, use the `reset_timing_derate` command with the `-increment` option.

To report incremental timing derating factors, use the `report_timing_derate` command with the `-increment` option. In the following example, incremental derating factors have been set globally on the design; therefore, only global incremental derating factors are reported.

```
pt_shell> set_timing_derate -data -increment -early -0.02
pt_shell> set_timing_derate -data -increment -late 0.06
pt_shell> set_timing_derate -clock -increment -early -0.05
pt_shell> set_timing_derate -clock -increment -late 0.10
pt_shell> report_timing_derate -increment

*****
Report : timing derate
    -scalar -increment
...
*****
----- Clock -----          ----- Data -----
      Rise      Fall          Rise      Fall
      Early     Late     Early     Late
----- Early   Late   Early   Late -----
design: simple_path
Net delay static      -0.05   0.10  -0.05   0.10      -0.02   0.06  -0.02   0.06
Net delay dynamic     -0.05   0.10  -0.05   0.10      -0.02   0.06  -0.02   0.06
Cell delay            -0.05   0.10  -0.05   0.10      -0.02   0.06  -0.02   0.06
Cell check           --     --     --     --      --     --     --     --
```

Using OCV Deratings in AOCV Analysis

PrimeTime supports a unified framework so that AOCV can use OCV deratings under certain conditions. By default, both OCV and AOCV deratings are considered for a given object. If derating factors are at the same level, the AOCV value is selected over the OCV value. To ensure the correct derating factor is applied for a given design object when AOCV is enabled, cell arc derating uses the following order of precedence, from highest to lowest priority:

1. OCV leaf-level cell
2. AOCV library cell
3. OCV library cell
4. AOCV hierarchical cell
5. OCV hierarchical cell

6. AOCV design

7. OCV design

To override the default behavior and completely ignore the OCV derating values during AOCV analysis, set the `timing_aocvm_ocv_precedence_compatibility` variable to `true`. When you do this, PrimeTime uses the precedence rules specified by the `set_timing_derate` command, and cell arc derating uses the following order of precedence, from highest to lowest priority:

1. AOCV library cell
2. AOCV hierarchical cell
3. AOCV design

Querying AOCV Deratings on Timing Paths

To obtain detailed information about AOCV deratings that are applied to a timing path, you can query the following AOCV attributes:

- `aocvm_coefficient`
- `applied_derate`
- `depth`
- `derate_factor_depth_distance`
- `distance`
- `guardband`
- `incremental`

Although AOCV deratings are based on timing arcs, these attributes are associated with timing point objects where the arcs end. You can use these attributes for both graph-based and path-based analysis.

Before using the AOCV attributes, you must do one of the following:

- Enable graph-based AOCV analysis by setting the `timing_aocvm_enable_analysis` variable to `true`.
- Run path-based AOCV analysis by using the `get_timing_paths` command with the `-pba_mode` option.

To query the AOCV attributes, you can use the following script example, which iterates over the timing paths:

```
set path_list [get_timing_paths ...]
foreach_in_collection path $path_list {
    foreach_in_collection point [get_attribute $path points] {
        echo [format "Derate value: %f" [get_attr $point applied_derate]]
    }
}
```

Parametric On-Chip Variation

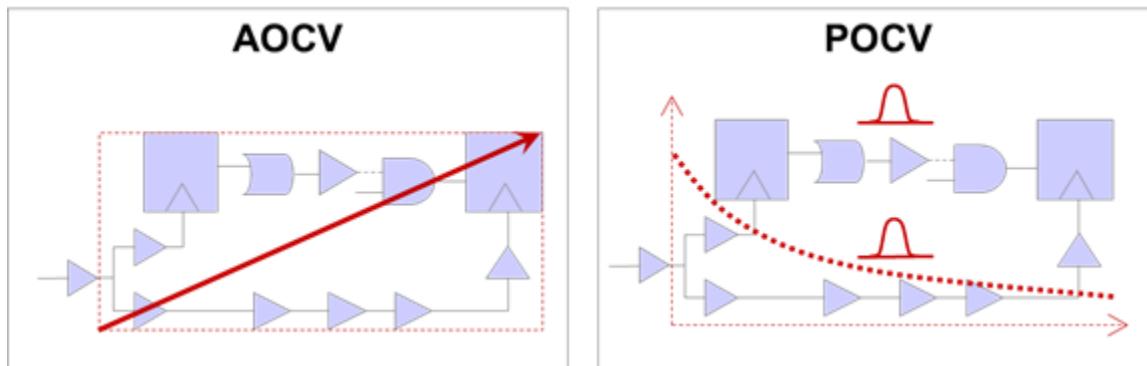
PrimeTime ADV parametric on-chip variation (POCV) models the delay of an instance as a function of a variable that is specific to the instance. That is, the instance delay is parameterized as a function of the unique delay variable for the instance. POCV provides the following:

- Statistical single-parameter derating for random variations
- Single input format and characterization source for both AOCV and POCV table data
- Nonstatistical timing reports
- Limited statistical reporting (mean, sigma) for timing paths
- Compatibility with existing PrimeTime functionality

[Figure 12-1](#) shows a comparison of advanced on-chip variation (AOCV) and POCV. Compared to AOCV, POCV provides

- Reduced pessimism gap between graph-based analysis and path-based analysis
- Less overhead for incremental timing analysis

Figure 12-1 Comparison of AOCV and POCV



To learn about performing POCV analysis, see the following topics steps:

- [Summary of Variables and Commands for Parametric On-Chip Variation](#)
- [Preparing Input Data for Parametric On-Chip Variation](#)
- [Importing a SPEF File With Physical Locations](#)
- [Enabling Parametric On-Chip Variation Analysis](#)
- [Loading the Parametric On-Chip Variation Input Data](#)
- [Specifying Guard Banding](#)
- [Scaling the Parametric On-Chip Variation Coefficient](#)
- [Enabling Constraint and Slew Variation](#)
- [Reporting Parametric On-Chip Variation Results](#)
- [Querying POCV Slack and Arrival Attributes](#)

Summary of Variables and Commands for Parametric On-Chip Variation

[Table 12-2](#) and [Table 12-3](#) summarize the variables and commands for POCV.

Table 12-2 Variables for POCV Analysis

Variable	Description
timing_enable_constraint_variation	Enables constraint variation for setup and hold constraints
timing_enable_slew_variation	Enables slew variation for cell delays
timing_pocvm_corner_sigma	Specifies the corner sigma for POCV analysis
timing_pocvm_enable_analysis	Enables POCV analysis
timing_pocvm_report_sigma	Specifies the sigma for reporting

Table 12-3 Commands for POCV Analysis

Command	Description
<code>read_aocvm pocvm_file</code>	Reads POCV tables
<code>report_delay_calculation -derate</code>	Reports details about deratings for delay calculation
<code>report_ocvm -type pocvm</code>	Displays POCV information, including POCV coefficient and distance-based derating table data,
<code>report_timing -derate</code>	Reports POCV information in timing report
<code>report_timing_derate -pocvm_coefficient_scale_factor</code>	Reports POCV scaling
<code>report_timing_derate -pocvm_guardband</code>	Reports POCV guard banding
<code>reset_timing_derate -pocvm_coefficient_scale_factor</code>	Removes POCV scaling
<code>reset_timing_derate -pocvm_guardband</code>	Removes POCV guard banding
<code>set_timing_derate -pocvm_coefficient_scale_factor</code>	Specifies POCV coefficient scaling
<code>set_timing_derate -pocvm_guardband</code>	Specifies POCV guard banding

Preparing Input Data for Parametric On-Chip Variation

To use parametric on-chip variation (POCV), you need one of the following types of input data:

- [POCV Single Coefficient Specified in a Side File](#)
- [POCV Slew-Load Table in Liberty Variation Format](#)

If the tool reads both types of data, the POCV single coefficient in a side file has higher precedence and overrides the POCV slew-load table in the library.

POCV Single Coefficient Specified in a Side File

The AOCV table format version 4.0 has an extended format that specifies POCV information with the following fields:

- `ocvm_type: aocvm | pocvm`
This field specifies the OCV methodology type.
- `coefficient: sigma_value`
This field specifies the random variation coefficient (sigma).

If you specify `ocvm_type: pocvm`,

- You cannot specify the `depth` field
- The `coefficient` and `distance` fields are mutually exclusive

You need to specify different tables for POCV coefficients (random variation) and distance-based variation. [Table 12-4](#) shows examples.

Table 12-4 Coefficient and Distance-Based POCV Tables

Variation type	POCV applies to	Example
Coefficient	Library cells	<pre>version : 4.0 ocvm_type : pocvm object_type: lib_cell rf_type : rise fall delay_type : cell derate_type: early object_spec: lib28nm/invx* coefficient: 0.05</pre>
Distance-based	Design level	<pre>version : 4.0 ocvm_type : pocvm object_type: design rf_type : rise fall delay_type : cell derate_type: early object_spec: distance : 1 10 50 100 500 table : 0.9116 0.9035 0.8917 0.8718</pre>

To extract the POCV coefficient for a library cell from Monte-Carlo HSPICE simulation, use the following equation:

$$\text{POCV coefficient} = \frac{\sigma(\text{delay variation})}{\mu(\text{nominal delay})}$$

POCV data generation is usually faster than AOCV data generation. Monte-Carlo simulation needs to be set up on a long chain of cells for AOCV, while it is limited to only a single or a few stages for POCV data generation.

POCV Slew-Load Table in Liberty Variation Format

PrimeTime can read a POCV slew-load table for each delay timing arc in Liberty variation format (LVF). [Example 12-1](#) shows a POCV slew-load table in the library.

Example 12-1 POCV LVF Slew-Load Table

```
ocv_sigma_cell_rise ("delay_template_7x7") {
    sigma_type : "early";
    index_1("0.0088000, 0.0264000, 0.0608000, 0.1296000, 0.2672000, 0.5424000,
1.0936000");
    index_2("0.0010000, 0.0024000, 0.0052000, 0.0108000, 0.0221000, 0.0445000,
0.0895000");
    values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.012922", \
"0.000651, 0.000901, 0.001303, 0.002081, 0.003678, 0.006818, 0.013144", \
"0.000840, 0.001166, 0.001714, 0.002558, 0.004112, 0.007249, 0.013529", \
"0.001115, 0.001520, 0.002193, 0.003317, 0.005087, 0.008153, 0.014445", \
"0.001521, 0.002033, 0.002883, 0.004242, 0.006522, 0.010072, 0.016258", \
"0.002155, 0.002793, 0.003853, 0.005563, 0.008424, 0.012955, 0.020171", \
"0.003204, 0.003977, 0.005321, 0.007515, 0.010960, 0.016582, 0.025786");
}

ocv_sigma_cell_rise ("delay_template_7x7") {
    sigma_type : "late";
    index_1("0.0088000, 0.0264000, 0.0608000, 0.1296000, 0.2672000, 0.5424000,
1.0936000");
    index_2("0.0010000, 0.0024000, 0.0052000, 0.0108000, 0.0221000, 0.0445000,
0.0895000");
    values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.012922", \
"0.000651, 0.000901, 0.001303, 0.002081, 0.003678, 0.006818, 0.013144", \
"0.000840, 0.001166, 0.001714, 0.002558, 0.004112, 0.007249, 0.013529", \
"0.001115, 0.001520, 0.002193, 0.003317, 0.005087, 0.008153, 0.014445", \
"0.001521, 0.002033, 0.002883, 0.004242, 0.006522, 0.010072, 0.016258", \
"0.002155, 0.002793, 0.003853, 0.005563, 0.008424, 0.012955, 0.020171", \
"0.003204, 0.003977, 0.005321, 0.007515, 0.010960, 0.016582, 0.025786");
}
```

LVF removes the limitation in POCV side file with single coefficient where POCV data was generated from a particular slew and load for each cell. LVF improves the overall accuracy

of POCV analysis. Currently, LVF supports only tables on delay timing arcs; constraint timing arcs are not supported.

Load the library with the POCV LVF data into PrimeTime before linking the design. PrimeTime requires the existence of POCV LVF data in the timing library. You cannot read POCV LVF from a separate file. The unit of POCV LVF data is the time unit of the library.

PrimeTime can also read distance-based derating in the Liberty variation format, as shown in [Example 12-2](#). While linking the design, PrimeTime automatically reads the distance-based derating table if it exists in the library.

Example 12-2 POCV LVF Distance-Based Derating

```
ocv_table_template("aocvm_distance_template") {
    variable_1 : path_distance;
    index_1 ("0, 10, 100, 1000");
}
...
ocv_derate(aocvm_distance_design){
    ocv_derate_factors(aocvm_distance_template) {
        rf_type : rise_and_fall;
        derate_type : late;
        path_type : clock_and_data;
        values ( "1.1, 1.2, 1.3, 1.4" );
    }
    ocv_derate_factors(aocvm_distance_template) {
        rf_type : rise_and_fall;
        derate_type : early;
        path_type : clock_and_data;
        values ( "0.9, 0.8, 0.7, 0.6" );
    }
}
default_ocv_derate_distance_group : aocvm_distance_design;
```

Importing a SPEF File With Physical Locations

If you use distance-based derating tables for POCV analysis, the tool needs coordinates to calculate the path distance. However, you do not need these coordinates to calculate the path depth.

To read the coordinates of nodes of nets, pins, and ports from a Standard Parasitic Exchange Format (SPEF) or Synopsys Binary Parasitics Format (SBPF) file,

1. Set this variable:

```
pt_shell> set_app_var read_parasitics_load_locations true
```

2. Read the parasitic data:

```
pt_shell> read_parasitics ...
```

Enabling Parametric On-Chip Variation Analysis

To enable graph-based POCV analysis, set this variable:

```
pt_shell> set_app_var timing_pocvm_enable_analysis true
```

PrimeTime performs graph-based POCV timing updates automatically as part of the update_timing command. By default, POCV analysis is performed at 3 sigma. To change the default, set the timing_pocvm_corner_sigma variable to a value larger than 3. For example:

```
pt_shell> set_app_var timing_pocvm_corner_sigma 4
```

Loading the Parametric On-Chip Variation Input Data

To load POCV single coefficient information or POCV distance-based derating table from a text file, use the `read_aocvm` command (similar to loading AOCV tables in the AOCV flow):

```
pt_shell> read_aocvm pocv_coefficient_file_name  
pt_shell> read_aocvm pocv_distance_based_derating_file_name
```

You must specify the coefficient or derating factors in a table using the Synopsys AOCV file format version 4.0 or later. (See [Preparing Input Data for Parametric On-Chip Variation](#).) PrimeTime applies the POCV coefficient values in the file directly to the instance delays. If the POCV tables have incorrect syntax, the tool issues an error message and does not annotate the data.

PrimeTime annotates the POCV tables onto one or more design objects. The design objects that can be annotated are library cells, hierarchical cells, and designs. For a cell, the precedence of data tables, from the lowest to highest, is the design POCV table, hierarchical cell POCV table, and library cell POCV table. For a net, the precedence from lowest to highest is the design POCV table and hierarchical cell POCV table.

Loading the library with POCV LVF data is similar to loading the regular timing library; specify the library with the `search_path` and `link` commands. PrimeTime automatically reads the POCV information from the library while linking the design.

You can apply both the library with POCV and the side file with POCV single coefficient in the same PrimeTime run. The delay variation is annotated accordingly depending on which POCV format is applied on the cell. If the library with POCV LVF and the side file with POCV single coefficient are applied on the same library cell, the POCV single coefficient takes precedence. It overrides POCV LVF for that particular library cell.

Specifying Guard Banding

In the POCV flow, similar to the AOCV flow, you can use guard banding to model non-process-related effects. To perform guard banding in POCV, specify the `set_timing_derate` command with the `-pocvm_guardband` option.

For example, the following commands apply a five percent guard band POCV derating on both early and late mode:

```
pt_shell> set_timing_derate -cell_delay \
           -pocvm_guardband -early 0.95
pt_shell> set_timing_derate -cell_delay \
           -pocvm_guardband -late 1.05
```

The POCV guard band derating factor is applied on both the nominal cell delay (mean) and cell delay variation (sigma).

To report only the guard band derating factors, use the `report_timing_derate -pocvm_guardband` command. To reset guard band derating factors, use the `reset_timing_derate -pocvm_guardband` command.

Scaling the Parametric On-Chip Variation Coefficient

In addition to guard banding, you can scale only the variation of the cell delay without changing the coefficient in POCV tables. To do this, use the `set_timing_derate -pocvm_coefficient_scale_factor` command.

For example, the following command scales the POCV coefficient by three percent for both early and late modes:

```
pt_shell> set_timing_derate -cell_delay \
           -pocvm_coefficient_scale_factor -early 0.97
pt_shell> set_timing_derate -cell_delay \
           -pocvm_coefficient_scale_factor -late 1.03
```

The command applies the POCV coefficient scaling factor to only the variation of cell delay (sigma).

To report the POCV coefficient scaling factors, use the `report_timing_derate -pocvm_coefficient_scale_factor` command. To reset the POCV coefficient scaling factors, use the `reset_timing_derate -pocvm_coefficient_scale_factor` command.

Enabling Constraint and Slew Variation

You can improve the accuracy of POCV analysis with the following optional settings:

- To enable variation for setup and hold constraints, set the `timing_enable_constraint_variation` variable to `true`.
- To enable slew variation for cell delays, set the `timing_enable_slew_variation` variable to `true`.

The tool reads the constraint and slew variation data provided in the Liberty Variation Format (LVF) in the `.lib` or `.db` file.

To report the constraint and slew variation, use the `report_timing -variation` or `report_ocvm` command.

Reporting Parametric On-Chip Variation Results

To see the results of parametric on-chip variation (POCV) analysis,

1. [Report the POCV Coefficient and Deratings](#)
2. [Report the POCV Analysis Results](#)

Report the POCV Coefficient and Deratings

To display POCV information, including POCV coefficient and distance-based derating table data, use the `report_ocvm -type pocvm` command. This command reports design objects annotated with early, late, rise, fall, cell, net coefficient or derating tables and also the annotation information for leaf cells and nets, as shown in [Example 12-3](#).

```
pt_shell> report_ocvm -type pocvm -cell_delay -list_not_annotated \
-coefficient
```

Example 12-3 Output of report_ocvm -type pocvm Command With POCV Side File

```
pt_shell> report_ocvm -type pocvm [get_cell I]

*****
POCV Table Set : *Default*
*****
POCV coefficient: 0.0500
Name Object Process Voltage Sense Path Delay Inherited
-----
I cell early * rise clock cell lib1/INV1
I cell early * fall clock cell lib1/INV1
I cell early * rise data cell lib1/INV1
I cell early * fall data cell lib1/INV1

*****
POCV Table Set : *Default*
*****
POCV coefficient: 0.0500
Name Object Process Voltage Sense Path Delay Inherited
-----
I cell late * rise clock cell lib1/INV1
I cell late * fall clock cell lib1/INV1
I cell late * rise data cell lib1/INV1
I cell late * fall data cell lib1/INV1
```

To report all cells missing POCV coefficient and distance-based derating data, use this command:

```
pt_shell> report_ocvm -type pocvm -cell_delay -list_not_annotated
```

To report all cells only missing POCV coefficient data (when you did not apply distance-based derating table in POCV analysis), use this command:

```
pt_shell> report_ocvm -type pocvm -cell_delay \
    -list_not_annotated -coefficient
```

The `report_ocvm` command does not require `update_timing`. You can do it before `update_timing` to check if there are any cells missing POCV coefficient or distance-based derating table.

For POCV LVF, since the slew-load table is defined on library timing arc, you need to specify library timing arc on the object list to make it work. The `report_ocvm` command displays all slew-load tables associated with the specified library timing arc, as shown in [Example 12-4](#).

Example 12-4 Output of report_ocvm -type pocvm Command With POCV LVF

```
pt_shell> report_ocvm -type pocvm [get_lib_timing_arcs \
    -from */INVD1/I -to */INVD1/ZN]

min rise table
Sense / Type: negative_unate
| Slew
Load | 0.0010000 0.0024000 0.0052000 0.0108000 0.0221000 0.0445000 0.0895000
-----+-----
0.0088000 | 0.0004760 0.0006770 0.0010750 0.0018700 0.0034380 0.0066260 0.0129220
0.0264000 | 0.0006510 0.0009010 0.0013030 0.0020810 0.0036780 0.0068180 0.0131440
0.0608000 | 0.0008400 0.0011660 0.0017140 0.0025580 0.0041120 0.0072490 0.0135290
0.1296000 | 0.0011150 0.0015200 0.0021930 0.0033170 0.0050870 0.0081530 0.0144450
0.2672000 | 0.0015210 0.0020330 0.0028830 0.0042420 0.0065220 0.0100720 0.0162580
0.5424000 | 0.0021550 0.0027930 0.0038530 0.0055630 0.0084240 0.0129550 0.0201710
1.0936000 | 0.0032040 0.0039770 0.0053210 0.0075150 0.0109600 0.0165820 0.0257860

min fall table
Sense / Type: negative_unate
| Slew
Load | 0.0010000 0.0024000 0.0052000 0.0108000 0.0221000 0.0445000 0.0895000
-----+-----
0.0088000 | 0.0003780 0.0005140 0.0007840 0.0013200 0.0024020 0.0045370 0.0088430
0.0264000 | 0.0004950 0.0006980 0.0010150 0.0015460 0.0026210 0.0047660 0.0090610
0.0608000 | 0.0005730 0.0008580 0.0012960 0.0019850 0.0030780 0.0052090 0.0095170
0.1296000 | 0.0006060 0.0009720 0.0015550 0.0024890 0.0039210 0.0061280 0.0103820
0.2672000 | 0.0005040 0.0009800 0.0017400 0.0029410 0.0048280 0.0077760 0.0122220
0.5424000 | 0.0001620 0.0007670 0.0017220 0.0033180 0.0057850 0.0095850 0.0154820
1.0936000 | 0.0006780 0.0000330 0.0012030 0.0032310 0.0064090 0.0114070 0.0189420
```

In addition to the data, each table displays the min (early) or max (late) analysis mode and the negative_unate or positive_unate sense type. If there are multiple library timing arcs with different input conditions, the sdf_cond is also displayed.

Report the POCV Analysis Results

POCV analysis works with all reporting commands in PrimeTime. For reporting, by default PrimeTime reports the timing at 3-sigma value. To change it, set the timing_pocvm_report_sigma variable to different value; the default is 3. Changing this variable does not trigger update_timing. It only affects reporting.

For example, to make POCV analysis more pessimistic, you can change the variable to 4.

```
pt_shell> set_app_var timing_pocvm_report_sigma 4
```

To check the slack without POCV, set the variable to 0. It shows the slack at 0 sigma (no POCV) without triggering update_timing.

```
pt_shell> set_app_var timing_pocvm_report_sigma 0
```

To display the variation in the timing report for POCV analysis, use the `report_timing -variation` command. For example:

Figure 12-2 Output of the report_timing -variation Command

pt_shell> report_timing -variation -significant_digits 5 -nosplit					
Point	Derate	Mean	Sensit	Incr	Path
Startpoint: i (input port clocked by CLK)					Nominal cell delay without POCV or other variation
Endpoint: o (output port clocked by CLK)					
Path Group: CLK					
Path Type: max					
Point	Derate	Mean	Sensit	Incr	Path
clock CLK (rise edge)				0.00000	0.00000
clock network delay (ideal)				0.00000	0.00000
input external delay				0.00000	0.00000 r
i (in)		0.00000	0.00000	0.00000	0.00000 r
I/I (INV1)	1.00000	0.00000	0.00000	0.00000	0.00000 r
I/ZN (INV1)	1.02000	0.00975	0.00049	0.01121 &	0.01121 f
o (out)		1.00000	0.00007	0.00000	0.00007 & 0.01128 f
data arrival time					0.01128
clock CLK (rise edge)				1.00000	1.00000
clock network delay (ideal)				0.00000	1.00000
output external delay				0.00000	1.00000
data required time		1.00000	0.00000		1.00000
data required time		1.00000	0.00000		1.00000
data arrival time		-0.00982	0.00049		-0.01128
slack (MET)		0.99018	0.00049		0.98872

The Incr column is calculated from Mean and Sensit columns with the following equation:

$$\text{Incr} = \text{Mean} \pm K * \frac{(\text{Sensit}_{\text{cell}})^2}{\text{Sensit}_{\text{slack}}}$$

where K is specified by the `timing_pocvm_report_sigma` variable; the default is 3.

To show the details how PrimeTime calculates the final derating factor from all derating factors applied in POCV analysis, use the `report_delay_calculation -derate` command. It reports in details how the final deratings are derived for cell delay and cell delay sigma as shown in [Example 12-5](#).

Example 12-5 Output of report_delay_calculation -derate Command With POCV Side File

```
pt_shell> report_delay_calculation -from I/I -to I/ZN -derate
...
Advanced driver-modeling used for rise and fall.
      Rise          Fall
-----
Input transition time = 0.011600  0.011600 (in library unit)
Effective capacitance = 0.002000  0.002000 (in pF)
Effective capacitance = 0.002000  0.002000 (in library unit)
Output transition time = 0.007237  0.006342 (in library unit)
Cell delay           = 0.008886  0.009559 (in library unit)

POCVM coefficient     = 0.050000  0.050000
POCVM coef scale factor = 1.000000 1.000000
POCVM distance derate = 1.000000 1.000000
POCVM guardband       = 1.020000 1.020000
Incremental derate     = 0.000000 0.000000
Cell delay derated    = 0.009064  0.009750 (in library unit)
Cell delay sigma       = 0.000453  0.000488 (in library unit)

Cell delay derated = "Cell delay" *
( "POCVM guardband" * "POCVM distance derate" + "Incremental derate" )
Cell delay sigma     = "Cell delay" *
( "POCVM guardband" * "POCVM coefficient" * "POCVM coef scale factor" )
```

In [Example 12-5](#), `report_delay_calculation -derate` command displays the equations how the cell delay and cell delay variation are derated by applying POCV guard banding, POCV distance-based derating, POCV coefficient, POCV coefficient scaling factor and incremental derate.

If you are using the POCV LVF instead of the POCV single coefficient, the format of `report_delay_calculation -derate` command changes slightly since the unit of POCV data in LVF is in time units. [Example 12-6](#) shows the output of `report_delay_calculation -derate` when the POCV slew-load table is applied.

Example 12-6 Output of the report_delay_calculation -derate command With POCV LVF

```
pt_shell> report_delay_calculation -from I/I -to I/ZN -derate
...
Advanced driver-modeling used for rise and fall.
          Rise      Fall
-----
Input transition time = 0.011600  0.011600 (in library unit)
Effective capacitance = 0.002000  0.002000 (in pF)
Effective capacitance = 0.002000  0.002000 (in library unit)
Drive resistance     = 0.001000  0.001000 (in Kohm)
Output transition time = 0.016620  0.011003 (in library unit)
Cell delay           = 0.013041  0.010004 (in library unit)

POCVM delay sigma    = 0.000652  0.000500
POCVM coef scale factor = 1.000000 1.000000
POCVM distance derate = 1.000000 1.000000
POCVM guardband      = 0.980000  0.980000
Incremental derate    = 0.000000  0.000000
Cell delay derated    = 0.012780  0.009804 (in library unit)
Cell delay sigma       = 0.000639  0.000490 (in library unit)

Cell delay derated = "Cell delay" *
( "POCVM guardband" * "POCVM distance derate" + "Incremental derate" )
Cell delay sigma = "POCVM delay sigma" *
( "POCVM guardband" * "POCVM coef scale factor" )
```

Querying POCV Slack and Arrival Attributes

POCV analysis uses slack and arrival attributes on pins, timing paths, and timing points to cover the mean and variation of the cell delays. [Table 12-5](#) lists these attributes.

Table 12-5 POCV Slack and Arrival Attributes

Attribute	Object class
max_fall_variation_arrival	pin
max_fall_variation_slack	pin
max_rise_variation_arrival	pin
max_rise_variation_slack	pin
min_fall_variation_arrival	pin
min_fall_variation_slack	pin
min_rise_variation_arrival	pin

Table 12-5 POCV Slack and Arrival Attributes (Continued)

Attribute	Object class
min_rise_variation_slack	pin
variation_arrival	timing_path, timing_point
variation_slack	timing_path, timing_point

POCV slack and arrival attributes are statistical quantities, i.e. they have mean and standard deviation. If you directly query the POCV slack and arrival attributes, the tool returns an empty list. You need to query the mean or std_dev (standard deviation) attribute of the POCV slack and arrival attributes, as shown in [Example 12-7](#).

Example 12-7 Querying POCV Slack and Arrival Attribute

```
pt_shell> get_attribute [get_pin I/ZN] max_fall_variation_slack
{}
pt_shell> get_attribute [get_attribute [get_pin I/ZN] \
max_fall_variation_slack] mean
0.990180
pt_shell> get_attribute [get_attribute [get_pin I/ZN] \
max_fall_variation_slack] std_dev
0.000488
pt_shell> set path [get_timing_paths -fall_through I/ZN]
_se196
pt_shell> get_attribute $path variation_slack
{}
pt_shell> get_attribute [get_attribute $path variation_slack] mean
0.990180
pt_shell> get_attribute [get_attribute $path variation_slack] std_dev
0.000488
```

13

Multivoltage Design Flow

PrimeTime supports the use of the IEEE 1801 Unified Power Format (UPF) Standard for specifying the multivoltage features of the design. PrimeTime correctly analyzes the timing of the design in the presence of multivoltage supplies and voltage values set on specific supply nets with the `set_voltage` command.

To learn about the using the multivoltage design flow, see

- [Multivoltage Analysis](#)
- [UPF Commands](#)
- [UPF Supply Sets](#)
- [UPF Supply Set Handles](#)
- [Virtual Power Network](#)
- [Setting Voltage and Temperature](#)
- [Library Support for Multivoltage Analysis](#)
- [Multivoltage Reporting and Checking](#)
- [Library PG Tcl File](#)
- [Golden UPF Flow](#)
- [Power Domain Mode of Version Z-2007.06](#)

- Multivoltage Method Before Version Z-2007.06
- Multiple Supply Voltage Analysis

Multivoltage Analysis

PrimeTime supports timing analysis with different power supply voltages on different cells. PrimeTime calculates delays and slews, PrimeTime SI calculates crosstalk delay and noise effects, and PrimeTime PX calculates power consumption based on the actual supply voltage on each supply pin of each cell.

To perform a multivoltage analysis, you provide information about the voltage on each power supply pin of each cell in the following ways:

- Specify the power intent of the design using UPF commands and related PrimeTime commands, including `create_power_domain`, `create_supply_net`, `create_supply_set`, `create_supply_port`, `connect_supply_net`, and `set_voltage`. This method was introduced in PrimeTime version A-2007.12 and is the preferred method.
- Use commands in the “power domain mode” of PrimeTime version Z-2007.06. The commands in this mode are `create_power_net_info`, `create_power_domain`, `connect_power_domain`, `connect_power_net_info`, `set_operating_conditions -object_list`, and `set_voltage`. To use the power domain mode from version Z-2007.06, you must set the `power_domains_compatibility` variable to `true`, which disables the other two methods. For more information, see [Power Domain Mode of Version Z-2007.06](#).

Note:

In the power domain mode, the `create_power_domain` command has a different effect from the command of the same name executed in the default UPF mode.

- Set the supply voltages on specific blocks by using either the `set_operating_conditions -object_list` or `set_rail_voltage` command or both commands. Using the `set_rail_voltage` command requires a PrimeTime SI license. This method was available before PrimeTime version Z-2007.06 and is still supported. For more information about using this method, see [Multivoltage Method Before Version Z-2007.06](#).

The three methods are mutually exclusive. By default, if you use any UPF commands, UPF is the voltage specification method for the PrimeTime session. If you do not use any UPF commands in the session, you can use the method available before PrimeTime version Z-2007.06 by using either the `set_operating_conditions -object_list` or `set_rail_voltage` command or both commands.

Most of the information presented here applies to the recommended UPF flow. For information that is specific to the two older flows, see

- [Power Domain Mode of Version Z-2007.06](#)
- [Multivoltage Method Before Version Z-2007.06](#)

Unless you are using the older method of specifying all supply voltages using `set_operating_conditions` or `set_voltage`, the library must contain power and ground (PG) pin information for each cell. The Liberty syntax for specifying PG pin information in the library uses the `voltage_map` and `pg_pin` statements. The `voltage_map` statement defines the power supplies and default voltage values, whereas the `pg_pin` statements specify the power supply associated with each pin of each cell, as well as some of the power-related pin parameters. For more information about library requirements for multivoltage analysis, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

PrimeTime supports partial and no PG pin cells in the Liberty PG pin library. Partial PG pin cells are cells that are missing power pins, ground pins, or both power and ground pins. For cells that have no signal pins, PrimeTime removes these cells from the design at link time. No timing or power information is reported for these types of cells.

PrimeTime and PrimeTime PX provide the capability to automatically convert and update the library power and ground (PG) pins. With this feature, the tool creates PG pins automatically through library conversion, and the library update specifications are provided in a PG Tcl file. For more information about the PG Tcl file, see [Library PG Tcl File](#).

The second essential requirement for multivoltage analysis is Liberty library data that accurately describes timing behavior at specific voltages. PrimeTime supports voltage scaling by interpolating between data in separate libraries that have been characterized at different supply voltages. You invoke voltage and temperature scaling by using the `define_scaling_lib_group` command. Note that the behavior of PrimeTime is different from that of Design Compiler, which links the design at precise corner voltages. For more information, see [CCS Cross-Library Scaling and Exact Matching](#).

UPF Commands

Multiple power supplies at different voltages can supply power to different domains occupying different areas of the chip. Some power domains can be selectively shut off to conserve power during periods of inactivity. Level shifter cells convert signals leaving one domain and entering another, while isolation cells supply constant signal levels at the outputs of domains that are shut down. Power-down domains can contain retention registers that can retain logic values during the power-down period. Power-switch cells, operating under the control of a power-controller block, switch the power on and off to specific domains. You can specify all of these multivoltage aspects of a design in the UPF language.

A standard PrimeTime license includes UPF support. No additional license is required specifically for UPF. However, power analysis requires a PrimeTime PX license, and IR drop annotation requires a PrimeTime SI license.

For background information about the Synopsys multivoltage flow and using UPF commands in the flow, see the *Synopsys Multivoltage Flow User Guide*.

PrimeTime uses UPF-specified power intent and the `set_voltage` command to determine the voltage on each power supply pin of each cell. Based on the UPF-specified power intent and `set_voltage` information, PrimeTime builds a virtual model of the power network and propagates the voltage values from UPF supply nets to the PG pins of leaf instances. Since PrimeTime does not directly read power state tables, the `set_voltage` command must be consistent with a specific state in the UPF power state table that you intend to verify.

You can enter the UPF commands at the shell prompt. You can also source these commands in a command file or with the `source` or `load_upf` command. Any loaded UPF information is removed upon relinking a design, just like timing assertions loaded from a Synopsys Design Constraints (SDC) file.

The following is a typical sequence of commands used in a PrimeTime timing analysis with UPF-specified power intent. The UPF-related power commands are highlighted in bold.

```
# Read libraries, designs
...
read_lib l1.db
read_verilog d1.v
...

# Read UPF file
# (containing commands such as the following ones)

create_power_domain ...
create_supply_set ...
create_supply_net ...
create_supply_port ...
create_power_switch ...
connect_supply_net ...
set_scope block1
load_upf block1_upf.tcl

# Read SDC and other timing assertions
source d1.tcl

# Define scaling library groups for voltage and temperature scaling
define_scaling_lib_group library_list

# Read SDC and other timing assertions
...
set_voltage -object_list supply_net_name
set_voltage -cell ... -pg_pin_name ... value
# (sets voltage on supply nets or IR drop on cell power pins;
# PrimeTime SI license required for -cell and -pg_pin_name options)
set_temperature -object_list cell_list value

# Perform timing, signal integrity analysis
report_timing
```

PrimeTime reads and uses the UPF information, but it does not modify the power domain description in any structural or functional way. Therefore, PrimeTime does not write out any UPF commands with the `write_script` command, and PrimeTime does not recognize the `save_upf` command.

PrimeTime supports a subset of the commands and command options in the IEEE 1801 (UPF) specification. The unsupported commands are either unrelated to the functions of PrimeTime (for example, synthesis and physical implementation) or represent capabilities that have not been implemented.

The supported power supply commands can be divided into the following categories:

- Power domain commands:

```
connect_supply_net  
create_power_domain  
create_power_switch  
create_supply_net  
create_supply_port  
create_supply_set  
set_domain_supply_net
```

- Isolation and retention commands:

```
set_isolation  
set_isolation_control  
set_port_attributes  
set_design_attributes  
set_retention  
set_retention_control
```

- Find and query commands:

```
find_objects  
query_cell_instances  
query_cell_mapped  
query_net_ports  
query_port_net
```

- Flow commands:

```
load_upf  
set_scope  
set_design_top  
upf_version
```

- Related non-UPF commands:

```
check_timing -include signal_level
check_timing -include supply_net_voltage
check_timing -include unconnected_pg_pins
get_power_domains
get_power_switches
get_supply_nets
get_supply_ports
get_supply_sets
report_power_domain
report_power_network
report_power_pin_info
report_power_switch
report_supply_net
report_supply_set
set_level_shifter_strategy
set_level_shifter_threshold
set_related_supply_net
set_temperature
set_voltage
```

Some UPF commands are essential for synthesis and other implementation tools, but they supply information that is either not used during PrimeTime analysis or is available from other sources. PrimeTime accepts these commands as valid UPF syntax, but otherwise ignores them. PrimeTime does not provide man pages for these commands. The following UPF commands are ignored by PrimeTime:

```
add_port_state
add_pst_state
create_pst
map_isolation_cell
map_level_shifter_cell
map_power_switch
map_retention_cell
name_format
set_level_shifter
connect_logic_net
create_logic_net
create_logic_port
```

The following commands are rejected by PrimeTime as unknown syntax. They are either not supported by the Synopsys UPF command subset in all Synopsys tools or they exist only in the RTL UPF.

```
add_domain_elements
bind_checker
create_hdl2upf_vct
create_upf2hdlvct
merge_power_domains
save_upf
set_pin_related_supply
```

For descriptions of the supported UPF commands in PrimeTime, see

- [UPF Supply Sets](#)
- [Virtual Power Network](#)
- [Setting Voltage and Temperature](#)
- [Library Support for Multivoltage Analysis](#)
- [Multivoltage Reporting and Checking](#)

For background information or more information about using each command throughout the synthesis, implementation, and verification flow, see the IEEE 1801 (UPF) specification. For more information about the Synopsys multivoltage flow and Synopsys usage of UPF commands, including usage in PrimeTime, see the *Synopsys Multivoltage Flow User Guide*.

UPF Supply Sets

A supply set is an abstract collection of supply nets, consisting of two supply functions, power and ground. A supply set is domain-independent, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can physically implemented (placed and routed), its supply sets must be refined or associated with actual supply nets.

For more information about supply sets, see the *Synopsys Multivoltage Flow User Guide*.

UPF Supply Set Handles

A supply set handle is an abstract supply set implicitly created for a power domain. These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can be physically implemented, its supply set handles must be refined or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

By default, when you create a power domain, PrimeTime creates the following predefined supply set handles for that power domain:

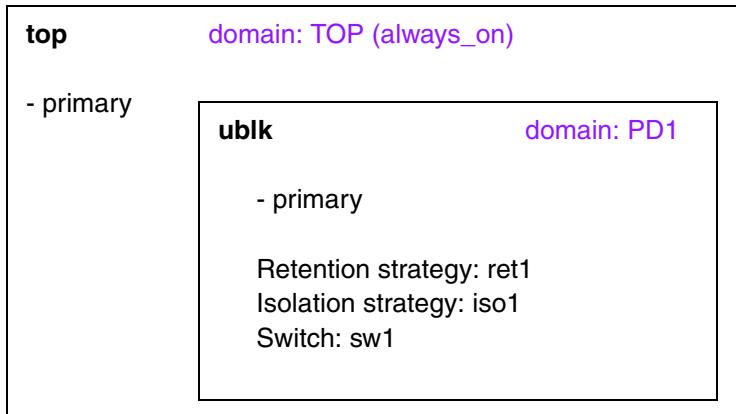
- primary
- default_isolation
- default_retention

In addition to the predefined supply set handles, you can create your own supply set handles with the following command:

```
create_power_domain -supply {supply_set_handle [supply_set_ref]}
```

The design example in [Figure 13-1](#) has two power domains, TOP and PD1. [Example 13-1](#) shows the use of supply set handles for this design example.

Figure 13-1 Design Example Using Supply Set Handles



Example 13-1 Design Example Using Supply Set Handles

```

# create power domains
create_power_domain TOP -include_scope
create_power_domain PD1 -elements {u1}

# retention/isolation/switch
set_retention ret1 -domain PD1
set_isolation iso1 -domain PD1
create_power_switch sw1 -domain PD1 \
    -input_supply_port {IN TOP.primary.power} \
    -output_supply_port {OUT PD1.primary.power} \
    -control_port {CTRL inst_on} \
    -on_state {on1 IN {CTRL}}

# set_voltage
set_voltage 0.9 -object_list {TOP.primary.power}
set_voltage 0.9 -object_list {PD1.primary.power}
set_voltage 0.0 -object_list {TOP.primary.ground PD1.primary.ground}

```

For more information about creating and working with supply set handles, see the *Synopsys Multivoltage Flow User Guide*.

Identifying the Power and Ground Supply Nets With Attributes

To report the power and ground supply nets associated with the primary supply sets of each power domain, query UPF attributes as shown in [Example 13-2](#). This script collects and reports the power and ground supply nets associated with the primary supply sets of each power domain.

Example 13-2 Identifying the Power and Ground Supply Nets

```
# ISS is on by default
# set upf_create_implicit_supply_sets true
...
set domain_list [get_power_domains *]
foreach_in_collection domain $domain_list {
    echo [format "Domain: %s " [get_attribute $domain full_name]]
    set sset [get_attribute $domain primary_supply]
    echo [format " Primary supply: %s " [get_attribute $sset full_name]]
    echo [format "     power net: %s " [get_attribute \
        [get_attribute $sset power] full_name]]
    echo [format "     ground net: %s " [get_attribute \
        [get_attribute $sset ground] full_name]]
    append_to_collection power_nets [get_attribute $sset power] -unique
    append_to_collection ground_nets [get_attribute $sset ground] -unique
}

echo "power nets associated with domains:"
query $power_nets

echo "ground nets associated with domains:"
query $ground_nets
...
```

The script uses the following UPF attributes, which return the predefined supply set handles associated with power domain objects:

- primary_supply
- default_isolation_supply
- default_retention_supply

In addition, the following UPF attributes return the power or ground net functions for a supply set:

- power
- ground

For more information about these attributes, see [Table 13-1](#).

Virtual Power Network

In the PrimeTime multivoltage analysis flow, you typically read in the design netlist, source a UPF command script that defines the power supply intent, and then source an SDC script that specifies the timing constraints for analysis. From the design netlist and UPF commands, PrimeTime builds a virtual power network that supplies power to all the PG pins of all the leaf-level cells in the design netlist.

PrimeTime builds the power supply network based on UPF commands, including `create_power_domain`, `create_supply_net`, `create_supply_set`, `create_supply_port`, `set_domain_supply_net`, and `connect_supply_net`. It determines which cells belong to which power domains and traces the supply connections through the supply nets and supply ports down to the leaf level.

The `set_retention` and `set_retention_control` UPF commands determine the supply connections to the retention registers. Similarly, the `set_isolation` and `set_isolation_control` UPF commands determine the connections of the backup supply nets of the isolation cells. The `set_port_attributes` and `set_design_attributes` commands specify the attributes of the source or sink elements for the `set_isolation` command. The `set_port_attributes` command specifies information that is relevant to ports on the interface of the power domains. This information is used to determine isolation and guard requirements for the port. The `set_related_supply_net` command specifies the attributes and their values on desired cells.

The voltage of the supply port is determined in the following order of precedence, from highest to lowest:

- The `-driver_supply` or `-receiver_supply` option, along with the `-ports` option, for driver supply on input ports and receiver supply on output ports
- The `-driver_supply` or `-receiver_supply` option, along with `-elements` option, for driver supply on input ports and receiver supply on output ports
- The `set_related_supply_net` command
- The primary supply of the power domain of the top design

The `set_level_shifter` UPF command is not supported in PrimeTime; therefore, the `connect_supply_net` command must explicitly specify the power supply connections of the PG pins of level shifters. Design Compiler automatically generates the `connect_supply_net` commands when it inserts level shifters and writes them out with the `save_upf` command. Therefore, you do not need to create these commands yourself when you read the UPF produced by Design Compiler. To specify the behavior of signal level mismatch checking by the `check_timing -include signal_level` command, use the

`set_level_shifter_strategy` and `set_level_shifter_threshold` non-UPF commands. These commands do not affect PG connectivity.

The `set_related_supply_net` UPF command associates a supply net to one or more ports of the design. Use the `object_list` option of the `set_related_supply_net` command to specify the list of ports associated with supply nets. In the absence of this command, PrimeTime assumes that ports are supplied by the voltage of the design operating condition. Using the `-receiver_supply` and `-driver_supply` options of the `set_port_attributes` command, you can replace the `set_related_supply_net` command with the `set_port_attributes` command. You can specify the `-receiver_supply` option only on the top-level ports and design. PrimeTime PX issues a warning message if you specify ports that are not on the top level of the design hierarchy, and the attribute is ignored.

Setting Voltage and Temperature

To determine the supply voltage on each PG pin of each cell, PrimeTime uses the following order of precedence, from highest to lowest. The UPF port voltage has higher precedence than the non-UPF port voltage.

- The `-driver_supply` or `-receiver_supply` option, along with the `-ports` option, for driver supply on input ports or receiver supply on output ports
- The `-driver_supply` or `-receiver_supply` option, along with `-elements` option, for driver supply on input ports or receiver supply on output ports
- The `set_related_supply_net` command
- The primary supply of the power domain of the top design
- The `set_voltage` on a PG pin (PrimeTime SI license required)
- The `set_voltage` on a supply net
- The `set_operating_conditions` applied at the design level
- The default supply voltage in library cell definition (`voltage_map` in Liberty syntax)

By default, PrimeTime uses the supply voltages in the library cell definitions, as specified by the `voltage_map` statement in Liberty syntax for the library cell. For more information, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

Override the library-specified voltages by using the `set_operating_conditions` command at the design level (in other words, without using the `-object_list` option of the command). For example,

```
pt_shell> set_operating_conditions WCIND
```

The supply voltages associated with the named operating condition are specified in the library definition for the operating condition or by the `create_operating_conditions` command.

Override both the library-specified and operating-condition voltage settings with the `set_voltage` command for specific supply nets or specific PG pins.

The command specifies one or more supply nets (`-object_list supply_nets`) or PG pins (`-cell cell -pg_pin_name pg_pin`) and applies a specified voltage to the listed objects. You can specify either a single voltage (`max_voltage`) or both the minimum-delay and maximum-delay voltages (`-min min_voltage` and `max_voltage`) used for analysis. Note that the `-min min_voltage` option specifies the voltage used for minimum-delay analysis, which is typically the larger voltage value. A PrimeTime SI license is required to use the `-cell` and `-pg_pin_name` options. For example, to set a voltage of 1.20 on the supply net VDD1, use this command:

```
pt_shell> set_voltage 1.20 -object_list VDD1
```

To set a maximum-delay voltage of 0.91 and a minimum-delay voltage of 1.18 on the PWR pin of cell XA91, use this command (PrimeTime SI license required):

```
pt_shell> set_voltage 0.91 -min 1.18 -cell XA91 -pg_pin_name PWR
```

A PAD pin is defined by having its `is_pad` attribute set to `true` in the Liberty description of the pin. PrimeTime detects ports connected to PAD cells and determines the port voltage in the following order of increasing priority:

1. The nominal voltage of the main library
2. The nominal voltage of the driving library cell
3. The design operating condition
4. The voltage of the connected pin of the PAD cell
5. The explicit port voltage, set with the `set_related_supply_net` command in UPF mode or the `set_operating_conditions -object_list port` command in non-UPF mode

Since the operating condition defines only the single voltage value, you must exercise caution when setting operating conditions on designs containing multirail cells, such as level shifters and power management cells. PrimeTime applies the operating condition voltage value only to the first library power rail. For example, the value is applied to the first nonground `voltage_map` line in the `.lib` file. Thus, the order of the `voltage_map` entries in the `.lib` file is important in this partial multivoltage flow.

The correct way to completely avoid this issue on multirail cells is to use a proper multivoltage flow, such as UPF, that defines all design rails using the `set_voltage` command. To verify voltage values of all rails of a multirail cell, use the `report_power_pin_info` command.

Use the `report_power_pin_info` and `report_timing -voltage` commands to display the voltage of PG and signal pins respectively. For more information, see [Reporting Power Supply Network Information](#).

You can similarly specify the operating temperature for specific cells by using the `set_temperature` command, overriding the design operating condition settings. Use this command to model the effects of on-chip temperature variation. For example, you can use the following syntax to set the temperature to 130 degrees Celsius on cell XA91:

```
pt_shell> set_temperature 130 -object_list [get_cells XA91]
```

Library Support for Multivoltage Analysis

You can provide a set of libraries characterized at a range of supply voltage and temperature values, and then scale the cell data between these libraries. This is the preferred way to provide multivoltage library data. The `define_scaling_lib_group` command specifies the list of CCS libraries that have been characterized at different voltages and temperatures, and also invokes delay and constraint scaling between those libraries. PrimeTime interpolates between the specified libraries to determine the cell behavior at the exact voltage and temperature conditions that have been set on each cell. For more information, see [CCS Cross-Library Scaling and Exact Matching](#).

Another way to provide multivoltage library data is to use the `link_path_per_instance` variable to direct PrimeTime to use different libraries for different cell instances. This variable, when set before linking the current design, overrides the default `link_path` setting for selected leaf cell or hierarchical cell instances. The format is a list of lists. Each sublist consists of a pair of elements: a set of instances and a `link_path` specification to be used for those instances. For example,

```
set link_path {* lib1.db}
set link_path_per_instance [list
    [list {ucore} {* lib2.db}]
    [list {ucore/usubblk} {* lib3.db}]]
```

The listed instances can be hierarchical blocks that represent individual power domains. The corresponding libraries might be characterized at different supply voltages or under varying conditions such as body bias. If a given block matches multiple entries in the per-instance list, the more specific entry overrides the more general entry. In the preceding example:

- lib3.db is used to link blocks “ucore/usubblk” and below.
- lib2.db is used to link “ucore” and below (except within “ucore/subblk”).
- lib1.db is used for the remainder of the design (everything except within “ucore”).

Using either type of library data method, PrimeTime uses the scaled or library-specified supply voltage information to accurately determine the delays and slews of signals.

For post-layout analysis using detailed annotated parasitics, PrimeTime determines the delays and slews using analog waveforms. For pre-layout analysis (in the absence of detailed annotated parasitics), PrimeTime performs geometric-like scaling of the transition times and net delays, taking into account the respective voltage swings of the driver and load, and the logic thresholds. In either case, PrimeTime reports transition times in terms of local-library thresholds and local voltages.

The use of local trip points and voltages can cause an apparent improvement in transition time along nets in a path. For example, a driver cell might have a transition time of 1.0 ns measured between 20 percent and 80 percent of VDD, while the load on the same net has a transition time of 0.67 ns measured between 30 percent and 70 percent of VDD:

$$1.0 * (70-30)/(80-20) = .67$$

To disable prelayout scaling, set the `timing_prelayout_scaling` variable to `false`.

Support for Fine-Grained Switch Cells in Power Analysis

PrimeTime supports the fine-grained switch cells that are defined in the library, using the Liberty syntax. A library cell is considered a fine-grained switch cell only when the cell is defined with the cell-level `switch_cell_type` attribute, as shown in the following example:

```
switch_cell_type: fine_grain;
```

Use the `report_power` command to view the power consumption of the fine-grained switch cells.

PrimeTime uses the voltage on the internal PG pins to determine the signal level of the pins that are related to the internal PG pins. Use the `set_voltage` command to specify design-specific voltage on the internal PG pins. If this is not specified, the voltage is derived from the `voltage_map` attribute in the library.

Note:

Using the `set_voltage` command on the external PG pins of the switch cell does not affect the voltage on its internal PG pins.

PrimeTime PX derives the `ON` and `OFF` states of the internal PG pins based on the `switch_function` and the `pg_function` definitions specified in the library for power analysis.

When you use the `connect_supply_net` command, PrimeTime supports explicit connection to the internal PG pins of fine-grained switch cells. Use the `report_power_pin_info` command to report the power on these pins.

Note:

The `report_power_pin_info` command reports the power of internal PG pins and corresponding external PG pins. The command does not report the power numbers separately for the internal and external PG pins.

For more information about defining a fine-grained switch, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*. For more information about power analysis, see the “Multivoltage Power Analysis” chapter in the *PrimeTime PX User Guide*.

Multivoltage Reporting and Checking

Several commands are available for reporting the power supply network and checking the network for errors or unusual conditions, including `get_*` commands, `report_*` commands, and `check_timing`.

Collection (`get_*`) Commands

The `get_*` commands each create a collection of objects for reporting purposes. You can create a collection of existing power domains, power, switches, supply nets, or supply ports with the following commands.

```
get_power_domains  
get_power_switches  
get_supply_nets  
get_supply_ports  
get_supply_sets
```

The `get_power_domains` command returns a collection of power domains previously created with the `create_power_domain` command. You can pass the collection to another command for further processing, for example, to extract the name, scope, and elements attributes associated the domains. The `create_power_domain` command options allow you to limit the collection to the power domains meeting specified criteria. For example, the `-of_objects` option causes the collection to include only the power domains to which the specified objects belong. The `patterns` option limits the collection to power domains whose names match the specified pattern. The `get_power_switches`, `get_supply_nets`, and `get_supply_ports` commands operate in a similar manner and have similar command options.

Table 13-1 lists the attributes of UPF power domain, supply net, supply port, and power switch objects.

Table 13-1 UPF Collection Objects

Command and object class	Attribute	Attribute type	Comment
get_power_domains	name	String	Name as entered
upf_power_domain	full_name	String	Hierarchical name
	elements	Collection	
	scope	String	
	supplies	String	List of function keyword and supply set name
	primary_supply	Collection	Returns the UPF supply set handle associated with the power domain
	default_isolation_supply		
	default_retention_supply		
get_supply_nets	name	String	Name as entered
upf_supply_net	full_name	String	Hierarchical name
	domains	Collection	
	resolve	String	one_hot, etc.
	voltage_min	Float	Minimum-delay voltage
	voltage_max	Float	Maximum-delay voltage
	static_prob	Float	Probability of logic 1 (for power analysis)

Table 13-1 UPF Collection Objects (Continued)

Command and object class	Attribute	Attribute type	Comment
get_supply_ports	name	String	Name as entered
upf_supply_port	full_name	String	Hierarchical name
	domain	Collection	
	direction	String	In, out
	scope	String	
get_supply_sets	name	String	Name as entered
upf_supply_set	full_name	String	Hierarchical name
	ground	Collection (one of upf_supply_net object)	Ground or power net of the supply set
	power		
get_power_switches	name	String	Name as entered
upf_power_switch	full_name	String	Hierarchical name
	domain	Collection	
	output_supply_port_name	String	Name as entered
	output_supply_net	Collection	Supply net
	input_supply_port_name	String	Name as entered
	input_supply_net	Supply net	
	control_port_name	List of strings	Name as entered
	control_net	Collection	
	on_state	List of strings	Format: state_name input_supply_port boolean_function

Reporting Power Supply Network Information

To obtain information about the power supply network, use these reporting commands:

```
report_power_domain  
report_power_network  
report_power_pin_info  
report_power_switch  
report_supply_net  
report_supply_set  
report_timing -voltage
```

The `report_power_domain` command reports the power domains previously defined with `create_power_domain` commands. The reports includes the names of the PG nets of each domain. For example,

```
pt_shell> report_power_domain [get_power_domains B]  
...  
Power Domain : B  
Scope : H1  
Elements : PD1_INST H2 H3  
  
Connections : -- Power -- -- Ground --  
Primary H1/H7/VDD H1/H7/VSS  
-----
```

The `report_power_network` command reports all connectivity of the entire power network, through ports and switches. You can restrict the report to one or more specified nets. For example,

```
pt_shell> report_power_network -nets H1/VDD  
...  
Supply Net: H1/VDD  
  
Connections:  
Name Object type Domain  
-----  
VDD Supply_port D1  
H1/VDD Supply_port D2  
U1/VDD PG_power_pin D1  
SW1/IN Switch_input D1
```

The `report_power_pin_info` command reports the PG connectivity of leaf-level cells or library cells used. For example,

```
pt_shell> report_power_pin_info [get_cells -hierarchical]
...
```

Note: Power connections marked by (*) are exceptional

Cell	Power Pin Name	Type	Voltage Max	Voltage Min	Power Net Connected
PDO_INST/I0	PWR	primary_power	0.9300	1.2700	int_VDD_5
PDO_INST/I0	GND	primary_ground	0.0000	0.0000	A_VSS
PDO_INST/I1	PWR	primary_power	0.9300	1.2700	int_VDD_5
PDO_INST/I1	GND	primary_ground	0.0000	0.0000	A_VSS
I0	PWR	primary_power	1.0000	1.0000	int_VDD_4 (*)
I0	GND	primary_ground	0.0000	0.0000	int_VSS_4 (*)
I1	PWR	primary_power	1.1500	1.1500	T_VDD
I1	GND	primary_ground	0.0000	0.0000	T_VSS
PD1_INST/I0	PWR	primary_power	1.0000	1.0000	int_VDD_4
PD1_INST/I0	GND	primary_ground	0.0000	0.0000	int_VSS_4
PD1_INST/I1	PWR	primary_power	1.0000	1.0000	int_VDD_4
PD1_INST/I1	GND	primary_ground	0.0000	0.0000	int_VSS_4

```
pt_shell> report_power_pin_info [get_lib_cells -of [get_cells -hier]]
...
```

Cell	Power Pin Name	Type	Voltage
INVX2	PWR	primary_power	1.0000
INVX2	GND	primary_ground	0.0000
BUFX2	PWR	primary_power	1.0000
BUFX2	GND	primary_ground	0.0000
AND2X1	PWR	primary_power	1.0000
AND2X1	GND	primary_ground	0.0000
OR2X1	PWR	primary_power	1.0000
OR2X1	GND	primary_ground	0.0000

The `report_power_switch` command reports the power switches previously created with the `create_power_switch` command. Here is an example of a power switch report:

```
pt_shell> report_power_switch
...
Total of 3 power switches defined for design 'top'.

Power Switch : sw1
-----
Power Domain : PD_SODIUM
Output Supply Port : vout VN3
Input Supply Port : vin1 VN1
Control Port: ctrl_small ON1
Control Port: ctrl_large ON2
Control Port: ss SUPPLY_SELECT
On State : full_s1 vin1 { ctrl_small & ctrl_large & ss }
-----
...

```

The `report_supply_net` command reports the supply net information for a power domain or specified object in a power domain. For example,

```
pt_shell> report_supply_net
...
Total of 14 power nets defined.

Power Net 'VDD_Backup' (power)
-----
Backup Power Hookups: A
-----

Power Net 'VSS_Backup' (ground)
-----
Backup Ground Hookups: A
-----

Power Net 'T_VDD' (power,switchable)
-----
Voltage states: {1.2}
Voltage ranges: {1.1 1.3}
Max-delay voltage: 1.15
Min-delay voltage: 1.15
Primary Power Hookups: T
-----

Power Net 'A_VDD' (power)
-----
Max-delay voltage: 1.15
Min-delay voltage: 1.15
Primary Power Hookups: A
-----
```

The `report_supply_set` command reports detailed information about the defined supply sets. For example,

```
pt_shell> report_supply_set
...
Total of 1 supply net defined for design "mydesign".
```

```
-----
Supply Set   : primary_sset
Scope        : top
Function     : power, supply net association: VDD
Function     : ground, supply net association: VSS
-----
```

The `report_timing -voltage` command allows you to determine the voltage of signal pins on timing paths. Similarly, you can use the `voltage` attribute on the `timing_point` objects in custom reports with the `get_timing_paths` command. The following example shows the `report_timing -voltage` command:

```
report_timing -voltage
...
-----
```

Point	Incr	Path	Voltage
clock HVCLK_SYNC_D (rise edge)	0.00	0.00	
clock network delay (propagated)	0.00	0.00	
input external delay	5.00	5.00 f	
in0[0] (in)	0.00	5.00 f	1.00
ALU1/in0[0] (alu_ao_3)	0.00	5.00 f	
ALU1/t0_reg_0_/D (SDFQM1RA)	0.00	5.00 f	1.03
data arrival time			5.00

Querying Power and Ground Pin Attributes

In addition to using reporting commands, you can obtain PG pin information on a cell, port, or library cell by querying the attributes in [Table 13-2](#).

Table 13-2 Power Pin Attributes

Attribute name	Object	Type	Description
pg_pin_info	cell, port	Collection	Contains a collection of these pg_pin_info objects: <ul style="list-style-type: none">• pin_name• type• voltage_for_max_delay• voltage_for_min_delay• supply_connection
lib_pg_pin_info	lib_cell	Collection	Contains a collection of these lib_pg_pin_info objects: <ul style="list-style-type: none">• pin_name• type• voltage

To query these PG pin attributes, you can use the following example, which iterates over a list of cells and returns attributes of interest:

```
set cell_list [get_cells ...]
foreach cell $cell_list {
    foreach_in_collection pg_info [get_attr $cell pg_pin_info] {
        echo [format "Pin name: %s" [get_attr $pg_info pin_name]]
        echo [format "Voltage: %f" [get_attr $pg_info voltage_for_max_delay]]
    }
}
```

Using the check_timing Command

You can use the `check_timing` command to check the validity of the power supply connections, including the following types of checks:

- Voltage set on each supply net segment (`supply_net_voltage`)
- Supply net connected to each PG pin of every cell (`unconnected_pg_pins`)
- Compatible signal levels between driver and load pins (`signal_level`)

Voltage Set on Each Supply Net Segment

Every supply net segment must have a voltage set on it with the `set_voltage` command. To verify that this is the case, include a supply net voltage check in the `check_timing` command, as in the following example:

```
pt_shell> check_timing -include supply_net_voltage -verbose
Information: Checking 'no_clock'.
Information: Checking 'no_input_delay'.
...
Information: Checking 'supply_net_voltage'.
Warning: There are '2' supply nets without set_voltage. (UPF-029)
    VG
    VS1
Warning: The voltage of driving cell 'INV'(0.700) does not match the
related supply voltage at port 'reset'(1.080). (UPF-408)
Information: Checking 'pulse_clock_non_pulse_clock_merge'.
```

Supply Net Connected to Each PG Pin of Every Cell

Each PG pin of every cell should be connected to a UPF supply net. To verify that this is the case, include an unconnected PG pin check in the `check_timing` command, as in the following example:

```
pt_shell> check_timing -include unconnected_pg_pins
```

Each UPF supply net connection can be either explicit (as specified by the `connect_supply_net` command) or implicit (due to the assignment of the cell to a power domain, isolation strategy, retention strategy, and so on).

Compatible Driver-to-Load Signal Levels

To have PrimeTime check for mismatching voltage levels between cells that use different supply voltages, use the following `check_timing` command:

```
pt_shell> check_timing -include signal_level
```

This type of timing check traverses all nets and determines whether the output voltage level of each driver is sufficient to drive the load pins on the net. PrimeTime reports any driver-load pairs that fail the voltage level check. It assumes that there is no voltage degradation along the net. You can fix a violation by changing the supply voltages or by inserting a level shifter between the driver and load.

PrimeTime performs signal level checking by comparing the input and output voltage levels defined in the library for the pins of leaf-level cells. The checked signal levels are based on either the gate noise immunity (or signal level range) margins defined by the Liberty syntax `input_voltage` and `output_voltage` or a comparison of driver/load supply voltages. You can control driver and load supply voltage mismatch reporting by using the `set_level_shifter_strategy` and `set_level_shifter_threshold` commands.

The `set_level_shifter_strategy` command specifies the type of strategy used for reporting voltage mismatches: `all`, `low_to_high`, or `high_to_low`. The `low_to_high` strategy reports the voltage level mismatches when a source at a lower voltage drives a sink at a higher voltage. The `high_to_low` strategy reports the voltage level mismatches when a source at a higher voltage drives a sink at a lower voltage. The `all` strategy (the default) reports both types of mismatches.

The `set_level_shifter_threshold` command specifies the absolute and relative mismatch amounts that trigger an error condition. If there is a voltage difference between driver and load, the difference must be less than both the absolute and relative thresholds to be considered acceptable. For example, the following command sets the voltage difference threshold to 0.1 volt and the percentage threshold to 5 percent:

```
pt_shell> set_level_shifter_threshold -voltage 0.1 -percent 5
```

A voltage difference that is more than 0.1 volt or more than five percent of the driver voltage is reported as a mismatch. The default thresholds are both zero, so if you set one threshold to a nonzero value, you need to set the other to a nonzero value as well. To disable either absolute or percentage threshold checking, set its threshold to a large value.

The percentage difference is determined as follows:

$$\text{abs}(\text{driver(VDD)} - \text{load(VDD)}) / \text{driver(VDD)} * 100$$

PrimeTime reports either of the following conditions as an “incompatible voltage” error:

- Driver VOmax > Load VImax
- Driver VOmin < Load VImin

PrimeTime reports either of the following conditions as a “mismatching driver-load voltage” warning:

- Driver VOH < Load VIH
- Driver VOL > Load VIL

Here is an example of a voltage mismatch report:

```
pt_shell> check_timing -verbose -include signal_level
...
Error: There are 2 voltage mismatches
      MIN-MAX - driver vomax > load vimax:
Driver      Voltage    Load      Voltage    Margin
-----
u2/Z        2.50      u3/A      0.90      -1.60
u3/Z        2.10      u5/A      1.47      -0.63

Warning: There is 1 voltage mismatch
      MIN-MAX - driver vol > load vil:
Driver      Voltage    Load      Voltage    Margin
-----
u2/Z        0.30      u3/A      0.20      -0.10
```

The logic library defines the input and output voltages in terms of the supply voltage in Liberty syntax. PrimeTime correctly calculates the voltages for comparison. For example, a library might define the input and output voltage levels as follows:

- $V_{IL} = 0.3 * VDD$, $V_{IH} = 0.7 * VDD$
- $V_{OL} = 0.4$, $V_{OH} = 2.4$
- $V_{Imin} = -0.5$, $V_{Imax} = VDD + 0.5$
- $V_{Omin} = -0.3$, $V_{Omax} = VDD + 0.3$

PrimeTime calculates the input and output voltages, taking into account the supply voltages that apply to each cell.

For proper transition time scaling for cells with multiple power rails, PrimeTime requires that you define the `input_signal_level` and `output_signal_level` attributes on multirail cells. For `check_timing -include signal_level`, PrimeTime uses the definitions of the `input_voltage` and `output_voltage` attributes on all pins.

Even without `input_voltage` or `output_voltage` specified, PrimeTime still attempts to report the 100 worst mismatches based on rail voltages not being exactly equal. For example,

```
pt_shell> check_timing -include signal_level -verbose
...
Warning: There are 2 voltage mismatches
    MAX-MAX - driver rail != load rail:
    The 100 worst voltage mismatches:
```

Driver	Voltage	Load	Voltage	Margin
u1/Y	4.75	u3/A	3.00	-1.75
u3/Y	3.00	ff2/CLK	4.75	-1.75

```
Warning: There are 2 voltage mismatches
    MIN-MIN - driver rail != load rail:
    The 100 worst voltage mismatches:
```

Driver	Voltage	Load	Voltage	Margin
u1/Y	5.25	u3/A	3.00	-2.25
u3/Y	3.00	ff2/CLK	5.25	-2.25

Table 13-3 summarizes the effects of the threshold and strategy settings.

Table 13-3 Effects of Threshold and Strategy Settings

Signal level check type	Strategy effect			Threshold effect	Explanation
	All	Low_to_high	High_to_low		
Driver VOmax > Load VImax	None	None	None	None	Driver exceeding breakage voltage of the load cell defined by the library is a fatal condition. It cannot be filtered.
Driver VOmin < Load VImin	None	None	None	None	Library-defined signal level margins define safe region of operation and cannot be suppressed.
Driver VOL > Load VIL	None	None	None	None	

Table 13-3 Effects of Threshold and Strategy Settings (Continued)

Signal level check type	Strategy effect			Threshold effect	Explanation
	All	Low_to_high	High_to_low		
Driver VDD != Load VDD	None	Filter any mismatch VDDdriver > VDDload	Filter any mismatch VDDdriver > VDDload	Further filter any mismatch smaller than the tolerance	The user-defined tolerance and strategy are fully used.

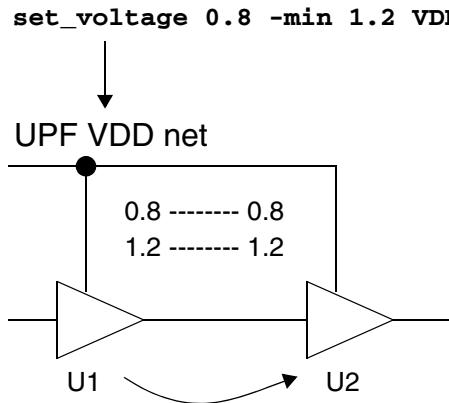
Effect of Correlated Supplies on Signal Level Checking

The signal level check correctly considers the case where minimum and maximum voltages are set on the power nets that supply both of the cells being compared. For example, suppose that you set the voltage for supply VDD as follows:

```
pt_shell> set_voltage 0.8 -min 1.2 VDD    # corner voltages
```

This command sets the supply voltage to 0.8 for maximum-delay analysis and 1.2 Volts for minimum-delay analysis. Suppose that the same VDD supply applies to both the driver and receiver cells of a net, as shown in [Figure 13-2](#).

Figure 13-2 Signal Level Checking With Supply Correlation



PrimeTime considers the fact that the supply voltages of the driver and receiver are correlated. When the supply voltage is low for U1, it is also low for U2, so there is no mismatch. The same is true for the case where the supply voltage is high.

Library PG Tcl File

PG library conversion and update information is specified in a PG Tcl file. You can generate this file with Design Compiler or IC Compiler. Otherwise, you can manually create the file. You can set the `library_pg_file_pattern` variable to specify the location of the PG Tcl file for library PG conversion and update feature. By setting the variable to a valid file name pattern, the tool locates the associated PG Tcl file for the logic libraries that are provided.

You can read the logic library files into PrimeTime using the `read_lib` command or into PrimeTime PX using the external reader (ptxr). You can also use the updated database written out by Library Compiler. At the time the libraries are loaded, the tool performs automatic library PG updates on the in-memory databases.

By default, the `library_pg_file_pattern` variable is set to "" (empty string) so that there is no PG Tcl side file. You can set this variable to use one Tcl file for all databases, one per group of databases, or one per database. You set the variable before loading the library by specifying the following name pattern, where *string* is either the path to the database directory or the leaf file name for the database:

```
pt_shell> set_app_var library_pg_file_pattern string
```

For example, to specify one Tcl file for all databases, use this command:

```
pt_shell> set_app_var library_pg_file_pattern "libpg_sidefile.pg"
```

To specify one Tcl file per database at the same location as the original database files, use a command similar to the following example:

```
pt_shell> set_app_var library_pg_file_pattern "__DIR__/_FILE__.pg"
```

Default Power and Ground Pin Names

You can use a non-PG library with UPF-specified power intent. When you load a non-PG library into memory without specifying a PG Tcl side file, PrimeTime can create one default power pin and one default ground pin for each cell.

You specify the names assigned to the power and ground pins by setting the `lp_default_power_pin_name` and `lp_default_ground_pin_name` variables. These variables are initially set to an empty string, so you must set them explicitly to the desired power and ground pin names before you load the non-PG library. For example,

```
pt_shell> set_app_var lp_default_power_pin_name "VDD"
pt_shell> set_app_var lp_default_ground_pin_name "VSS"
```

The specified names are used for naming the newly created power and ground pins of the library cells stored in memory and for creating the voltage map defined at the library level. Only library cells from non-PG libraries are affected by these variable settings, and only when there is no PG Tcl side file specified by the `library_pg_file_pattern` variable.

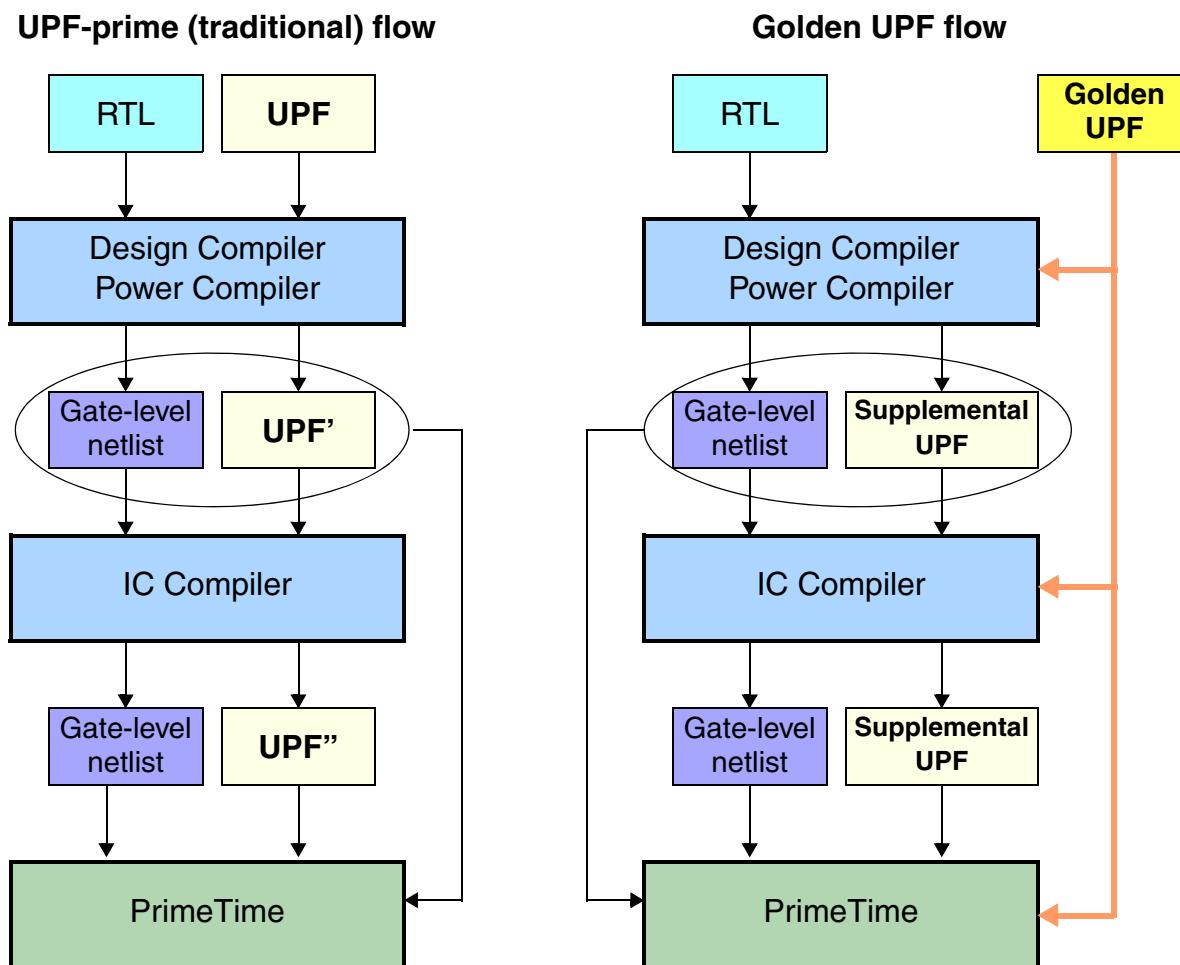
When library loading occurs within PrimeTime, you receive messages about the PG pins being added to the library. If your flow requires multiple loads of the library, you might see these messages more than one time.

Golden UPF Flow

The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Design Compiler and IC Compiler tools.

[Figure 13-3](#) compares the traditional UPF flow with the golden UPF flow.

Figure 13-3 UPF-Prime (Traditional) and Golden UPF Flows



The golden UPF flow maintains and uses the same, original “golden” UPF file throughout the flow. The Design Compiler and IC Compiler tools write power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF’ or UPF” file.

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.
- You can optionally use the Verilog netlist to store all PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

To use the golden UPF flow, you must enable it by setting a variable:

```
icc_shell> set_app_var enable_golden_upf true
```

After you enable this mode, to execute any UPF commands other than query commands, you must put the commands into a script and execute them with the `load_upf` command. You cannot execute them individually on the command line or with the `source` command.

For more information about using the golden UPF mode, see [SolvNet article 1412864, “Golden UPF Flow Application Note.”](#)

Power Domain Mode of Version Z-2007.06

If you want to use the power domain flow based on setting object-specific voltages and operating conditions, set the `power_domains_compatibility` variable to `true`. This causes PrimeTime to enter the “power domain mode,” which enables the Z-2007.06 multivoltage syntax, disables the UPF syntax, and removes any UPF-specified power supply data in the design. Due to differences in command syntax and data infrastructure, the Z-2007.06 power domain capabilities are not compatible with the UPF features.

When PrimeTime is in the power domain mode, you can view the list of power domain commands with the following command:

```
pt_shell> help "power domains"
Power Domains:
  connect_power_domain    # Connect power domain
  connect_power_net_info  # Connect power net to cell power pin
  create_power_domain     # Create power domain object
  create_power_net_info   # Create power net info object
  get_power_domains       # Create a collection of power domains
  report_power_domain    # Report power domain
  report_power_net_info  # Report all power net objects
  report_power_pin_info  # Report power pin info
  set_voltage             # Set voltage
```

There is some overlap in command syntax between the power domain mode and the UPF mode. The interpretation of a command such as `create_power_domain` depends on the operating mode.

Physical tools such as IC Compiler and Design Compiler create power distribution networks based on user-specified connectivity in Tcl syntax. PrimeTime SI can use the same set of commands to build a virtual model of the power distribution network. The power domain model, together with the `set_voltage` command, allows PrimeTime SI to determine the voltage at the power pins of each leaf-level instance. These are the power domain commands:

- The `create_power_net_info` command specifies the name of a power supply net or ground net in the design. If it is a power supply, the command also specifies the voltage and whether the power can be switched off.
- The `create_power_domain` command specifies the name of a power domain and lists the hierarchical cells associated with the domain. If no list is provided, the power domain applies to the top level; there can be no more than one such top-level domain. The command also specifies whether the domain can be powered down and if so, the control net.
- The `connect_power_domain` command creates logical power connections for a specified power domain. It specifies the primary, backup, and internal PG nets for a specified power domain. All cells in the power domain inherit the specified power connections.

The backup PG nets are for always-on logic, retention registers, isolation cells, and enable level shifter cells. The internal PG nets are for the switching cells inside the power domain.

- The `connect_power_net_info` command makes power net connections for a specific power pin of a leaf cell. The pin-level connections override the domain-level connections made with the `connect_power_domain` command.

- The `set_voltage` command defines the operating voltage on the power nets defined by the `create_power_net_info` command. You can specify a single voltage or a minimum and a maximum voltage for the power net. If you do not use this command, the available operating condition settings are used.

If you link the design again, the power domain information is discarded.

The following script example performs timing analysis on a multivoltage design:

```
# Enable power domains (non-UPF) mode
set power_domains_compatibility TRUE
# Read libraries, designs
...
read_lib l1.db
read_verilog d1.v
...

# Read libraries, design, SDC, update design and setup link to HSPICE
create_power_net_info -power power_net_name
create_power_domain domain_name -elements object_list
connect_power_domain -primary_power_net power_net \
    -backup_power_net power_net
connect_power_net_info object_list \
    -power_pin_name pin_name -power_net_name net_name

# Read SDC and other timing assertions
source d1.tcl

# Read SDC and other timing assertions
set_voltage on power nets or IR drop on power pins

# Perform timing, signal integrity analysis
report_timing
```

The following commands report information about the power rails:

- `report_power_domain` – Reports the power domains and their connections. If you specify a list of objects, only the power domains of those objects are reported.
- `report_power_net_info` – Reports the names of the power nets and their associated power domain hookups.
- `report_power_pin_info` – Reports power pin information for library cells or for leaf-level cells in the current design. Specify a list of library cells to find out the names, types, and voltage specifications of the power pins in the library cells. Specify a list of leaf-level cells to find out the power net connections to the power pins of those cells.

Multivoltage Method Before Version Z-2007.06

To perform multivoltage analysis, you can specify multiple voltages by applying different operating conditions to different cells with the `set_operating_conditions -object_list` command, by directly annotating voltage drop information with the `set_voltage` command, or by linking blocks to different libraries. (Using the `set_voltage` command requires a PrimeTime SI license.) Any subsequent timing analysis takes into account the supply voltages specified for the design. The `report_cell` command reports the instance-specific operating conditions and supply voltage information.

These commands were available before PrimeTime version Z-2007.06 and are still supported. However, if you use any UPF commands in a session, the pre-Z-2007.06 multivoltage specification method is disabled, and only UPF commands can be used for specifying power supply information. In that case, you can no longer use `set_operating_conditions -object_list` or `set_voltage`, and any multivoltage information previously set with those commands is discarded.

In the pre-Z-2007.06 multivoltage flow, there are two ways to set supply voltages on a linked design:

- Create different operating conditions having different rail voltages, either by defining them in the logic library or by using the `create_operating_conditions` command. Then apply different operating conditions to different hierarchical blocks or leaf-level cells with the `set_operating_conditions -object_list` command. This method is appropriate for specifying voltage areas on the chip.
- Use the `set_voltage` command to set rail voltages directly on hierarchical blocks or leaf-level cells, overriding any applicable operating condition voltages. This method is appropriate for back-annotating voltage drop information from a power rail analysis tool.

You can combine these two methods in the same design. To do so, first apply the operating conditions to define the voltage areas, then apply the voltage drop information. Rail voltages set with the `set_voltage` command override those set with operating conditions on a cell-by-cell basis. However, they do not override operating conditions set at lower levels of hierarchy.

If you do not set conditions with `set_operating_conditions`, PrimeTime SI uses the default operating conditions of the individual libraries that the cells came from. To have the cells use the operating conditions from the main library instead (the first library defined in the link path), set the `default_oc_per_lib` variable to `false`.

To specify different NLDM library cells for different instances, set the `link_path_per_instance` variable to a list, with each list element consisting of a list of instances and the corresponding link paths that override the default link path for each of those instances. For an example, see [Library Support for Multivoltage Analysis](#).

Setting Operating Conditions on Cells

By default, the `set_operating_conditions` command sets the operating conditions for the whole current design. To override global operating conditions and set the conditions for a hierarchical block or cell instance, use the `-object_list` option and list the applicable cells. For example,

```
pt_shell> set_operating_conditions WCCOM5.0
pt_shell> set_operating_conditions -object_list ALU/A4 WCCOM3.5
```

The first command specifies the set of operating conditions for the whole chip. The second command overrides the whole-chip settings for the cell ALU/A4.

You can also set operating conditions on ports. The specified operating conditions apply to the driving cells for the ports (as specified by the `set_driving_cell` command).

With different operating conditions applied to different levels of the cell hierarchy, the lowest-level setting for a cell has priority over higher-level settings. To report the applicable minimum and maximum operating condition settings for a cell, use the `report_cell` command.

Setting Rail Voltages Directly on Cells

To set the rail voltage on a hierarchical block or leaf-level cell (irrespective of operating conditions), use the `set_rail_voltage` command. Using this command requires a PrimeTime SI license. The command is used in a manner similar to the `set_operating_conditions -object_list` command. For example,

```
pt_shell> set_rail_voltage -max -rail_value 1.2 [get_cells {ALU/a5}]
pt_shell> set_rail_voltage -min -rail_value 1.4 [get_cells {ALU/a5}]
```

Multiple Supply Voltage Analysis

PrimeTime SI can analyze designs with different power supply voltages for different cells.

To learn about multiple supply voltage analysis, see

- [Multivoltage Analysis Overview](#)
- [Simultaneous Multivoltage Analysis](#)
- [IR Drop Annotation](#)

Multivoltage Analysis Overview

PrimeTime SI has the ability to analyze designs with different power supply voltages for different cells. It accurately determines the effects of delay, slew, and noise in the presence of differences in supply voltage, taking advantage of cell delay models specified in the logic library.

The multivoltage infrastructure lets you do the following:

- Calculate uncoupled signal net delay, constraints, and timing violations while considering exact driver and load power rail voltages.
- Calculate coupled delay, noise bumps, timing violations, and noise violations while considering exact aggressor rail voltages.
- Perform signal level mismatch checking between drivers and loads.
- Perform the foregoing types of analysis with instance-specific annotated voltage (IR) drop on power rails, and with block-specific or instance-specific temperature.

Most PrimeTime analysis features are capable of producing results that are fairly close to physical (SPICE simulation) analysis. For accurate analysis in PrimeTime SI, the following types of information must be available:

- Power and ground (PG) connectivity, as described in [Multivoltage Design Flow](#).
- Voltage values on PG nets specified with the `set_voltage` command, as described in [Multivoltage Design Flow](#).
- IR drop values specified for PG nets with the `set_voltage` command using the `-cell` and `-pg_pin_name` options, as described in [IR Drop Annotation](#). Specifying IR drop is optional. A PrimeTime SI license is required.
- A set of CCS libraries with delay and constraint data (used with the `define_scaling_lib_group` command) or library delay calculation data for each voltage (used with the `link_path_per_instance` variable). CCS-based library models are recommended for best accuracy. The `define_scaling_lib_group` command invokes delay and constraint scaling between a set of libraries that have been characterized at different voltages and temperatures. The `link_path_per_instance` variable links different cell instances to different library models. For more information, see [Library Support for Multivoltage Analysis](#).
- Settings that define how to use the library data to build delay calculation models. For more information, see the descriptions of the `link_path_per_instance` and `define_scaling_lib_group` commands in [Library Support for Multivoltage Analysis](#).

In previous versions of PrimeTime, multivoltage analysis was supported by connectivity based on power domain syntax (version Z-2007.06) or voltages annotated directly on cell

instances (before version Z-2007.06). These methods are still supported for backward compatibility. A PrimeTime SI license is no longer required for multivoltage analysis, except for instance-specific IR drop annotation with the `set_voltage` command.

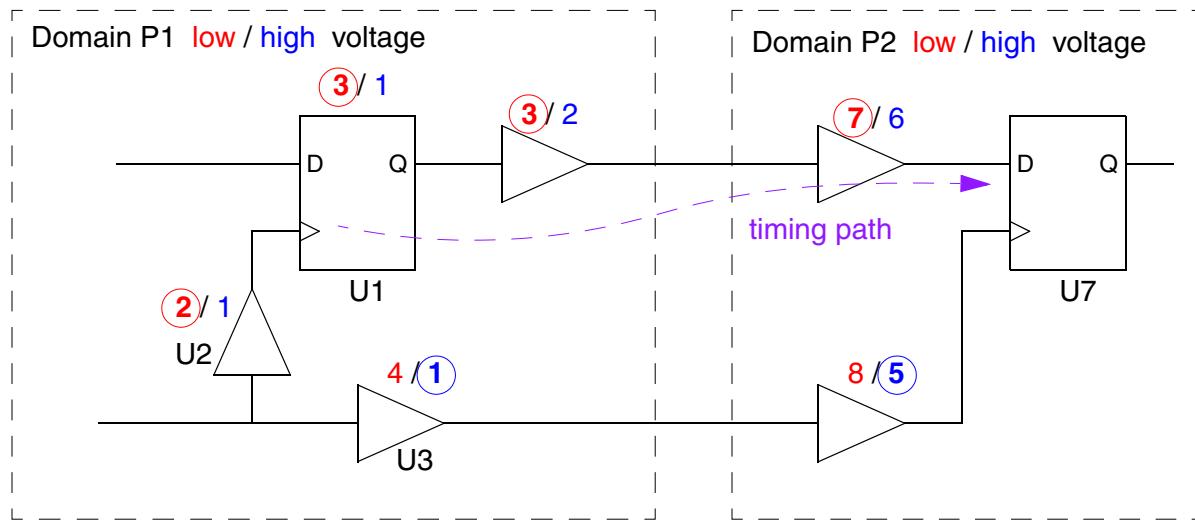
For more information about multivoltage analysis methods used in previous versions, see [Power Domain Mode of Version Z-2007.06](#) and [Multivoltage Method Before Version Z-2007.06](#).

Simultaneous Multivoltage Analysis

In multivoltage designs, some timing paths can cross from one power domain to another. When the power domains can operate at multiple supply voltages, PrimeTime needs to consider the worst possible combination of operating voltages for a given path. To ensure that all potential timing violations are detected, the default analysis considers the worst supply voltage for each cell in the timing path. However, this analysis is pessimistic because cells belonging to the same domain cannot operate at different supply voltages at the same time.

In [Figure 13-4](#), the timing path from U1/CK to U7/D spans two power domains, P1 and P2. Each domain can operate at either of two voltages, low and high. The delay of each cell is shown next to the cell. The red and blue values represent the delays at the low and high supply voltages, respectively.

Figure 13-4 Timing Path Through Two Power Domains



$$\begin{aligned} \text{slack} &= (\text{earliest required time}) - (\text{latest arrival time}) \\ &= (1 + 5) - (2 + 3 + 3 + 7) \\ &= -9 \end{aligned}$$

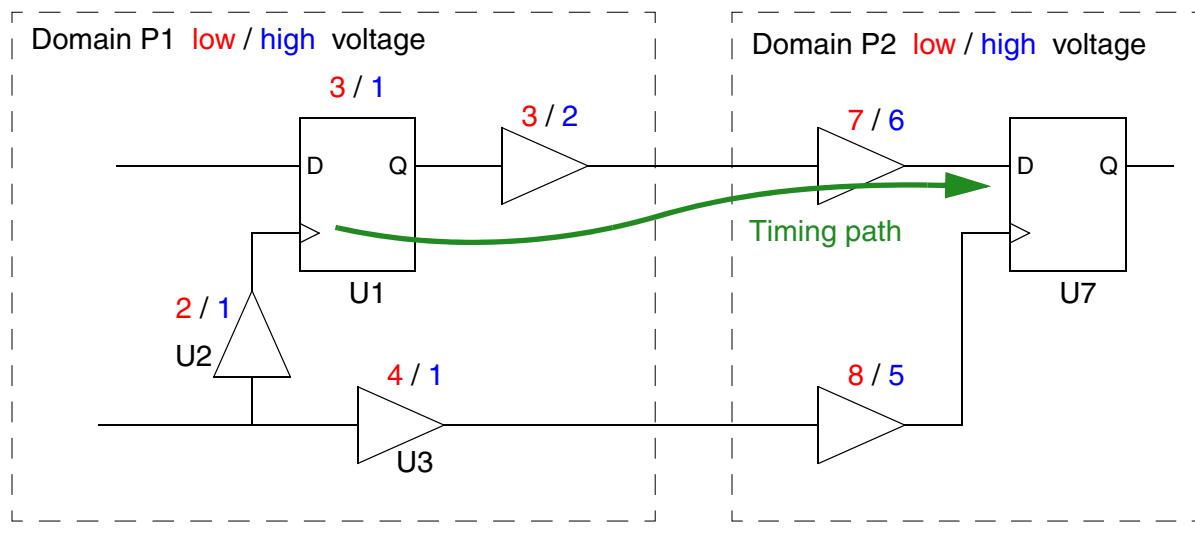
For a setup check, the default analysis considers the longest delays along the data path and shortest delays along the clock path. Therefore, it uses the worst-case delay values shown circled in the figure. The results are pessimistic because, for example, cell U2 cannot operate at the low voltage while cell U3 operates at the high voltage. These two cells belong to the same power domain, so they must both operate at either the low or high voltage at any given time.

If timing violations are found in this analysis, you could reduce this pessimism by analyzing the design multiple times using every possible combination of fixed supply voltages, for example, with V1/V2 set to low/low, low/high, high/low, and high/high. However, a large number of power domains results in a very large number of voltage combinations. The cost of so many analysis runs can be prohibitive.

Simultaneous multivoltage analysis (SMVA) eliminates cross-domain voltage pessimism in a single path-based analysis run, with minimal runtime overhead. Using this strategy, PrimeTime considers the effect of all combinations of low and high voltages for the domains crossed by each path. It does this only for the cross-domain paths, using only the allowed combinations of supply voltages for the domains as defined by the UPF commands.

Figure 13-5 shows the path setup timing calculation using simultaneous multivoltage analysis. During path-based analysis, PrimeTime considers the contribution to the setup slack from power domains P1 and P2 separately, and it considers the both the high and low supply voltages for each domain, but without mixing high and low within a given domain.

Figure 13-5 Simultaneous Multivoltage Slack Calculation



$$\begin{aligned}
 \text{slack} &= (\text{worst slack contribution from P1}) + (\text{worst slack contribution from P2}) \\
 &= \min [\{4 - (2+3+3)\} \text{ or } \{1 - (1+1+2)\}] + \min [\{8-7\} \text{ or } \{5-6\}] \\
 &= \min [-4 \text{ or } -3] + \min [+1 \text{ or } -1] \\
 &= -4 + -1 \\
 &= -5
 \end{aligned}$$

In this example, the worst-case slack is found to be -5 rather than the pessimistic value -9 calculated earlier by conventional worst-case timing analysis. The worst combination of supply voltages is low for P1 and high for P2. Note that simultaneous multivoltage analysis is done only during path-based analysis because the worst-case combination of supply voltages can vary from one path to another.

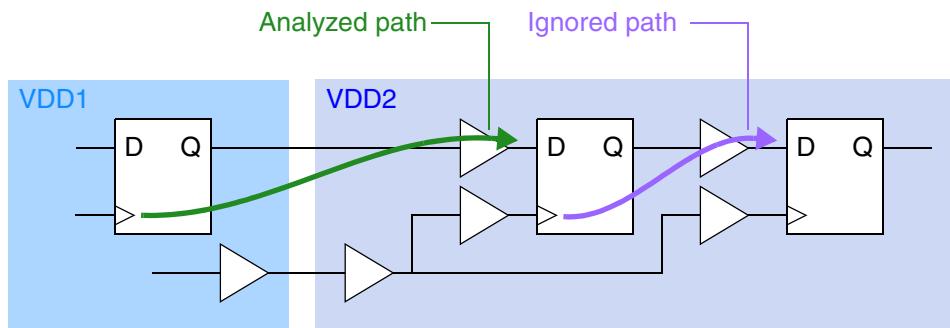
To perform simultaneous multivoltage analysis during path-based analysis, set the `timing_enable_cross_voltage_domain_analysis` variable to `true`. The default is `false`, which results in conventional worst-case timing analysis.

When the variable is set to `true`, during the next timing update, PrimeTime identifies paths that traverse multiple power domains and subsequently limits the reporting to only those paths. If a path-based analysis is then performed by using the `-pba_mode path` option of the `report_timing` command, PrimeTime performs an efficient path-based recalculation of the cross-domain paths. This analysis finds the worst-case voltage, as defined by the `set_voltage` command and UPF commands, for each domain crossed by each path, while eliminating the pessimism that occurs in standard on-chip variation analysis.

During simultaneous multivoltage analysis, the tool performs the following functions:

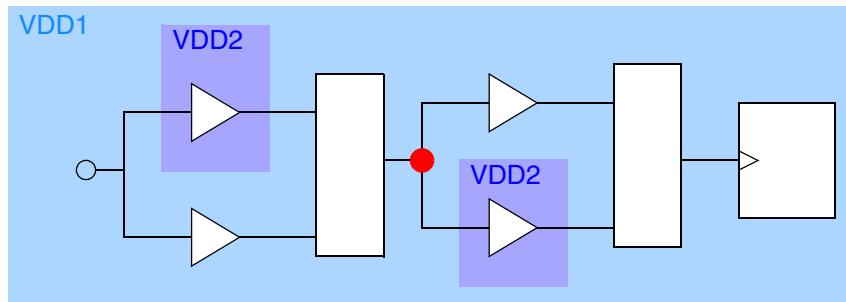
- Identifies and ignores paths that cross power domains before the CRPR common point, as shown in [Figure 13-6](#)

Figure 13-6 Analyzing Only Paths Affected by Domain Crossings



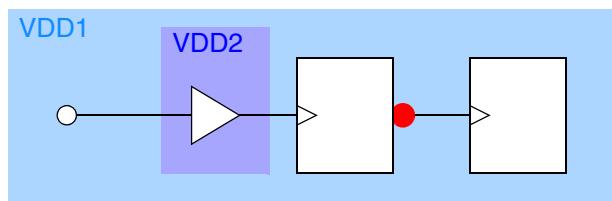
- Identifies reconvergent clock paths, as shown in [Figure 13-7](#)

Figure 13-7 Clock Path Reconverges



- Improves pessimism reduction for sequential clock networks when the path crosses power domains, as shown in [Figure 13-8](#)

Figure 13-8 Sequential Clock Network Crosses Power Domains



Simultaneous Multivoltage Analysis Usage Flow

During simultaneous multivoltage analysis, by default, the delay of each cell at each voltage level is determined by the existing multivoltage delay calculation methodology of PrimeTime. You get the most accurate results by using CCS libraries with library scaling groups invoked with the `define_scaling_lib_group` command. If these models are not available, PrimeTime uses the available scalable polynomial delay models (SPDM) or NLDM models.

Simultaneous multivoltage analysis is intended to analyze and report only the timing paths that cross power domain boundaries. Before you use this feature, it is suggested that you

follow your usual methodology for fixing within-domain timing paths. During these runs, you can ignore any cross-domain timing paths. After this is complete, execute the simultaneous multivoltage analysis run to comprehensively analyze the cross-domain paths.

When libraries with multivoltage characterization data are available, this is the typical usage flow for simultaneous multivoltage analysis of cross-domain paths, after all within-domain violations have been fixed:

```
set search_path
set link_path "* mylib_1V.db"
read_verilog ...
link_design
read_parasitics
load_upf ...
read_sdc ... #includes set_voltage commands
#derate margins not including variation due to voltage level settings
set_timing_derate -early 0.92
set_timing_derate -late 1.08
define_scaling_lib_group "mylib_1V.db mylib_0.8V.db \
                         mylib_1.2V.db mylib_1.4V.db"
# invoke accurate multivoltage analysis for cross-domain paths
set timing_enable_cross_voltage_domain_analysis true
# conservative cross-domain-only timing report:
report_timing ...
# path-based cross-domain timing report with reduced pessimism:
report_timing -pba_mode path ...
```

Analysis Using Cross-Domain Derating

If you do not have any reliable library characterization information with respect to voltage variation, you can use cross-domain derating to analyze the effect of different supply voltages. To do this, specify a derating factor adjustment value with the `set_cross_voltage_domain_analysis_guardband` command. For example,

```
pt_shell> set_cross_voltage_domain_analysis_guardband -late 0.1
```

This derating is added to any ordinary derating specified with the `set_timing_derate` command and advanced on-chip variation (AOCV) analysis invoked with the `timing_aocvm_enable_analysis` variable. In this example, if the regular late derating factor set by the `set_timing_derate` command is 1.05, an additional late guard band of 0.1 makes the final late derating factor $1.05 + 0.1 = 1.15$. The derating factor of 1.15 models the combined timing variation due to on-chip variation effects and varying voltage for timing paths that cross power domains.

The additional guard band of 0.1 is intended to model the timing effects of supply voltage variation in the absence of library characterization data for this voltage variation. If you have multivoltage library characterization data, you do not need the `set_cross_voltage_domain_analysis_guardband` command. However, PrimeTime still accepts the command and applies additional derating if cross-domain analysis is enabled.

When you do not have multivoltage library characterization data, this is the typical usage flow for simultaneous multivoltage analysis of cross-domain paths using derating, after all within-domain violations have been fixed:

```
set search_path
set link_path
read_verilog ...
link_design
read_parasitics
load_upf ...
read_sdc #includes set_voltage commands
# example margins, not including V_OCV
set_timing_derate -early 0.92
set_timing_derate -late 1.08
# invoke accurate multivoltage analysis for cross-domain paths
set timing_enable_cross_voltage_domain_analysis true
# specify derating for voltage supply settings, +/- 10%
set_cross_voltage_domain_analysis_guardband -early 0.9
set_cross_voltage_domain_analysis_guardband -late 1.1
# conservative cross-domain-only timing report with low-high derating:
report_timing ...
# path-based cross-domain timing report with reduced pessimism:
report_timing -pba_mode path ...
```

IR Drop Annotation

You can annotate supply voltages on the PG pins of cells, and thereby model the effects of IR drop on timing and noise, by using the `-cell` and `-pg_pin_name` options of the `set_voltage` command. For example, this command sets the supply voltage to 1.10 volts on the VDD1 pins of all U5* cells.

```
pt_shell> set_voltage 1.10 -cell [get_cells U5*] -pg_pin_name VDD1
```

The `-cell` and `-pg_pin_name` options, if used in the `set_voltage` command, must be used together. These two options require a PrimeTime SI license. You do not need a PrimeTime SI license for the other options of the `set_voltage` command, such as the `-object_list` option used to set voltage on power supply nets.

Signal Level Checking With IR Drop

PrimeTime checks for mismatch in voltage levels between cells using different supply voltages when you use the `check_timing` command:

```
pt_shell> check_timing -include signal_level
```

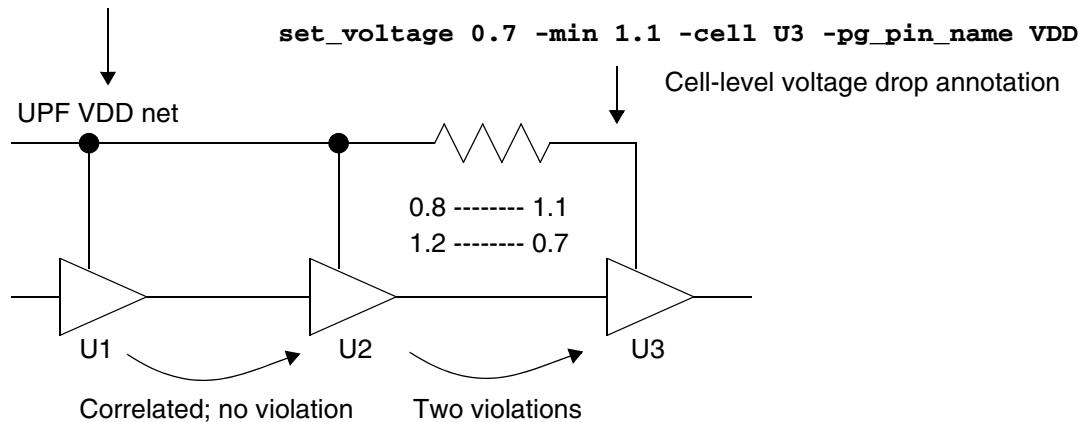
The signal level check correctly considers the case where minimum and maximum voltages are set on voltage rails that supply both the cells being compared. Suppose, however, that IR drop is annotated on a cell using commands similar to the following:

```
pt_shell> set_voltage 0.8 -min 1.2 VDD    # corner voltages
pt_shell> set_voltage 0.7 -min 1.1 -cell U3 -pg_pin_name VDD  # IR drop
```

This corresponds to the circuit shown in [Figure 13-9](#).

Figure 13-9 Signal Level Checking With Cell-Level Voltage Annotation

```
set_voltage 0.8 -min 1.2 VDD
```



In this case, PrimeTime SI assumes that the supply voltage annotated at the cell level is not correlated with the main supply voltage. It considers the full worst-case differences between the driver and receiver, which is to compare 0.8 to 1.1 volts and 1.2 to 0.7 volts, thus triggering two mismatch violations.

14

Signal Integrity Analysis

PrimeTime SI is an optional tool that adds crosstalk analysis capabilities to the PrimeTime static timing analyzer. PrimeTime SI calculates the timing effects of cross-coupled capacitors between nets and includes the resulting delay changes in the PrimeTime analysis reports. It also calculates the logic effects of crosstalk noise and reports conditions that could lead to functional failure.

PrimeTime SI is much easier, faster, and more thorough than using a circuit simulator such as SPICE. Instead of analyzing only a single path or a few paths for crosstalk effects, PrimeTime SI analyzes the whole circuit using the familiar PrimeTime analysis flow. To run PrimeTime SI, you need a PrimeTime SI license.

To learn about crosstalk analysis and PrimeTime SI, see:

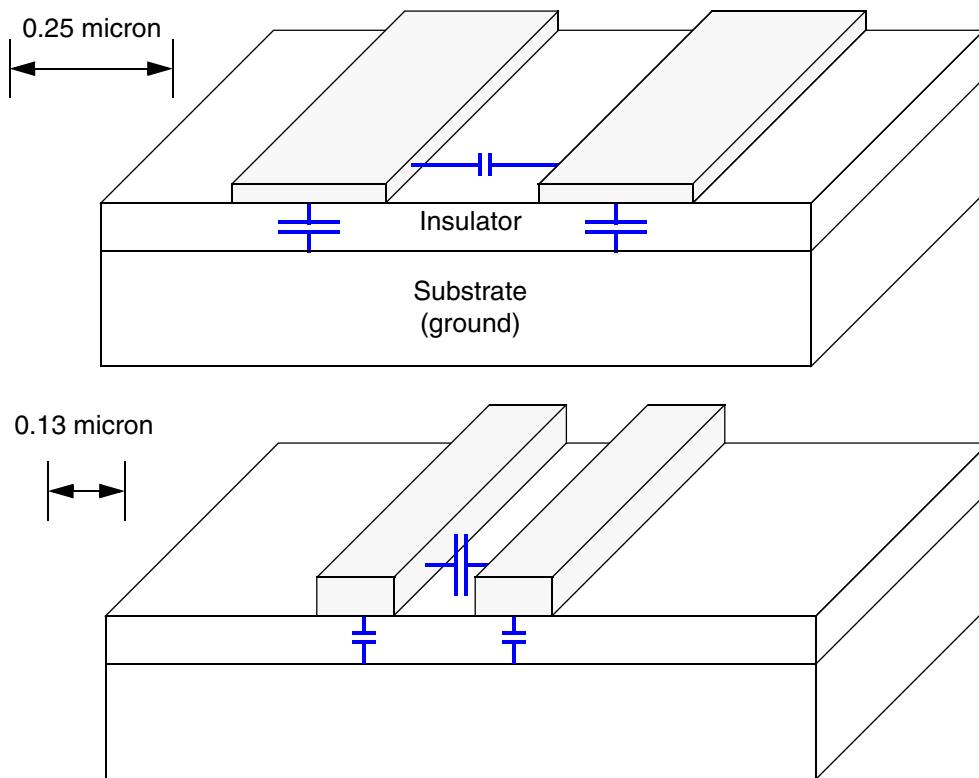
- [Overview of Signal Integrity and Crosstalk](#)
- [Aggressor and Victim Nets](#)
- [Crosstalk Delay Analysis](#)
- [Static Noise Analysis](#)

Overview of Signal Integrity and Crosstalk

Signal integrity is the ability of an electrical signal to carry information reliably and resist the effects of high-frequency electromagnetic interference from nearby signals. Crosstalk is the undesirable electrical interaction between two or more physically adjacent nets due to capacitive cross-coupling. As integrated circuit technologies advance toward smaller geometries, crosstalk effects become increasingly important compared to cell delays and net delays.

Figure 14-1 shows an enlarged view of two parallel metal interconnections in an integrated circuit, first for a 0.25-micron technology and then for a 0.13-micron technology.

Figure 14-1 Cross-Coupling Capacitance Increases With Smaller Feature Sizes



As circuit geometries become smaller, wire interconnections become closer together and taller, thus increasing the cross-coupling capacitance between nets. At the same time, parasitic capacitance to the substrate becomes less as interconnections become narrower, and cell delays are reduced as transistors become smaller.

With circuit geometries at 0.25 micron and above, substrate capacitance is usually the dominant effect. However, with geometries at 0.18 micron and below, the coupling

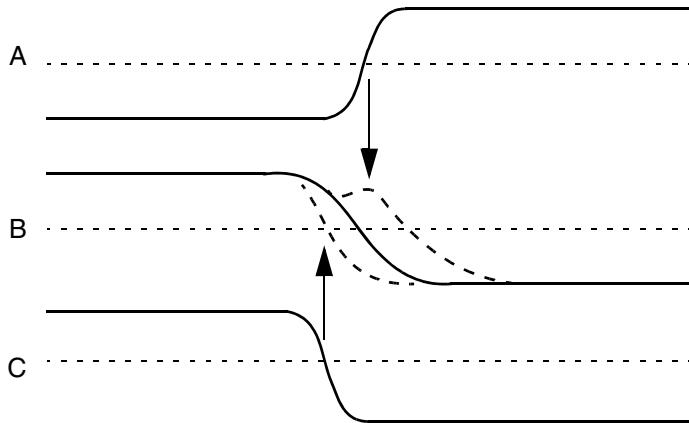
capacitance between nets becomes significant, making crosstalk analysis increasingly important for accurate timing analysis.

PrimeTime SI has the ability to analyze and report two major types of crosstalk effects: delay and static noise. You can choose to have PrimeTime calculate crosstalk delay effects, crosstalk noise effects, or both.

Crosstalk Delay Effects

Crosstalk can affect signal delays by changing the times at which signal transitions occur. For example, [Figure 14-2](#) shows the signal waveforms on cross-coupled nets A, B, and C.

Figure 14-2 Transition Slowdown or Speedup Caused by Crosstalk



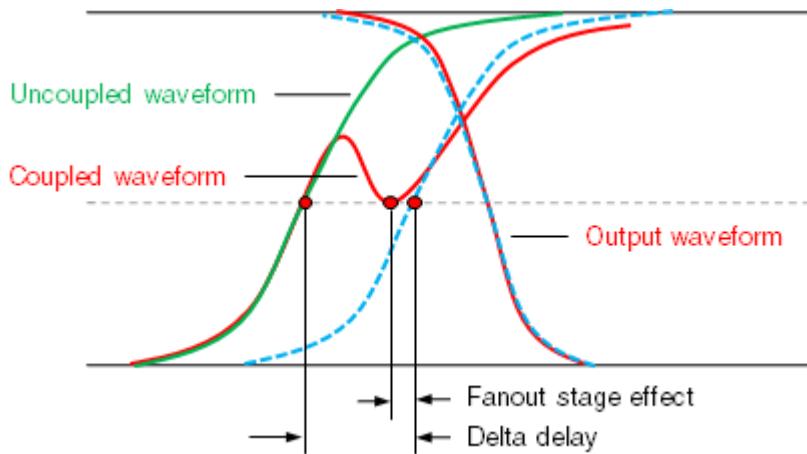
Because of capacitive cross-coupling, the transitions on net A and net C can affect the time at which the transition occurs on net B. A rising-edge transition on net A at the time shown in [Figure 14-2](#) can cause the transition to occur later on net B, possibly contributing to a setup violation for a path containing B. Similarly, a falling-edge transition on net C can cause the transition to occur earlier on net B, possibly contributing to a hold violation for a path containing B.

PrimeTime SI determines the worst-case changes in delay values and uses this additional information to calculate and report total slack values. It also reports the locations and amounts of crosstalk delays so that you can change the design or the layout to reduce crosstalk effects at critical points.

Delta Delay and Fanout Stage Effect

Crosstalk effects distort a switching waveform, which adds delay to the propagated waveforms of the fanout stages, as shown in [Figure 14-3](#).

Figure 14-3 Delta Delay Includes the Fanout Stage Effect

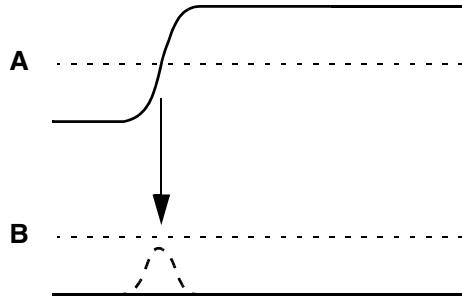


PrimeTime SI calculates the *delta delay*, which is the amount of additional delay induced by crosstalk on a switching net. The *fanout stage effect* represents the delay propagated to the fanout stages. PrimeTime SI includes this fanout stage effect as part of the delta delay.

Crosstalk Noise Effects

PrimeTime SI also determines the logic effects of crosstalk noise on steady-state nets. [Figure 14-4](#) shows an example of a noise bump due to crosstalk on cross-coupled nets A and B.

Figure 14-4 Noise Bump Due to Crosstalk



Net B should be constant at logic 0, but the rising edge on net A causes a noise bump or glitch on net B. If the bump is sufficiently large and wide, it can cause an incorrect logic value to be propagated to the next gate in the path containing net B.

PrimeTime SI considers these effects and determines where crosstalk noise bumps have the largest effects. It reports the locations of potential problems so that they can be fixed.

Aggressor and Victim Nets

A net that receives undesirable cross-coupling effects from a nearby net is called a *victim net*. A net that causes these effects in a victim net is called an *aggressor net*. Note that an aggressor net can itself be a victim net; and a victim net can also be an aggressor net. The terms aggressor and victim refer to the relationship between two nets being analyzed.

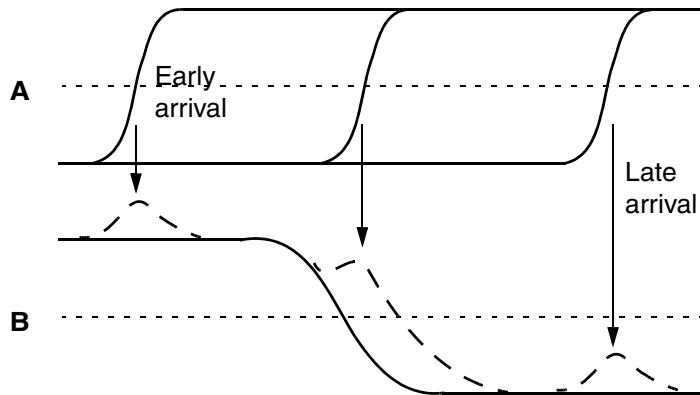
The timing effect of an aggressor net on a victim net depends on several factors:

- The amount of cross-coupled capacitance
- The relative times and slew rates of the signal transitions
- The switching directions (rising, falling)
- The combination of effects from multiple aggressor nets on a single victim net

PrimeTime SI takes all of these factors into account when it calculates crosstalk effects. It saves a lot of computation time by ignoring situations where the cross-coupling capacitors are too small to have an effect and by ignoring cases where the transition times on cross-coupled nets cannot overlap.

[Figure 14-5](#) shows the importance of timing considerations for calculating crosstalk effects. The aggressor signal A has a range of possible arrival times, from early to late.

Figure 14-5 Effects of Crosstalk at Different Arrival Times



As shown in [Figure 14-5](#), if the transition on A occurs at about the same time as the transition on B, it could cause the transition on B to occur later, possibly contributing to a setup violation; otherwise, it could cause the transition to occur earlier, possibly contributing to a hold violation.

If the transition on A occurs at an early time, it induces an upward bump or glitch on net B before the transition on B, which has no effect on the timing of signal B. However, a

sufficiently large bump can cause unintended current flow by forward-biasing a pass transistor. PrimeTime SI reports the worst-case occurrences of noise bumps.

Similarly, if the transition on A occurs at a late time, it induces a bump on B after the transition on B, also with no effect on the timing of signal B. However, a sufficiently large bump can cause a change in the logic value of the net, which can be propagated down the timing path. PrimeTime SI reports occurrences of bumps that cause incorrect logic values to be propagated.

Timing Windows and Crosstalk Delay Analysis

PrimeTime offers two analysis modes with respect to operating conditions: single and on-chip variation. PrimeTime SI uses the on-chip variation mode to derive the timing window relationships between aggressor nets and victim nets.

Using the on-chip variation mode, PrimeTime SI finds the earliest and the latest arrival times for each aggressor net and victim net. The range of switching times, from earliest to latest arrival, defines a timing window for the aggressor net and another timing window for the victim net. Crosstalk timing effects can occur only when the aggressor and victim timing windows overlap.

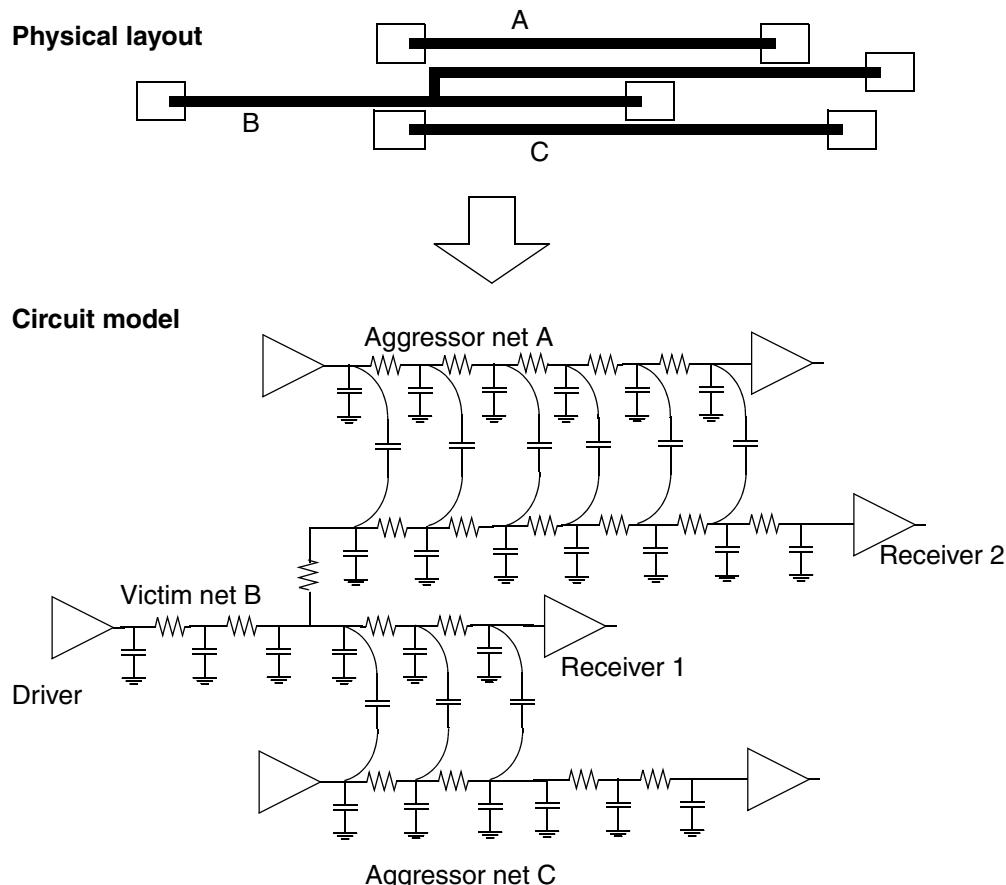
By default, PrimeTime SI performs crosstalk analysis using two iterations. During the first iteration, it ignores the timing windows and assumes that all transitions can occur at any time. This results in pessimistic crosstalk delay values. During the second and subsequent iterations, PrimeTime SI considers the timing windows and eliminates some victim-aggressor relationships from consideration, based on the lack of overlap between the applicable timing windows.

When an overlap occurs, PrimeTime SI calculates the effect of a transition occurring on the aggressor net at the same time as a transition on the victim net. The analysis takes into account the drive strengths and coupling characteristics of the two nets.

Cross-Coupling Models

[Figure 14-6](#) shows the physical layout for a small portion of an integrated circuit, together with a detailed model of the circuit that includes cross-coupled capacitance. Each physical interconnection has some distributed resistance along the conductor and some parasitic capacitance to the substrate (ground) and to adjacent nets. The model divides each net into subnets and represents the distributed resistance and capacitance as a set of discrete resistors and capacitors.

Figure 14-6 Detailed Model of Cross-Coupled Nets



A detailed model such as this can provide a very accurate prediction of crosstalk effects in simulation. For an actual integrated circuit, however, a model might have too many circuit elements to process in a practical amount of time. Given a reasonably accurate (but sufficiently simple) network of cross-coupled capacitors from an external tool, PrimeTime SI can obtain accurate crosstalk analysis results in a reasonable amount of time.

Crosstalk Delay Analysis

To learn the basics of using PrimeTime SI for crosstalk delay analysis, see

- [Performing Crosstalk Delay Analysis](#)
- [How PrimeTime SI Operates](#)
- [Usage Guidelines](#)

- [Timing Reports](#)
- [Reporting Crosstalk Settings](#)
- [Double-Switching Detection](#)

Performing Crosstalk Delay Analysis

Crosstalk delay analysis with PrimeTime SI uses the same command set, libraries, and Tcl scripts as ordinary PrimeTime analysis. To use PrimeTime SI, you only need to perform the following additional steps:

1. Enable PrimeTime SI:

```
pt_shell> set_app_var si_enable_analysis true
```

2. Back-annotate the design with cross-coupling capacitance information in a Standard Parasitic Exchange Format (SPEF) or Synopsys Binary Parasitic Format (SBPF) file:

```
pt_shell> read_parasitics -keep_capacitive_coupling file_name.spf
```

or

```
pt_shell> read_parasitics -keep_capacitive_coupling \  
-format SBPF file_name.spf
```

3. (Optional) Specify the parameters that control the speed and performance of the crosstalk portion of the analysis. These parameters are controlled by a set of variables. See [PrimeTime SI Variables](#).
4. If significant crosstalk effects are apparent in the timing report, debug or correct the timing problem. To assist in this task, the PrimeTime SI graphical user interface (GUI) lets you generate histograms of crosstalk delays and induced bump voltages. You can also create your own Tcl scripts to extract the crosstalk attributes from the design database, and then generate your own custom reports or histograms.

Here is an example of a script that uses crosstalk analysis, with the crosstalk-specific items shown in boldface:

```
set_operating_conditions -analysis_type on_chip_variation  
set_app_var si_enable_analysis TRUE  
read_db ./test1.db  
current_design test1  
link_design  
read_parasitics -keep_capacitive_coupling SPEF.spf  
create_clock -period 5.0 clock  
check_timing -include { no_driving_cell ideal_clocks \  
partial_input_delay unexpandable_clocks }  
report_timing  
report_si_bottleneck  
report_delay_calculation -crosstalk -from pin -to pin
```

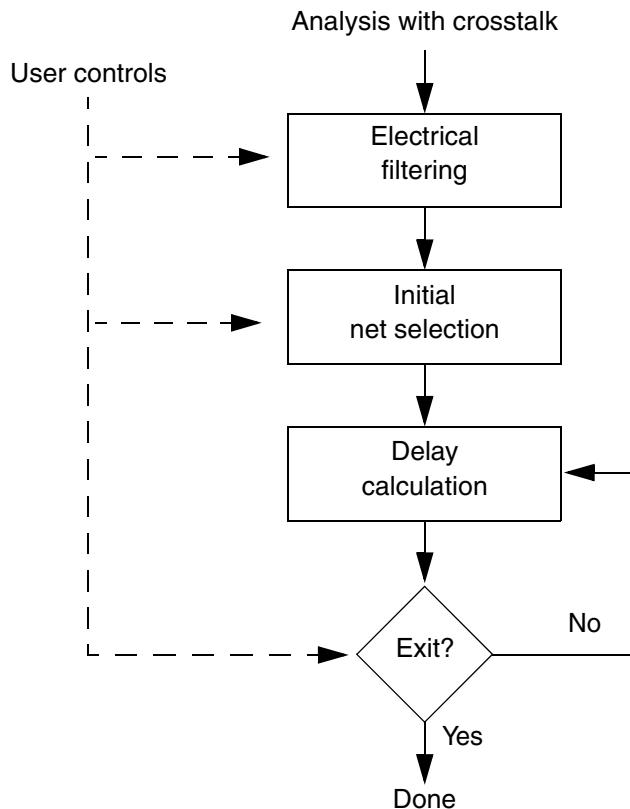
The `set_operating_conditions` command sets the analysis type to `on_chip_variation`, which is necessary to allow PrimeTime SI to correctly handle the min-max timing window relationships. If you do not specify this analysis type explicitly, PrimeTime SI automatically switches to that mode.

When reading the Standard Parasitic Exchange Format (SPEF), you must ensure that the coupling capacitances in the SPEF file are symmetric. First, read in the design, and then use both the `-syntax_only` and `-keep_capacitive_coupling` options of the `read_parasitics` command. PrimeTime issues warning messages if asymmetric couplings are found. Confirm that the SPEF files contain valid coupling capacitances before proceeding.

How PrimeTime SI Operates

PrimeTime SI performs crosstalk analysis in conjunction with your regular PrimeTime analysis flow. With crosstalk analysis enabled, when you update the timing (for example, by using the `update_timing` or `report_timing` command), PrimeTime SI performs the steps shown in [Figure 14-7](#).

Figure 14-7 PrimeTime SI Crosstalk Analysis Flow



The first step is called electrical filtering. This means removing from consideration the aggressor nets whose effects are too small to be significant, based on the calculated sizes of bump voltages on the victim nets. You can specify the threshold level that determines which aggressor nets are filtered.

After filtering, PrimeTime SI selects the initial set of nets to be analyzed for crosstalk effects from those not already eliminated by filtering. You can optionally specify that certain nets be included in, or excluded from, this initial selection set.

The next step is to perform delay calculation, taking into account the crosstalk effects on the selected nets. This step is just like ordinary timing analysis, but with the addition of crosstalk considerations.

Crosstalk analysis is an iterative process, taking multiple passes through the delay calculation step, to obtain accurate results. For the initial delay calculation (using the initial set of selected nets), PrimeTime SI uses a conservative model that does not consider timing windows. In other words, PrimeTime SI assumes that every aggressor net can have a worst-type transition (rising or falling) at the worst possible time, causing the worst possible slowdown or speedup of transitions on the victim net. The purpose of this behavior is to obtain the worst-case delay values, which are later refined by PrimeTime SI in the next analysis iteration.

In the second and subsequent delay calculation iterations, PrimeTime SI considers the possible times that victim transitions can occur and their directions (rising or falling), and removes from consideration any crosstalk delays that can never occur, based on the separation in time between the aggressor and victim transitions or the direction of the aggressor transition. The result is a more accurate, less pessimistic analysis of worst-case effects.

By default, PrimeTime SI exits from the loop upon completion of the second iteration, which typically provides good results in a reasonable amount of time.

You can interrupt any crosstalk analysis iteration by pressing Ctrl+C. PrimeTime SI finishes the current iteration, exits from the loop, and reports diagnostic information regarding the state of the analysis at the end of the iteration.

PrimeTime SI Variables

Table 14-1 lists the variables that control crosstalk analysis. To enable crosstalk analysis, set the `si_enable_analysis` variable to `true`.

Table 14-1 PrimeTime SI Variables

Variable	Default setting
<code>delay_calc_waveform_analysis_mode</code>	<code>disabled</code>
<code>si_analysis_logical_correlation_mode</code>	<code>true</code>
<code>si_ccs_aggressor_alignment_mode</code>	<code>lookahead</code>
<code>si_enable_analysis</code>	<code>false</code>
<code>si_enable_multi_valued_coupling_capacitance</code>	<code>false</code>
<code>si_filter_accum_aggr_noise_peak_ratio</code>	0.03
<code>si_filter_per_aggr_noise_peak_ratio</code>	0.01
<code>si_ilm_keep_si_user_excluded_aggressors</code>	<code>false</code>
<code>si_noise_composite_aggr_mode</code>	<code>disabled</code>
<code>si_noise_endpoint_height_threshold_ratio</code>	0.75
<code>si_noise_limit_propagation_ratio</code>	0.75
<code>si_noise_slack_skip_disabled_arcs</code>	<code>false</code>
<code>si_noise_update_status_level</code>	<code>none</code>
<code>si_use_driving_cell_derate_for_delta_delay</code>	<code>false</code>
<code>si_xtalk_composite_aggr_mode</code>	<code>disabled</code>
<code>si_xtalk_composite_aggr_noise_peak_ratio</code>	0.01
<code>si_xtalk_composite_aggr_quantile_high_pct</code>	99.73
<code>si_xtalk_delay_analysis_mode</code>	<code>all_paths</code>
<code>si_xtalk_double_switching_mode</code>	<code>disabled</code>

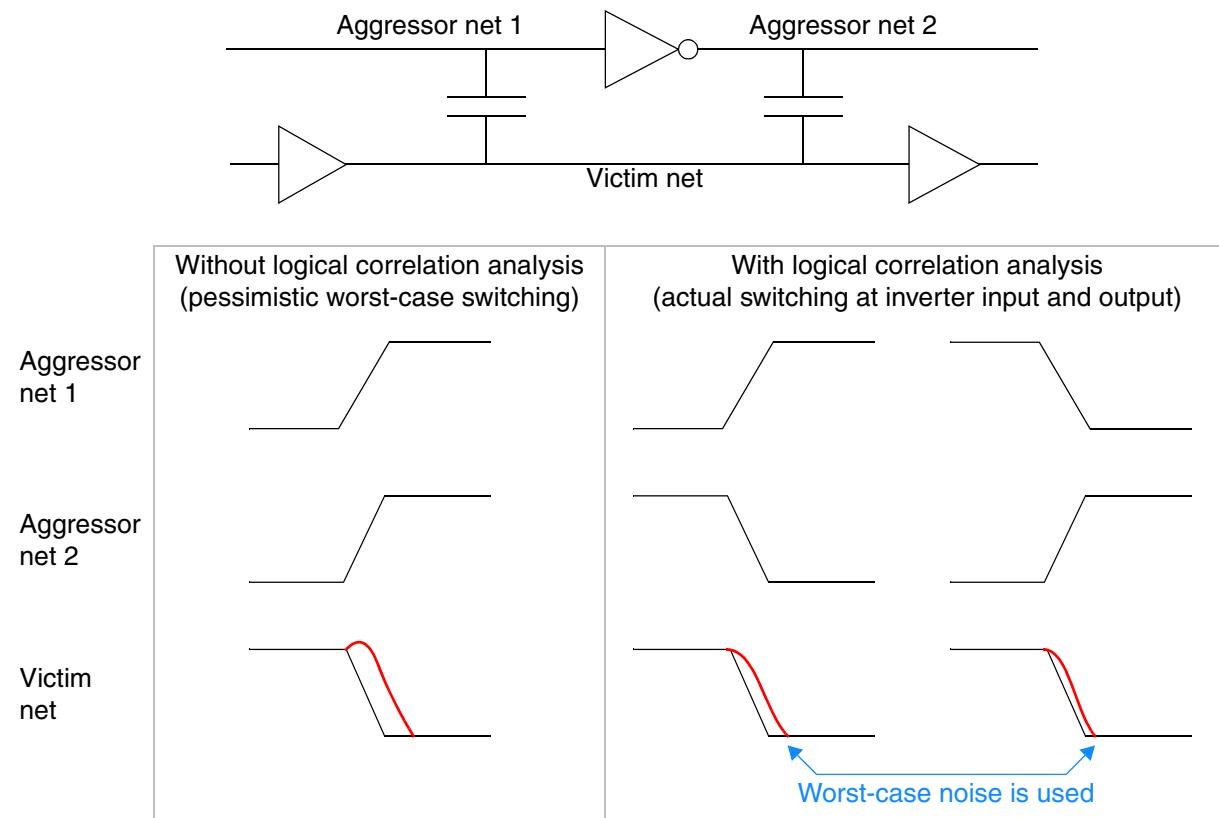
Table 14-1 PrimeTime SI Variables (Continued)

Variable	Default setting
si_xtalk_exit_on_max_iteration_count	2
si_xtalk_max_transition_mode	uncoupled

Logical Correlation

In a conservative analysis, the analysis tool assumes that all aggressor nets can switch together in a direction to cause a worst-case slowdown or speedup of a transition on the victim net. In some cases, due to a logical relationship between the signals, the aggressor nets cannot actually switch together in the same direction. [Figure 14-8](#) shows an example of this situation.

By default, PrimeTime SI considers the logical relationships between multiple aggressor nets where buffers and inverters are used, thus providing a more accurate (less pessimistic) analysis of multiple aggressor nets. Consideration of the logical correlation between different nets requires CPU resources.

Figure 14-8 Logical Correlation

For a faster but more pessimistic analysis, you can disable logical correlation consideration. To do so, set the `si_analysis_.logical_correlation_mode` variable to `false`.

Electrical Filtering

To achieve accurate results in a reasonable amount of time, PrimeTime SI filters (removes from consideration) aggressor nets that are considered to have too small an effect on the final results. When filtering occurs, the aggressor net and the coupling capacitors connected to it are not considered for analysis between that victim net and aggressor net. If the bump height contribution of an aggressor on its victim net is very small (less than 0.00001 of the victim's nominal voltage), this aggressor is automatically filtered.

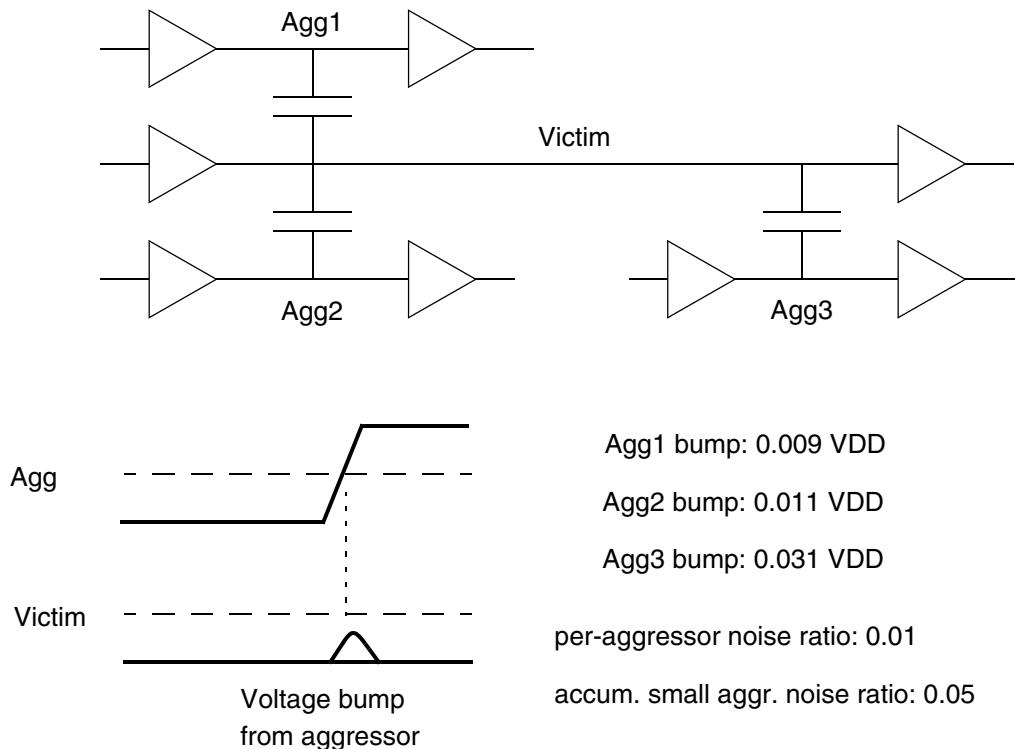
Filtering eliminates aggressors based on the size of the voltage bump induced on the victim net by the aggressor net. The bump sizes depend on the cross-coupling capacitance values, drive strengths, and resistance values in the nets. An aggressor net is filtered if the peak voltage of the noise bump induced on the victim net, divided by VDD (the power supply

voltage), is less than the value specified by the `si_filter_per_aggr_noise_peak_ratio` variable. By default, this variable is set to 0.01.

In addition, if a combination of smaller aggressors is below a different, larger threshold, all of those smaller aggressors are filtered. This threshold is set by the `si_filter_accum_aggr_noise_peak_ratio` variable. If the combined height of smaller noise bumps, divided by VDD, is less than this variable setting, all of those aggressors are removed from consideration for that set of analysis conditions. By default, the `si_filter_accum_aggr_noise_peak_ratio` variable is set to 0.03.

[Figure 14-9](#) shows how PrimeTime SI compares voltage bump sizes for a case with three aggressor nets. It first calculates the voltage bumps induced on the victim net by transitions on each aggressor net. It then considers the bump sizes in order. For this example, assume that the `si_filter_accum_aggr_noise_peak_ratio` variable is set to 0.05.

Figure 14-9 Voltage Bumps From Multiple Aggressors



PrimeTime SI filters out aggressor 1 immediately because the bump size, 0.009, is below the per-aggressor threshold, 0.01. PrimeTime SI then considers the combined bumps of smaller aggressors. The combination of aggressor 1 and 2 bump heights is 0.02, which is below the accumulated small aggressor threshold of 0.05, so both of those aggressors are filtered out. However, the combination of all three aggressor bump heights is 0.051, which is above the accumulated small aggressor threshold, so aggressor 3 is not filtered out, even though it is below the threshold by itself.

In summary, aggressor 1 is filtered due to the per-aggressor threshold alone, while both aggressors 1 and 2 are filtered out, but not aggressor 3, due to the accumulated small aggressor threshold.

You can set the thresholds higher for more filtering and less runtime, or lower for less filtering and increased accuracy.

Usage Guidelines

Most of the PrimeTime SI variables let you trade analysis accuracy against execution speed. For example, you can get more accuracy by running more delay calculation iterations, at the cost of more runtime. The following suggestions and guidelines help ensure reasonable accuracy with a reasonable runtime.

Preparing to Run Crosstalk Analysis

First make sure that your test design is well-constrained and passes normal static timing analysis (without crosstalk analysis enabled). There should be no timing violations.

Capacitive Coupling Data

A good crosstalk analysis depends on getting an accurate and reasonably simple set of cross-coupling capacitors.

If your extraction tool supports the filtering of small capacitors based on a threshold, it might be more efficient to let the extraction tool rather than PrimeTime SI do electrical filtering. To further trade accuracy for simplicity, you might consider limiting the number of coupling capacitors per aggressor-victim relationship.

PrimeTime SI ignores any cross-coupling capacitance between a net and itself. If possible, configure your extraction tool to suppress generation of such self-coupling capacitors.

When you read in the capacitive coupling data with the `read_parasitics` command, remember to use the `-keep_capacitive_coupling` option to retain the data.

For more information, see [Reading Parasitic Files](#).

Operating Conditions

PrimeTime SI uses on-chip variation of operating conditions to find the arrival window for each victim net and aggressor net. It automatically switches the analysis mode to `on_chip_variation` for crosstalk analysis if it is not already set to that mode.

These are the consequences of automatic switching to `on_chip_variation` mode:

- If you were already using `on_chip_variation` mode for noncrosstalk analysis before invoking PrimeTime SI, crosstalk analysis continues in that mode.
- If you were using the single operating condition mode, crosstalk analysis occurs in the `on_chip_variation` mode with the single operating condition setting for both minimum and maximum analysis. This is equivalent to using a single operating condition.

Using `check_timing`

The `check_timing` command can check for several conditions related to crosstalk analysis, making it easier to detect conditions that can lead to inaccurate crosstalk analysis results. After you set the constraints and before you start an analysis with the `update_timing` or `report_timing` command, run the `check_timing` command.

The following types of checking are specific to crosstalk analysis:

- `no_driving_cell` – The `check_timing` command reports any input port that does not have a driving cell and does not have case analysis set on it. When no driving cell is specified, that net is assigned a strong driver for modeling aggressor effects, which can be pessimistic.
- `ideal_clocks` – The `check_timing` command reports any clock networks that are ideal (not propagated). For accurate determination of crosstalk effects, the design should have a valid clock tree and the clocks should be propagated.
- `partial_input_delay` – The `check_timing` command reports any inputs that have only the minimum or only the maximum delay defined with the `set_input_delay` command. To accurately determine timing windows, PrimeTime SI needs both the earliest and latest arrival times at the inputs.
- `unexpandable_clocks` – The `check_timing` command reports any clocks that have not been expanded to a common time base. For accurate alignment of arrival windows, all of the synchronous and active clocks of different frequencies must be expanded to a common time base.

With the exception of `ideal_clocks`, crosstalk-related checks are on by default. To enable `ideal_clocks`, you can either set the `timing_check_defaults` variable or use the `-include` option of the `check_timing` command. For example,

```
pt_shell> check_timing -include { ideal_clocks }
```

Including or Excluding Specific Nets From Crosstalk Analysis

To include or exclude particular nets from crosstalk analysis, use these commands:

- `set_si_delay_analysis` – Includes or excludes specified nets for crosstalk delay analysis.

- `set_si_noise_analysis` – Includes or excludes specified nets for crosstalk noise analysis.
- `set_si_aggressor_exclusion` – Excludes aggressor-to-aggressor nets that switch in the same direction.
- `set_coupling_separation` – Excludes nets or net pairs from crosstalk delay and crosstalk noise analysis.

Use the `set_si_delay_analysis` command to include or exclude nets for crosstalk delay analysis in the following ways:

- Set specific nets to use infinite arrival windows
- Exclude specific nets as aggressors
- Exclude specific nets as victims
- Exclude specific aggressor-victim relationships between net pairs

If there is a conflict between the settings of the `set_coupling_separation`, `set_si_delay_analysis`, or `set_si_noise_analysis` commands, the `set_coupling_separation` command has precedence.

Excluding Rising/Falling Edges or Setup/Hold Analysis

To exclude only rising or only falling edges at the victim net, use the `set_si_delay_analysis` command with the `-rise` or `-fall` option. To exclude only the maximum (setup) or only minimum (hold) path analysis, use the `set_si_delay_analysis` command with the `-max` or `-min` option.

Excluding Clock Nets

If there are clock signal delays in your design that are not significantly affected by crosstalk, you can exclude them from crosstalk delay analysis. The following example excludes the clock net CLK1 from consideration as a victim net:

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets CLK1]
```

In this case, PrimeTime SI excludes the net CLK1 as a potential victim net for crosstalk delay analysis, thereby reducing the analysis runtime. However, CLK1 can still be considered as an aggressor net.

Excluding Nets from Crosstalk Noise Analysis

To exclude specific nets from crosstalk noise analysis, use the `set_si_noise_analysis` command. The following example excludes the clock net CLK1 from consideration as a victim net for crosstalk noise analysis:

```
pt_shell> set_si_noise_analysis -exclude -victims [get_nets CLK1]
```

Excluding Analysis of Noise Bumps

To exclude the analysis of above-high and below-low noise bumps for the CLK1 net, use these commands:

```
pt_shell> set_si_noise_analysis -exclude [get_nets CLK1] -above -high  
pt_shell> set_si_noise_analysis -exclude [get_nets CLK1] -below -low
```

Excluding Quiet Aggressor Nets

To mark specific nets as quiet (nonswitching) aggressors and exclude them from timing and noise analysis, use the `set_case_analysis` command on these nets. These quiet aggressor nets are not considered to be effective aggressors of any victim net.

Excluding Aggressor-Victim Pairs

To exclude pairs of specific aggressor-victim nets, use the `-exclude` option with both the `-victims` and `-aggressors` options.

For example, suppose that you know that the scan clock signals (`SCN_CLK_*`) and clock network signals (`CLK_NET_*`) in your design do not affect each other for timing. To exclude these signals from consideration:

```
pt_shell> set_si_delay_analysis -exclude -victims [get_nets CLK_NET_*] \  
          -aggressors [get_nets SCN_CLK_*]  
  
pt_shell> set_si_delay_analysis -exclude -victims [get_nets SCN_CLK_*] \  
          -aggressors [get_nets CLK_NET_*]
```

The first of these two commands excludes any `SCN_CLK_*` signal as an aggressor to any `CLK_NET_*` signal as a victim during crosstalk delay analysis. The second command excludes the aggressor-victim relationship of these signals in the opposite direction. However, note that any of these signals can still be considered aggressors or victims relative to signals not specified by these commands. For example, `CLK_NET1` can still be considered an aggressor to `CLK_NET2` as a victim.

Excluding Aggressor-to-Aggressor Nets

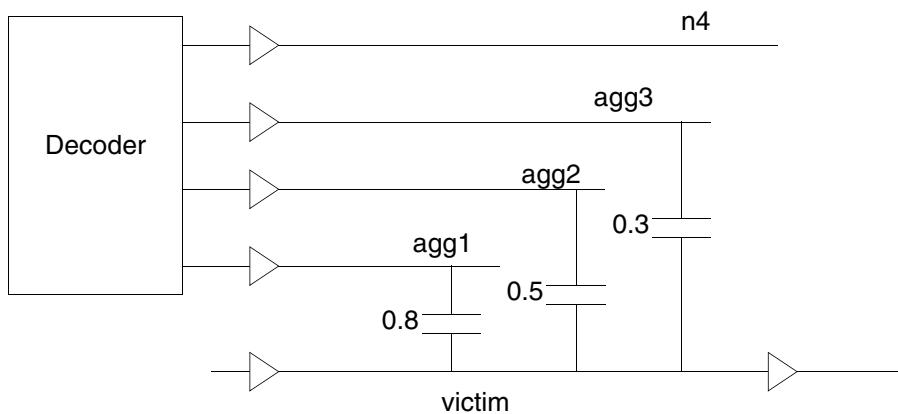
By default, when multiple aggressor arrival windows overlap with a victim transition window, PrimeTime SI considers the worst case of multiple aggressor transitions occurring in the same direction at the same time. In some cases, however, the logical relationship between the aggressor signals prevents simultaneous switching of the aggressors in the same direction.

You can reduce pessimism in crosstalk analysis by specifying groups of aggressor nets as exclusive during worst-case alignment. Exclusive aggressor nets are nets among which only a specified number of aggressors can switch simultaneously in the same direction. The nets can be exclusive for rise, fall, or both. Exclusive for rise means that only the specified

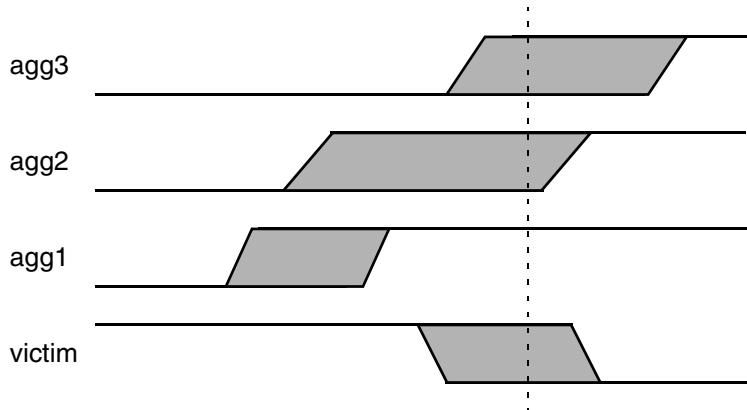
maximum number of aggressors within the exclusive set can rise to affect a victim, and all the other aggressors within the exclusive set are considered quiet (not switching).

PrimeTime SI considers these exclusive aggressor groups during crosstalk delay and noise analysis. [Figure 14-10](#) is an example of one-hot signal nets, where only one net is active at a given time; therefore, only one net can have a value of 1, while the remaining nets have a value of 0. Here, the aggressors agg1, agg2, and agg3 can be defined as an exclusive group for both the rising and falling directions.

Figure 14-10 One-Hot Decoder Multiple Aggressor Example



In the initial crosstalk analysis iteration, PrimeTime SI considers the combined effect of three aggressor transitions on the victim net. In the subsequent iterations, however, the analysis considers the arrival windows of the aggressor and victim transitions. When analyzing the delay of a falling transition on the victim net, PrimeTime SI finds the arrival windows of rising transitions on the aggressor nets as shown in [Figure 14-11](#). [Table 14-2](#) describes how the aggressors are considered during crosstalk analysis.

Figure 14-11 Multiple Aggressor Arrival Windows at Decoder Outputs**Table 14-2** Crosstalk Analysis With and Without Exclusive Group

	agg1	agg2	agg3
No exclusive group	Quiet	Active	Active
With exclusive group {agg1 agg2 agg3}	Quiet	Active	Quiet

In both cases, agg1 is quiet because it does not overlap with the victim window. When no exclusive group is specified, both agg2 and agg3 are considered to be active because their arrival windows overlap the arrival window of the victim. When the exclusive group is specified, only the aggressor inducing the largest bump, agg2, is considered to be active.

To specify aggressors to be exclusive while switching in the same direction for crosstalk delay and noise analysis, use the `set_si_aggressor_exclusion` command. The `-rise` option specifies that aggressor nets are exclusive in the rise direction; the `-fall` option specifies that nets are exclusive in the fall direction. If you don't specify either of these options, the nets are assumed to be exclusive in both the rise and fall directions. To specify the maximum number of active aggressors, use the `-number_of_active_aggressors` option; if you do not specify this option, by default, only one of the exclusive aggressors is assumed to be active.

In the following example, a group of aggressors is exclusive in the rise direction:

```
pt_shell> set_si_aggressor_exclusion [get_nets {A1 A2 A3 A4}] -rise
```

In the following example, only two of the aggressors in the exclusive group can switch simultaneously:

```
pt_shell> set_si_aggressor_exclusion [get_nets {DECODER*}] \
-number_of_active_aggressors 2
```

The application of commands is order independent. The most restrictive number of active aggressors is chosen. Those aggressors you have already excluded using the `-exclude` option to either the `set_si_delay_analysis` or `set_si_noise_analysis` commands remain in their excluded state. Note that the next trigger of the `update_timing` command, after running these commands, results in a full update.

To remove exclusive groups set in your design, use the `remove_si_aggressor_exclusion` command. This command removes only the exclusive groups set by the `set_si_aggressor_exclusion` command. For example, the following command removes an exclusive group set in the rise direction:

```
pt_shell> remove_si_aggressor_exclusion [get_nets {A1 A2 A3}] -rise
```

To remove all exclusive groups, use the `remove_si_aggressor_exclusion -all` command.

Coupling Separation

The `set_coupling_separation` command creates a separation constraint on nets, like using both the `set_si_delay_analysis` and `set_si_noise_analysis` commands with the `-exclude` option. The following example prevents crosstalk delay and noise analysis between the net CLK1 and all other nets:

```
pt_shell> set_coupling_separation [get_nets CLK1]
```

To ignore the cross-coupling capacitance between two particular nets, use the `-pairwise` option. For example,

```
pt_shell> set_coupling_separation -pairwise \
           [get_nets CLK1] [get_nets NET1]
```

Removing Exclusions

To remove exclusions, use the commands listed in [Table 14-3](#).

Table 14-3 Command for Removing Exclusions

To remove exclusions set by...	Use this command
<code>set_si_delay_analysis</code>	<code>remove_si_delay_analysis</code>
<code>set_si_noise_analysis</code>	<code>remove_si_noise_analysis</code>
<code>set_si_aggressor_exclusion</code>	<code>remove_si_aggressor_exclusion</code>
<code>set_coupling_separation</code>	<code>remove_coupling_separation</code>

These commands can remove only those separations or exclusions that you set directly. These commands cannot remove any coupling separations or exclusions that were implicitly set on the nets due to coupling.

For example, suppose your design has a victim net V, with three aggressor nets A1, A2, and A3. The following command excludes the victim net V from consideration and implicitly exclude the aggressor nets A1, A2, and A3:

```
pt_shell> set_si_delay_analysis -exclude -victims V
```

Although the exclusion on net V implicitly excludes A1 as an aggressor for net V, the exclusion between nets V and A1 cannot be removed by using the `remove_si_delay_analysis -aggressors` command, because no exclusion was directly set on net A1. Therefore, after the following command, the excluded list for net V is still {A1 A2 A3}:

```
pt_shell> remove_si_delay_analysis -aggressors A1
Warning: Cannot remove global separation or exclusion that
was not set on net(s) A1. (XTALK-107)
```

Similarly, although the exclusion on net V implicitly excludes A2 as an aggressor for net V, this exclusion cannot be removed by using the victim-aggressor option because no exclusion was directly set for the victim-aggressor pair V and A2. Therefore, after the following command, the excluded list for net V is still {A1 A2 A3}:

```
pt_shell> remove_si_delay_analysis -victims V -aggressors A2
Warning: Cannot remove global separation or exclusion that
was not set on net(s) V A2. (XTALK-107)
```

To reverse the effect of the `set_si_delay_analysis -exclude -victims` command, use this command:

```
pt_shell> remove_si_delay_analysis -victims V
```

Initial Crosstalk Analysis Run

For the first analysis run with crosstalk analysis, it is a good idea to use the default settings for the crosstalk variables so that you can obtain results quickly. For example,

```
pt_shell> set_app_var si_enable_analysis TRUE
pt_shell> report_timing
```

With the default variable settings, PrimeTime SI performs the crosstalk analysis using two delay calculation iterations. In the first iteration, PrimeTime SI ignores the timing windows to quickly estimate crosstalk delay effects. In the second and final iteration, PrimeTime SI performs a detailed analysis by considering the timing windows or all nets and the transitions (rising or falling) of victim-aggressor nets with overlapping timing windows.

Using the default settings, you can quickly determine the following design and analysis characteristics:

- The effectiveness and accuracy of the current electrical filtering parameter
- The approximate overall signal integrity of the design (by the presence or absence of a large number of constraint violations)
- The approximate runtime behavior for the critical path in the current design
- The detailed timing effects calculated for the critical path

At this point, if no timing violations are reported, it is likely that your design meets the timing specifications. If PrimeTime SI reports violations or small slack values, you need to do a more detailed analysis to find the causes of these conditions. Also, if you are near the end of the design cycle, perform a more detailed analysis to confirm the results of the fast (default) analysis.

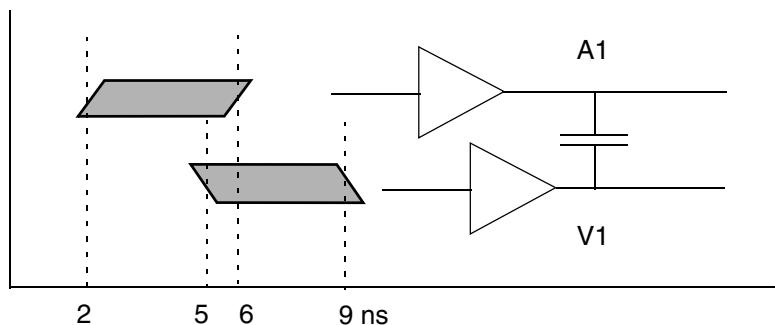
Timing Window Overlap Analysis

Depending on the alignment of the victim and aggressor switching times and the switching directions, the crosstalk effect could cause a victim net to slow down or speed up.

PrimeTime SI calculates the crosstalk effect based on the timing window of the aggressors and victim. This process is referred as timing window overlap analysis or aggressor alignment.

During timing window overlap analysis, PrimeTime SI calculates the crosstalk delta delay per load pin of a net. For this purpose, the timing arrival windows are used by PrimeTime SI, because it encapsulates all the timing paths passing through the net. If the aggressor partially overlaps with the victim's timing window, the partial effect (smaller delta delay) is considered. [Figure 14-12](#) shows how timing windows can overlap.

Figure 14-12 Victim-Aggressor Switching Time Alignment



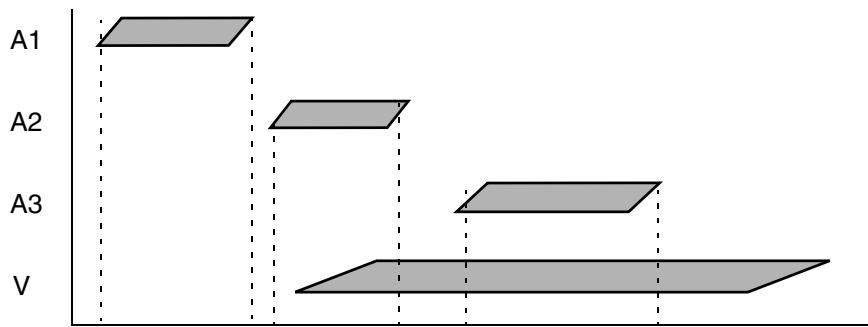
In this example, victim net V1 is coupled to aggressor net A1. The timing arrival windows are 2 ns to 6 ns for V1, and 5 ns to 9 ns for A1. Since the timing window of V1 overlaps with the timing window of A1, PrimeTime SI calculates the crosstalk delta delay of V1 due to A1.

When timing windows from different clocks exist on a victim and aggressor nets, PrimeTime SI considers the different combinations of these clocks with respect to the clock periods.

Multiple Aggressors

When there are multiple aggressors, the signal integrity engine finds the combination of aggressors that could produce the worst crosstalk effect and calculates the crosstalk delta delay for this combination. [Figure 14-13](#) shows a victim net V with three aggressors, A1, A2, and A3.

Figure 14-13 Multiple Aggressor Alignment



In this example, A1 is stronger than A2, and A2 is stronger than A3. Since aggressor A1's window does not overlap with the victim's window, and A2 is stronger than A3, the signal integrity engine calculates the crosstalk delay due to aggressor A2. A1 and A3 are not considered for delta delay calculation.

The `report_delay_calculation -crosstalk` command reports the attributes for aggressors A1 and A3 as follows:

N – aggressor does not overlap for the worst case alignment

Asynchronous Clocks

If the victim timing window clock and the aggressor timing window clocks are asynchronous, they have no fixed timing relationship with each other. The aggressor is treated as infinite window with respect to the victim. The `report_delay_calculation -crosstalk` command reports this as follows:

I – aggressor has Infinite arrival with respect to the victim

For multiple aggressors, if the aggressor clocks are synchronous with each other, but asynchronous with the victim, the timing relationships between the aggressors are respected, but they are still treated as infinite windows with respect to the victim.

Crosstalk Delay Analysis for All Paths

By default, the `si_xtalk_delay_analysis_mode` variable is set to `all_paths` mode. In this mode, PrimeTime SI calculates the maximum possible delta delay (worst crosstalk effect) for the victim and aggressor arrival windows. This ensures that crosstalk delta delay is considered for all paths passing through the victim net, and that the maximum delta delay value is applied on that net. This guarantees that all the paths going through the victim net are conservative.

The disadvantage of the `all_paths` mode is that the largest crosstalk delta delay value is applied to the critical paths, making them pessimistic. When a path is recalculated using path-based analysis, this pessimism is removed. You can also remove this pessimism by using other available crosstalk delay analysis modes.

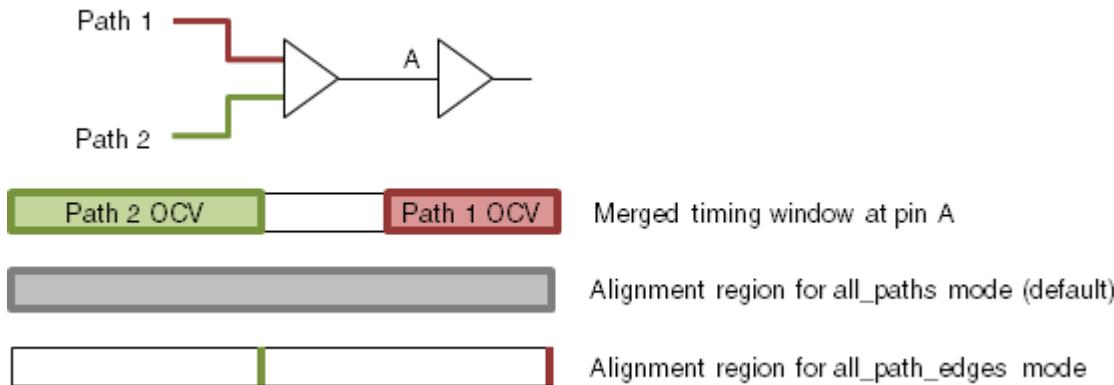
Crosstalk Delay Analysis for All Path Edges

In the `all_paths` alignment mode, two main factors cause pessimism in the crosstalk delay of a stage. First, the switching region of the victim is derived from the early and late timing windows without considering the individual subwindows that constitute it. Therefore, this might include regions where there is no switching on the victim. Second, the entire on-chip variation of the path is considered, creating the effect of multiple paths even when only a single path exists, for example, in a chain of inverters.

The `all_path_edges` mode overcomes the drawbacks of the `all_paths` mode. In this alignment mode, PrimeTime SI considers all paths that pass through a victim net by keeping track of the individual leading edges of those paths. To use this type of analysis, set the `si_xtalk_delay_analysis_mode` variable to the `all_path_edges` mode.

[Figure 14-14](#) shows a comparison of the `all_paths` and `all_path_edges` modes.

Figure 14-14 Comparison of all_paths and all_path_edges Analysis Modes



For example, when computing the maximum crosstalk delay of an inverter chain, PrimeTime SI considers only the late edge of the path and not the early edge. Similarly, when computing the minimum crosstalk delay, PrimeTime SI considers only the early edge of the path and not the late edge.

Note that as in the `all_paths` mode, PrimeTime SI computes the worst-possible crosstalk delay for all paths through the victim net. However, because PrimeTime SI separately considers the effects of early (minimum-delay) and late (maximum-delay) arrivals on delta delay, the analysis of the early paths does not affect that of the late paths, and vice versa. Therefore, the `all_path_edges` mode is less pessimistic than the `all_paths` mode.

Clock Groups

When multiple clocks exist in a design, you can use the `set_clock_groups` command to specify the relationships between the clocks. Doing so allows PrimeTime SI to correctly analyze the crosstalk interactions between the clocks.

The `-group` option specifies the names of the clocks that belong to a group, whereas the `-name` option assigns an arbitrary name to the group. The remaining options specify the relationship between the clocks in the group. The clocks in a group can be defined as logically exclusive, physically exclusive, or asynchronous.

Two clocks defined to be logically exclusive of each other have no logical timing paths between them. PrimeTime does not check the logical timing between the clocks, but PrimeTime SI still checks for possible crosstalk interaction between them.

Two clocks defined to be physically exclusive of each other have no logical timing paths between them, and furthermore, are considered physically isolated from each other. PrimeTime does not check the logical timing between the clocks and PrimeTime SI assumes no possible crosstalk interaction between them.

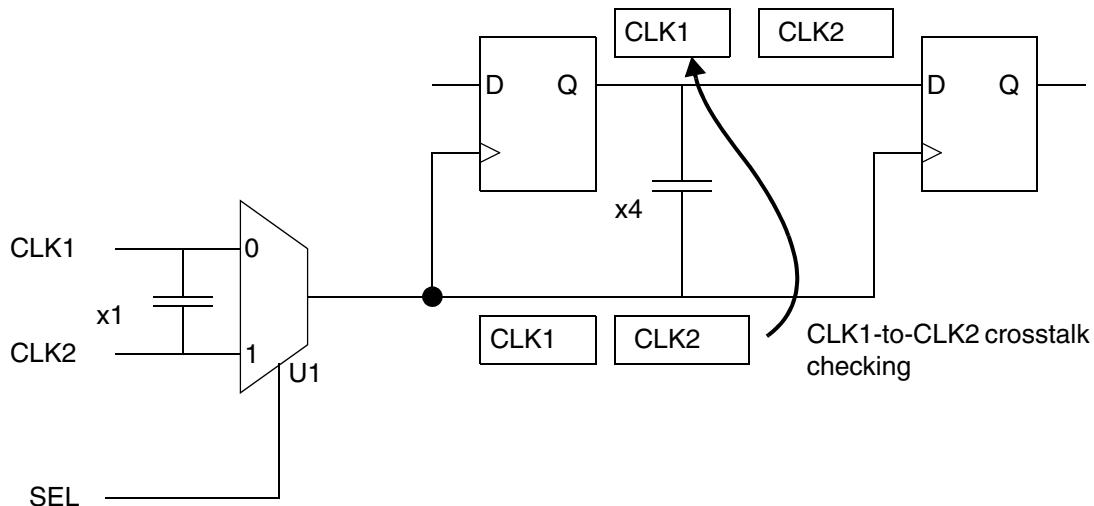
Two clocks defined to be asynchronous with each other have no timing relationship at all. PrimeTime does not check the logical timing between the clocks, but PrimeTime SI still checks for possible crosstalk interaction between them, using infinite arrival windows.

The `-allow_paths` option can be used with asynchronous clocks to restore analysis of the timing paths between clock groups, but still using infinite alignment windows for crosstalk analysis.

Logically and Physically Exclusive Clocks

The most accurate way to analyze multiple clocks is to run a separate analysis for each possible combination of clocks. If there are many such combinations, you can use the distributed multi-scenario analysis (DMSA) feature of PrimeTime to run the analyses and get unified results. However, modern designs often use many clocks, perhaps hundreds or even thousands, to save power. If the number of clocks is very large, it might not be practical to analyze each combination separately. In that case, you can analyze multiple clocks simultaneously in a single run, and use the `set_clock_groups` command to specify the timing relationships between groups of clocks. For example, consider the circuit shown in [Figure 14-15](#). Only one of the two input clocks is enabled at any given time. However, by default, PrimeTime SI considers the crosstalk across capacitor $x4$, between the overlapping arrival windows of CLK1 and CLK2.

Figure 14-15 Circuit with Multiplexed Clocks



The most accurate way to handle this situation is to use case analysis, first setting the MUX control signal to 0 and then to 1. This method ensures that there is no interaction between the clocks and correctly handles all crosstalk situations. However, it requires two analysis runs. For a design with many clocks, it might not be practical to analyze every possible combination of enabled clocks.

To analyze both conditions at the same time, you can define the clocks to be logically exclusive. For example,

```
pt_shell> set_clock_groups -logically_exclusive \
           -group {CLK1} -group {CLK2}
```

The `-logically_exclusive` option causes PrimeTime to suppress any logical (timing path) checking between CLK1 and CLK2, similar to setting a false path constraint between the clocks. However, PrimeTime SI still computes crosstalk delta delays across coupling capacitor x4 between the two clocks, which is pessimistic if the two clocks are not simultaneously present on the nets.

To eliminate this pessimism, you can define the clocks to be physically exclusive. For example,

```
pt_shell> set_clock_groups -physically_exclusive \
           -group {CLK1} -group {CLK2}
```

PrimeTime SI does not compute any delta delays between the clocks defined to be physically exclusive, thereby eliminating the pessimistic analysis of crosstalk between CLK2 and CLK1 across capacitor x4. However, crosstalk across capacitor x1 is also eliminated, which can be optimistic if the MUX is deep inside the chip and x1 is significant.

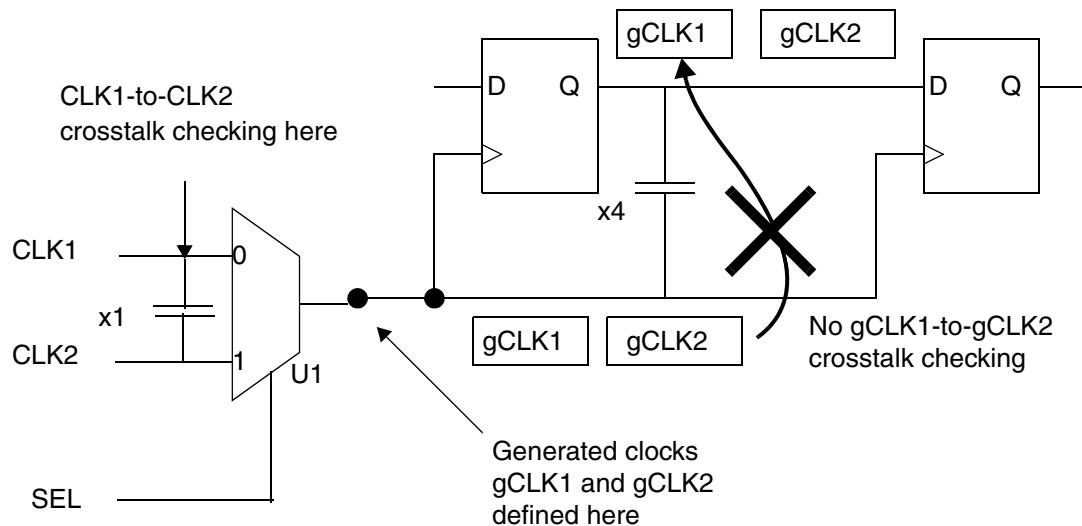
To correctly handle both cross-coupling capacitors, instead of declaring the original clocks to be exclusive, you can define two generated clocks at the output of the MUX and define them to be physically exclusive:

```
pt_shell> create_generated_clock -name gCLK1 \
           -source [get_ports CLK1] -divide_by 1 \
           -add -master_clock [get_clocks CLK1] \
           [get_pins U1/z]

pt_shell> create_generated_clock -name gCLK2 \
           -source [get_ports CLK2] -divide_by 1 \
           -add -master_clock [get_clocks CLK2] \
           [get_pins U1/z]

pt_shell> set_clock_groups -physically_exclusive \
           -group {gCLK1} -group {gCLK2}
```

In that case, PrimeTime SI computes delta delays between CLK1 and CLK2 across x1 before the MUX, but not between gCLK1 and gCLK2 across x4 or elsewhere in the generated clock tree. See [Figure 14-16](#).

Figure 14-16 Circuit with Multiplexed Clocks

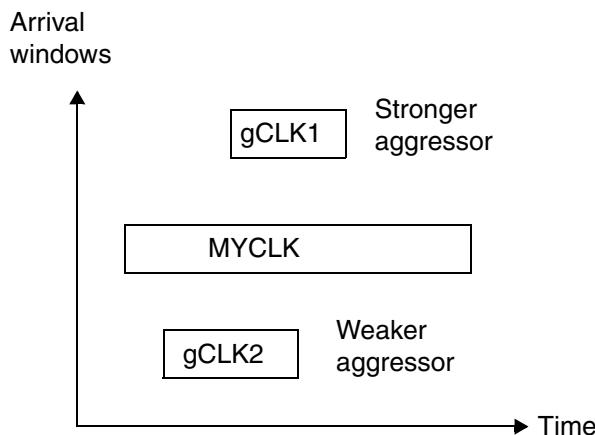
The definition of physically exclusive clock groups removes pessimism by eliminating crosstalk analysis where no crosstalk exists. The pessimism removal applies to both crosstalk timing analysis and crosstalk noise analysis. It is your responsibility to correctly define the clock groups. PrimeTime SI does not verify that a particular setting makes sense for the design.

PrimeTime detects only group-to-group conflicts, not implied conflicts. For example, if you declare $CLK1$ and $CLK2$ to be asynchronous to each other, they are both still synchronous to $CLK3$ by default. This is an implied three-way conflict that PrimeTime does not detect. You are responsible for resolving such conflicts by using the `set_clock_groups` command.

An exclusive or asynchronous path group definition has higher priority than the `set_false_path` command timing exception. Furthermore, the `reset_path` command does not cancel path group relationships set with the `set_clock_groups` command.

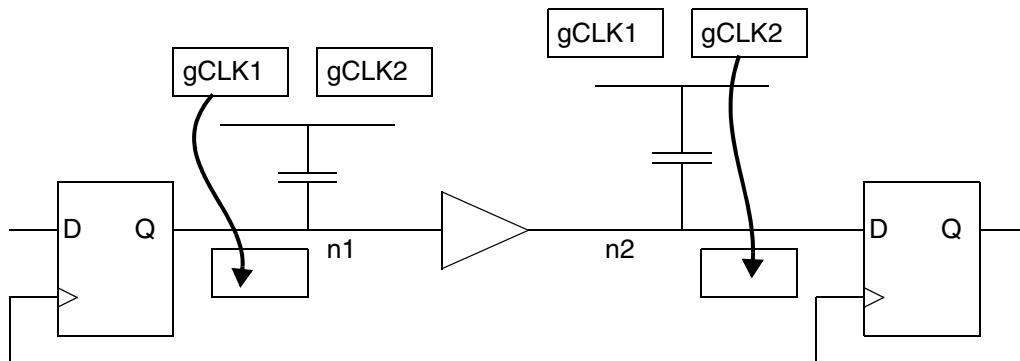
Path-Based Physical Exclusion Analysis

If two or more clocks are defined to be physically exclusive of each other, no more than one of those clocks can operate as an aggressor to a given victim net. For example, consider the crosstalk alignment diagram in [Figure 14-17](#).

Figure 14-17 Crosstalk Alignment Diagram

The physically exclusive clock signals gCLK1 and gCLK2 are aggressors to clock signal MYCLK. Both of the aggressors overlap the arrival window of MYCLK. However, because gCLK1 and gCLK2 are physically exclusive, only one can be an aggressor at that victim net at any given time. PrimeTime SI chooses the stronger aggressor that causes a larger delta delay (in this example, gCLK1), and does not consider the weaker one (gCLK2).

For each net, a different aggressor could be the stronger one. For example, consider the circuit in [Figure 14-18](#). The two clocks gCLK1 and gCLK2 are physically exclusive, and both operate as aggressors to victim nets n1 and n2 in a timing path.

Figure 14-18 Different Exclusive Aggressors on a Path

Because of the different arrival times at the two victim nets, gCLK1 is the aggressor for net n1 and gCLK2 is the aggressor for net n2. This analysis is correct for each individual net, but it is pessimistic for the path as a whole because gCLK1 and gCLK2 are physically exclusive. They cannot both contribute to a decrease in the slack for the path.

You can optionally have PrimeTime consider the worst possible set of allowable (nonexclusive) clocks resulting in the least slack for each path. This whole-path analysis

reduces pessimism, but requires slightly more runtime than considering nets individually. Even with the increased runtime, it is still typically much faster than repeated analysis runs that consider all the clock combinations separately. For the most accurate worst-case, whole-path analysis of physically exclusive clocks, set the `pba_enable_path_based_physical_exclusivity` variable to `true`.

By default, it is set to `false`. It is suggested that you leave this variable set to `false` for most analysis runs, and set it to `true` only for final signoff of the design timing.

Infinite Alignment Windows

The `-allow_paths` option can be used with the `-asynchronous` option to restore logical (timing path) checking between clock groups, while still using infinite alignment windows for crosstalk analysis. This option allows the timing paths between the clock paths to remain in place, but applies infinite windows between the clock groups for conservative crosstalk analysis. For example, the following command defines clocks CLK1 and CLK2 to be asynchronous for purposes of crosstalk analysis (infinite arrival windows), without affecting normal logical timing checks between the two clocks:

```
pt_shell> set_clock_groups -asynchronous -allow_paths \
-group {CLK1} -group {CLK2}
```

You can restrict checking of some or all clock-to-clock paths by using the `set_false_path` command in conjunction with the `set_clock_groups` command.

Crosstalk Analysis with Composite Aggressors

In finer geometries, nets can have a large number of small aggressors. Filtering these small aggressors completely ignores their effects. However, performing detailed analysis on all of them can result in extremely long runtimes. PrimeTime can analyze the effects of many small aggressors in a reasonable amount of runtime with the composite aggressor mode.

The composite aggressor mode is ideal for analyzing designs with the following characteristics:

- A large number of aggressors per victim net
- Little or no filtering of aggressors
- Crosstalk calculations performed in high effort mode

The composite aggressor mode makes an effective tradeoff between runtime and accuracy by evaluating all “small” aggressors, including those that are filtered, in a fast but conservative manner. It also reduce pessimism by utilizing a statistical analysis for the aggressors in the composite aggressor group.

A composite aggressor is a single composite waveform that represents the effects of all the small aggressors, including those that are filtered. This concept is shown in [Figure 14-19](#) and [Figure 14-20](#).

Figure 14-19 Composite Aggressor Mode Disabled

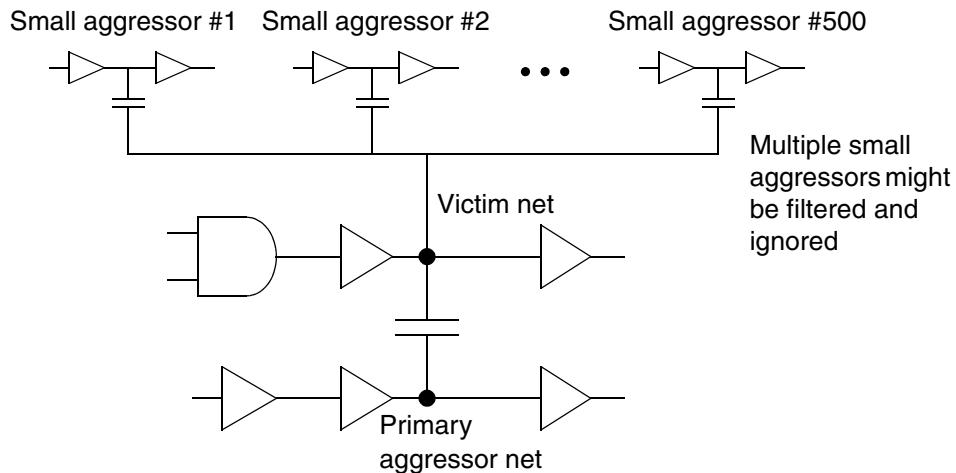
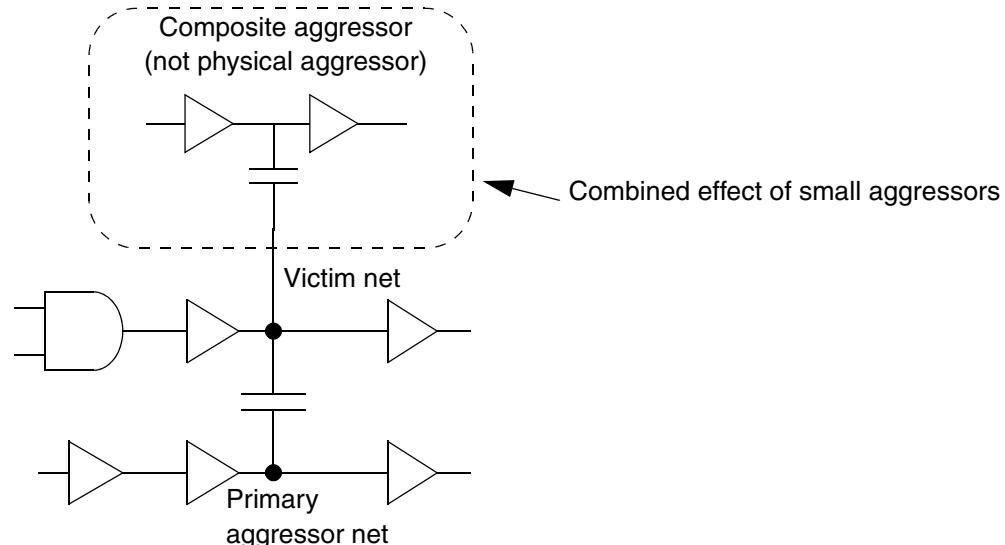


Figure 14-20 Composite Aggressor Mode Enabled



In [Figure 14-19](#), the circuit has 500 small aggressors. Composite aggressor mode replaces these aggressors with a single composite aggressor waveform, as shown in [Figure 14-20](#). Note that the composite aggressor waveform represents the combined effect of the small physical aggressors; the composite aggressor is not made of physical cells.

Enabling Composite Aggressor Mode for Delay Analysis

By default, the composite aggressor mode for crosstalk delay analysis is disabled. To enable it, set the `si_xtalk_composite_aggr_mode` variable to `statistical`.

To specify a bump height threshold below which an aggressor is treated as part of the composite aggressor, set the `si_xtalk_composite_aggr_noise_peak_ratio` variable. Specify the bump height threshold as a fraction of the power supply voltage. The default is 0.01, which specifies that aggressors with a bump height below 1 percent of the power supply voltage become part of the composite aggressor. In addition, any aggressors meeting the normal electrical filtering criteria (both peak and accumulated) also become part of the composite aggressor instead of being discarded.

In the `statistical` mode, all of the small aggressors below the threshold are included as part of the composite aggressor. PrimeTime SI applies statistical analysis to adjust the composite bump height to a lower level. This analysis considers the finite probability that all the small aggressors switch simultaneously to adversely affect the victim.

Operation of the `statistical` mode is based on the assumption that each aggressor contributing to the composite aggressor can switch in the rising or falling direction to either help or hurt the victim, and that the possible range for the composite aggressor bump height is any value from zero and the worst-case value.

Based on these assumptions, PrimeTime SI statistically combines the individual aggressors below the threshold. It determines largest composite aggressor bump height having a probability of occurrence no greater than a certain threshold value. You can set this value with the `si_xtalk_composite_aggr_quantile_high_pct` variable.

By default, the `si_xtalk_composite_aggr_quantile_high_pct` variable is set to 99.73, representing a probability of 99.73 percent that the generated composite bump is at least as large as the actual effect of the small aggressors. This is three standard deviations (3 sigma) from the mean value of a normal distribution. To specify a different probability, set the variable to the desired percentage. For example, choose a smaller value such as 90 to generate a smaller, less conservative composite bump that is closer to the mean value.

Excluding Nets from Statistical Mode

If you want to use `statistical` mode but have certain nets in your design for which you do not want to statistically reduce the aggressor effect, you can disable statistical analysis for those nets by using the `set_si_delay_disable_statistical` and `remove_si_delay_disable_statistical` commands. These commands take a list of nets as an argument. The commands have no effect if a net is not part of the composite aggressor group.

Reporting Composite Aggressors

To report unfiltered aggressors that are part of a composite aggressor, use the `report_delay_calculation -crosstalk` reporting command. Only aggressors that meet the electrical filtering threshold are reported, although all are considered in the composite aggressor. Aggressors that are part of a composite aggressor are labeled with a “C” in the report. A line in the report lists the composite aggressor mode as disabled or statistical.

If you want to see all aggressors, including those that fall below the electrical filtering thresholds, you can use the `get_attribute` command. The four attributes that show a collection in composite aggressor group are:

- `si_xtalk_composite_aggr_min_rise`
- `si_xtalk_composite_aggr_min_fall`
- `si_xtalk_composite_aggr_max_rise`
- `si_xtalk_composite_aggr_max_fall`

You use them for four different analysis types: `min_rise`, `min_fall`, `max_rise`, and `max_fall`. For example, the following command gets aggressor information for `min_rise` analysis:

```
pt_shell> get_attribute -class net victim1 \
    si_xtalk_composite_aggr_min_rise
{ "aggr1", "aggr2", "aggr3" }
```

Path-Based Crosstalk Analysis

In a path-based analysis, PrimeTime SI recalculates the crosstalk effects using new victim arrival times and slews taken from the path collection, but using aggressor arrival windows determined by the previous timing update.

Path-based analysis is useful when there are only a few violations remaining in the analysis, and you want to find out whether these violations are caused by pessimistic analysis of arrival and slew times.

To perform path-based analysis, use the `-pba_mode` option with the `get_timing_paths` or `report_timing` commands.

For more information, see [Path-Based Timing Analysis](#).

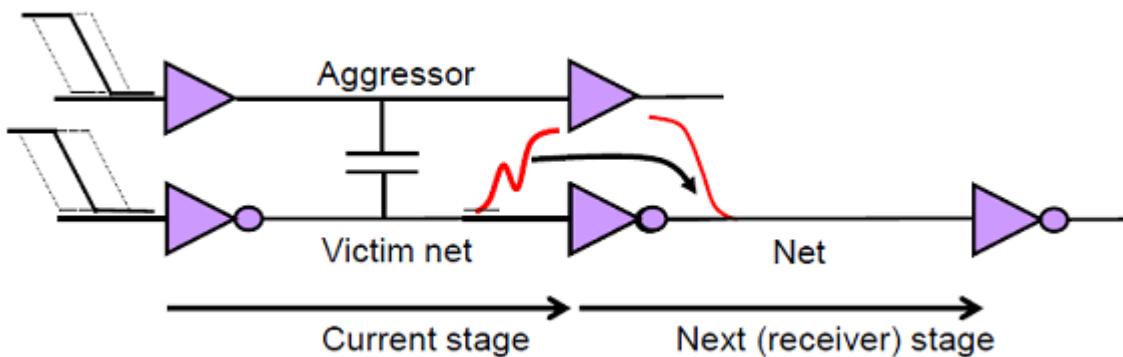
PrimeTime SI Crosstalk Delay Calculation Using CCS Models

For libraries with CCS timing and noise models, PrimeTime SI applies an advanced gate-level simulation method that achieves greater accuracy for crosstalk delay analysis. Gate-level simulation uses CCS timing and noise library data to model the drivers and receivers of the victim net, as well as for those of the aggressor nets. It builds a multi-input,

multi-output, reduced-order model of the coupled interconnects. The tool then performs a time-step based analysis to generate the distorted waveforms due to crosstalk effects at the inputs of the victim receivers.

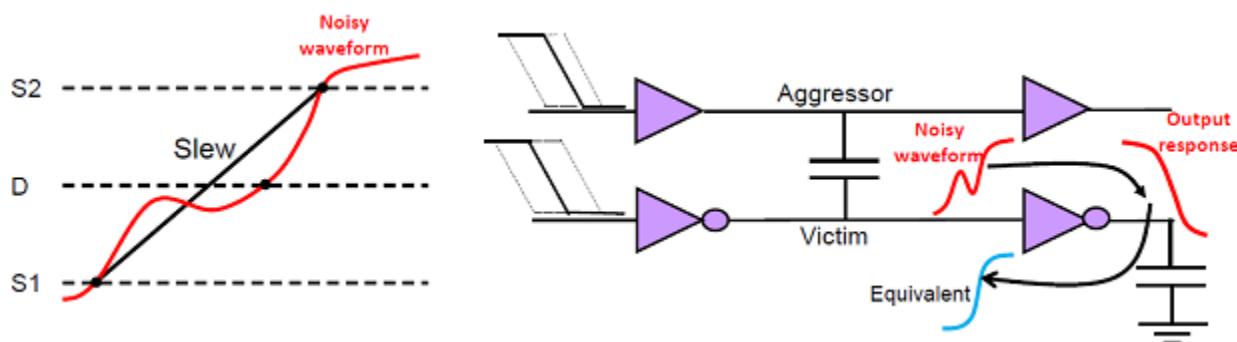
Crosstalk causes distortions in the switching waveforms and affects the delay of the victim stage and its fanouts. The circuit example in [Figure 14-21](#) has a victim net with a single aggressor. Because of cross-coupling between the aggressor net and the victim net, the switching waveform at the input pin of the victim receiver is distorted. This distorted coupled waveform affects the delay of the victim net and the receiver stage. PrimeTime models the effect of the distorted coupled waveform as delta delay at the victim stage.

Figure 14-21 Crosstalk Effect Modeled as Delta Delay for Current Stage and Fanout



PrimeTime SI uses an equivalent waveform approach to model the distorted coupled waveform. PrimeTime delay calculation is stage-based, and the transition time calculated for the current stage is propagated to the next stage for delay calculation. In presence of severely distorted waveform as shown [Figure 14-22](#), using the traditional approach of measuring the slew at slew trip points (S_1 , S_2) would be pessimistic.

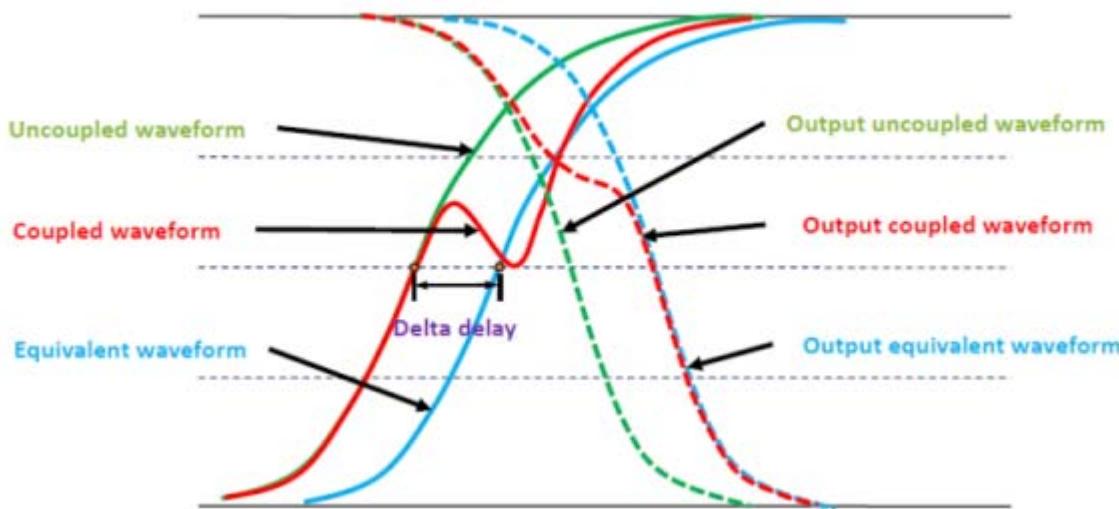
Figure 14-22 Equivalent Waveform Calculation



Using the distorted coupled waveform, PrimeTime SI computes the output response of the receiver (output coupled waveform) and derives an equivalent waveform that bounds the output coupled waveform. The equivalent waveform is the uncoupled waveform shifted in time such that its output (output equivalent waveform) bounds the distorted waveform's

output (output coupled waveform), as shown in [Figure 14-23](#). This ensures a conservative analysis irrespective of the distorted coupled waveform's transition time.

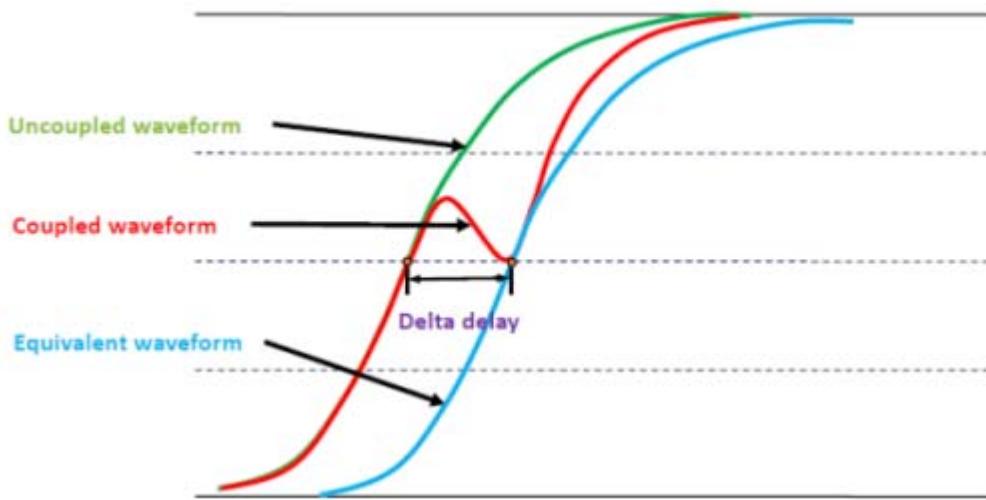
Figure 14-23 Signal Integrity Effect With CCS Noise Library or Forward Timing Arc



The equivalent waveform is computed when the victim receiver cell has CCS noise library data as this is required to calculate the output coupled waveform.

The equivalent waveform is estimated when the victim receiver cell does not have CCS noise library data. This can occur when the victim arc ends at a macro cell or memory block that does not have CCS noise library data. PrimeTime SI cannot forward propagate the distorted waveform to the victim receiver's output. PrimeTime SI estimates an equivalent waveform at the receiver's input pin such that the estimated waveform bounds the distorted waveform at input pin, as shown in [Figure 14-24](#). This ensures a conservative analysis.

Figure 14-24 Signal Integrity Effect Without CCS Noise Library and Forward Timing Arc



The delta delay for the victim stage is measured as the difference between uncoupled waveform and the equivalent waveform. The equivalent waveform is propagated to the next stage for delay calculation.

Gate-level simulation requires that libraries include library characterization waveform information (`normalized_driver_waveform`). PrimeTime SI uses the library characterization waveforms as the input waveforms to the CCS noise driver models. PrimeTime SI also uses the library characterization waveform in the equivalent waveform computation. If the library is missing `normalized_driver_waveform` information, PrimeTime SI assumes that the library was characterized using the Synopsys pre-driver 0.5 waveform. You should check that `normalized_driver_waveform` information is present by using the `check_library` command in the Library Compiler tool.

Waveform Propagation

Waveform effects such as the long tail effect and the receiver Miller effect can be significant in 16-nm and smaller geometries. Ignoring these effects can lead to inaccuracies in delay calculation when design waveforms deviate significantly from the waveforms used during library characterization.

To improve accuracy in the presence of waveform distortions, you can enable waveform propagation, which uses a combination of CCS timing and CCS noise models during delay calculation. Waveform propagation supports both graph- and path-based analyses and requires libraries that contain nonlinear delay models (NLDM), CCS timing, and CCS noise models. The libraries also need the normalized driver waveform.

To enable waveform propagation, set the `delay_calc_waveform_analysis_mode` variable to `full_design`. Waveform propagation improves accuracy for all arcs that have the required library data. If the required data is not available, the tool performs the standard delay calculation.

Annotated Delta Delays

Instead of allowing PrimeTime SI to calculate delta delays resulting from crosstalk, you can set delta delay values explicitly with the `set_annotated_delay -net -delta_only` command. This feature lets you annotate delta delays calculated by an external tool and then perform timing analysis in the presence of those delta delays. For example, to set a delta delay of 0.12 time units on the net arc from output pin U1/Z to input pin U2/A for maximum-delay analysis, use this command:

```
pt_shell> set_annotated_delay -net -max -delta_only \
           -from U1/Z -to U2/A 0.12
```

If you run PrimeTime with crosstalk analysis disabled (`si_enable_analysis` set to `false`), PrimeTime applies the annotated delta delay to the path. However, if you run a crosstalk analysis, a delta delay value calculated by PrimeTime SI overrides any delta delay value set manually with the `set_annotated_delay -delta_only` command.

If you annotate both a delta delay and an overall delay for the same timing arc by using the `set_annotated_delay` command, both with and without the `-delta_only` option, the annotated overall delay is assumed to include any delta delay and the `-delta_only` value is not used.

You can similarly set delta transition times on specified ports or pins with the `set_annotated_transition -delta_only` command.

Iteration Count and Exit

PrimeTime SI calculates crosstalk effects using an iterative loop. The first iteration uses infinite timing windows between aggressor and victim nets. In successive iterations, PrimeTime SI continues to analyze all nets.

To specify the maximum number of iterations, irrespective of the analysis results, use the `si_xtalk_exit_on_max_iteration_count` variable. The default is 2, which is also the maximum allowed value.

Depending on your design, you might need to perform only one iteration. To do this, specify,

```
pt_shell> set_app_var si_xtalk_exit_on_max_iteration_count 1
```

You can terminate an analysis in progress by pressing Ctrl+C one time. PrimeTime SI completes the current analysis iteration before exiting from the loop. Note that pressing Ctrl+C multiple times causes an exit from PrimeTime SI upon completion of the loop.

Timing Reports

There are several different commands for generating reports on crosstalk effects:

- The `report_timing` command generates a slack timing report that includes crosstalk delay effects.
- The `report_si_bottleneck` command helps determine the major victim nets or aggressor nets that are causing multiple violations.
- The `report_delay_calculation -crosstalk` command provides detailed information about crosstalk calculations for a particular victim net.
- The `report_si_double_switching` command helps determine those victim nets with double-switch violations as described in [Fixing Double-Switching Violations](#).
- The `report_noise` command reports static noise effects (noise bumps on quiet victim nets as described in [Static Noise Analysis](#)).

Viewing the Crosstalk Analysis Report

When crosstalk analysis is enabled, the report generated by the `report_timing` command shows the path delays, including crosstalk effects. To see detailed crosstalk information in the report, use the `-crosstalk_delta` option with the `report_timing` command. For example,

```
pt_shell> report_timing -transition_time -crosstalk_delta \
           -input_pins -significant_digits 4
```

Using the `-crosstalk_delta` option causes input pins to be displayed in the report, even if you do not use the `-input_pins` option.

[Example 14-1](#) shows a timing report with crosstalk effects.

Example 14-1 Timing Report With Crosstalk Effects

```
*****
Report : timing
         -path full
         -delay_type max
         -input_pins
         -max_paths 1
         -transition_time
         -crosstalk_delta
...
*****
Startpoint: reset (input port)
Endpoint: hostif_0/host_data_rx28x
          (recovery check against rising-edge clock clock)
Path Group: **async_default**
Path Type: max
```

Point	DTrans	Trans	Delta	Incr	Path
<hr/>					
clock (input port clock) (rise edge)			0.0000	0.0000	
input external delay			0.0000	0.0000	f
reset (in)	0.0000	0.0066	0.0000	0.0028 &	0.0028 f
U26/A (BF1T2)	0.0000	1.0368		0.7040 &	0.7068 f
U26/Z (BF1T2)		0.0000		0.0000	0.7068 f
hostif_0/reset (hostif_mstest_1)	0.0023	1.0414	0.0087	0.0388 &	0.7456 f
hostif_0/U1836/A (IV)	0.0000	0.7593		0.5008 &	1.2463 r
hostif_0/U1836/Z (IV)		0.7593		0.0002 &	1.2466 r
hostif_0/U1835/A (BF1T4)	0.0000	1.1855		0.7458 &	1.9924 r
hostif_0/host_data_regex28x/CD (FD2S)	0.0000	1.1891	0.0597	0.1036 &	2.0960 r
data arrival time					2.0960
clock clock (rise edge)	0.0000		5.0000	5.0000	
clock network delay (ideal)			0.0000	5.0000	
hostif_0/host_data_regex28x/CP (FD2S)				5.0000	r
library recovery time			-0.2470	4.7530	
data required time				4.7530	

data required time				4.7530	
data arrival time				-2.0960	

slack (MET)				2.6571	
<hr/>					
Startpoint: hostif_0/access_state_regex0x (rising edge-triggered flip-flop clocked by clock)					
Endpoint: hostif_0/host_address_regex21x (rising edge-triggered flip-flop clocked by clock)					
Path Group: clock					
Path Type: max					

Point	DTrans	Trans	Delta	Incr	Path
<hr/>					
clock clock (rise edge)	0.0000		0.0000	0.0000	
clock network delay (ideal)			0.0000	0.0000	
hostif_0/access_state_regex0x/CP (FD2SP)	0.0000		0.0000	0.0000	r
hostif_0/access_state_regex0x/Q (FD2SP)		0.7233		0.6764 &	0.6764 r
hostif_0/U1752/A (ND2)	0.0000	0.7233	0.0115	0.0164 &	0.6928 r
hostif_0/U1752/Z (ND2)		0.4238		0.3678 &	1.0606 f
hostif_0/U1751/A (IV)	0.0000	0.4238	0.0114	0.0117 &	1.0722 f
hostif_0/U1751/Z (IV)		0.2866		0.1890 &	1.2612 r
hostif_0/U2275/C (ND3)	0.0000	0.2866	0.0066	0.0067 &	1.2679 r
hostif_0/U2275/Z (ND3)		0.4255		0.2220 &	1.4899 f
hostif_0/U2276/A (IV)	0.0000	0.4255	0.0189	0.0191 &	1.5089 f
hostif_0/U2276/Z (IV)		0.5016		0.2960 &	1.8050 r
hostif_0/U1736/A (ND2)	0.0000	0.5016	0.0389	0.0389 &	1.8439 r
hostif_0/U1736/Z (ND2)		0.4531		0.3601 &	2.2040 f
hostif_0/U2266/B (NR4X05)	0.0000	0.4531	0.0219	0.0228 &	2.2268 f
hostif_0/U2266/Z (NR4X05)		0.6603		0.3408 &	2.5676 r
hostif_0/U2264/C (AO7CNP)	0.0000	0.6603	0.0200	0.0200 &	2.5875 r
hostif_0/U2264/Z (AO7CNP)		0.2220		0.6179 &	3.2054 f
hostif_0/U2271/C (AO7X05)	0.0000	0.2221	0.0077	0.0085 &	3.2139 f

hostif_0/U2271/Z (AO7X05)		1.7745	0.6765 &	3.8904 r
hostif_0/U2272/A (IVP)	0.0000	1.7745 0.1337	0.1347 &	4.0250 r
hostif_0/U2272/Z (IVP)		1.0285	0.9720 &	4.9970 f
hostif_0/U1718/B (ND2)	0.0000	1.0286 0.0246	0.0314 &	5.0285 f
hostif_0/U1718/Z (ND2)		0.9974	0.6208 &	5.6492 r
hostif_0/U1894/A (IV4)	0.0000	0.9974 0.0561	0.0581 &	5.7073 r
hostif_0/U1894/Z (IV4)		0.4672	0.4603 &	6.1676 f
hostif_0/U1895/A (BF1T4)	0.0000	0.4673 0.0116	0.0155 &	6.1831 f
hostif_0/U1895/Z (BF1T4)		0.4958	0.5413 &	6.7244 f
hostif_0/U1589/B (ND2)	0.0000	0.4968 0.0298	0.0443 &	6.7688 f
hostif_0/U1589/Z (ND2)		0.2541	0.2019 &	6.9706 r
hostif_0/U1781/A (ND4X05)	0.0000	0.2541 0.0413	0.0414 &	7.0120 r
hostif_0/U1781/Z (ND4X05)		0.8272	0.4066 &	7.4186 f
hostif_0/U2044/A (MUX21)	0.0000	0.8272 0.6563	0.6568 &	8.0754 f
hostif_0/U2044/Z (MUX21)		0.1661	0.4333 &	8.5087 f
hostif_0/host_address_regex21x/D (FD2S)		0.0000 0.1661	0.0120 0.0121 &	8.5208 f
data arrival time				8.5208
clock clock (rise edge)	0.0000		5.0000	5.0000
clock network delay (ideal)			0.0000	5.0000
hostif_0/host_address_regex21x/CP (FD2S)				5.0000 r
library setup time			-0.3633	4.6367
data required time				4.6367

data required time				4.6367
data arrival time				-8.5208

slack (VIOLATED)				-3.8841 1

PrimeTime SI calculates and reports crosstalk effects on a per-stage basis. A stage consists of one cell together with its fanout net. PrimeTime SI stores all the delta delay and delta slew per-stage values on the path's input pin.

The report contains columns labeled Dtrans (delta transition) and Delta (delta delay) to show the contribution of crosstalk to the delay per stage in the path. The column labeled Incr (increment) already includes the calculated delta delay. An ampersand character (&) in the Incr column indicates the presence of parasitic data.

You can customize your reports by using a Tcl script that comes with PrimeTime SI. Use the `install_path/auxx/pt/examples/tcl/custom_timing_1.tcl` Tcl script for customizing reports.

Bottleneck Reports

If the `report_timing` command reports a large number of crosstalk violations, the `report_si_bottleneck` command can help determine the major victim nets or aggressor nets that are causing multiple violations, in the same way that the `report_bottleneck` command determines the causes of multiple minimum and maximum delay violations.

The `report_si_bottleneck` command reports the nets having the highest “cost function,” or highest contribution to undesirable crosstalk effects that cause timing violations. You can choose any one of four different cost functions:

- `delta_delay` – Lists the victim nets having the largest absolute delta delay, among all victim nets with less than a specified slack.
- `delta_delay_ratio` – Lists the victim nets having the largest delta delay relative to stage delay, among all victim nets with less than a specified slack.
- `total_victim_delay_bump` – Lists the victim nets having the largest sum of all unfiltered bump heights (as determined by the net attribute `si_xtalk_bumps`), irrespective of delta delay, among all victim nets with less than a specified slack.
- `delay_bump_per_aggressor` – Lists the aggressor nets that cause crosstalk delay bumps on victim nets, listed in order according to the sum of all crosstalk delay bumps induced on affected victim nets, counting only those victim nets having less than a specified slack.

By default, the specified slack level is zero, which means that costs are associated with timing violations only. If there are no violations, there are no costs and the command does not return any nets. To get a more extensive report, you can set the slack threshold to a larger value. For example, to get a list of all the victim nets with a delay violation or within 2.0 time units of a violation, listed in order of delta delay:

```
pt_shell> report_si_bottleneck -cost_type delta_delay \
           -slack_lesser_than 2.0
```

Unless you specify either the `-max` or `-min` option, the command considers both maximum-delay (setup) and maximum-delay (hold) constraints.

Nets reported by the bottleneck command can be targeted for further investigation with the `report_delay_calculation -crosstalk`. If you find that the reported violations are valid, you can use command and “what-if” repair commands such as `size_cell` and `set_coupling_separation` to see the effects of different repair strategies.

By default, clock nets are not included in the bottleneck analysis because there are no slack values associated with those nets. However, you can include clock nets in the analysis by using the `-include_clock_nets` option.

You might want to investigate situations where many aggressors affect a single net. To get a bottleneck report that only includes these situations, use the `minimum_active_aggressors` option. In the following example, the bottleneck command only reports nets where three or more active aggressors are affecting the net:

```
pt_shell> report_si_bottleneck -cost_type delta_delay \
           -minimum_active_aggressors 3
```

Crosstalk Net Delay Calculation

You can use the `report_delay_calculation` command with the `-crosstalk` option to get detailed information about crosstalk calculations done by PrimeTime SI for a particular victim net. The command lists the aggressor nets and describes how they affect the victim net.

The `-crosstalk` option can be used only with a net timing arc, not a cell timing arc. To get information about a victim net, specify the net driver pin and load pin as in this example:

```
pt_shell> report_delay_calculation -crosstalk \
           -from [get_pins g1/Z] -to [get_pins g2/A]
```

The `report_delay_calculation` command reports the following information:

- The number of cross-coupled aggressor and active aggressors remaining after filtering
- Victim analysis information such as active or inactive status and reselection status
- Detailed information about each active aggressor such as bump height and window alignment status
- Reasons for inactive aggressor status such as filtering, case analysis, or bump height too small
- Delta delay and delta slew (reported with or without the `-crosstalk` option)

Note:

In PrimeTime SI, the delta slew for setup analysis is always positive or zero, and the delta slew for hold analysis is always negative or zero.

Reporting Crosstalk Settings

To check your crosstalk settings, use these commands:

```
report_si_delay_analysis
report_si_noise_analysis
report_si_aggressor_exclusion
```

The `report_si_delay_analysis` command reports the crosstalk settings made with the following delay analysis commands:

```
set_si_delay_analysis
remove_si_delay_analysis
set_si_delay_disable_statistical
remove_si_delay_disable_statistical
set_coupling_separation
remove_coupling_separation
```

Similarly, the `report_si_noise_analysis` command reports the crosstalk settings made with the following noise analysis commands:

```
set_si_noise_analysis
remove_si_noise_analysis
set_si_noise_disable_statistical
remove_si_noise_disable_statistical
set_coupling_separation
remove_coupling_separation
```

By default, crosstalk settings for all nets are reported. If you provide a list of one or more nets, the commands report the crosstalk settings applied directly to those nets.

The `report_si_delay_analysis` and `report_si_noise_analysis` commands do not trigger a timing or noise update because they only report (not change) existing attributes.

For net-specific reporting, only crosstalk settings directly applied to the specified net are reported. This does not include global coupling separations or exclusions on other nets which implicitly affect the specified net. For example, consider a design where net n1 couples to nets n2 and n3. If you apply the following global coupling separation to net n1, the bump from n1 is disabled in the delay calculation reports for nets n2 and n3:

```
pt_shell> set_coupling_separation n1
```

However, if you request a report for net n2, no crosstalk settings are reported because the global coupling separation was applied to n1, and affects n2 implicitly:

```
pt_shell> report_si_delay_analysis n2
```

If desired, the crosstalk settings can be reported for all nets coupled to n2:

```
pt_shell> report_si_delay_analysis [get_attribute \
[get_nets n2] effective_aggressors]
```

If the coupling separation was explicitly applied between n1 and n2 using the `-pairwise` option, then the crosstalk constraint is shown in the reports for n2 and n3:

```
pt_shell> set_coupling_separation n1 -pairwise {n2 n3}
```

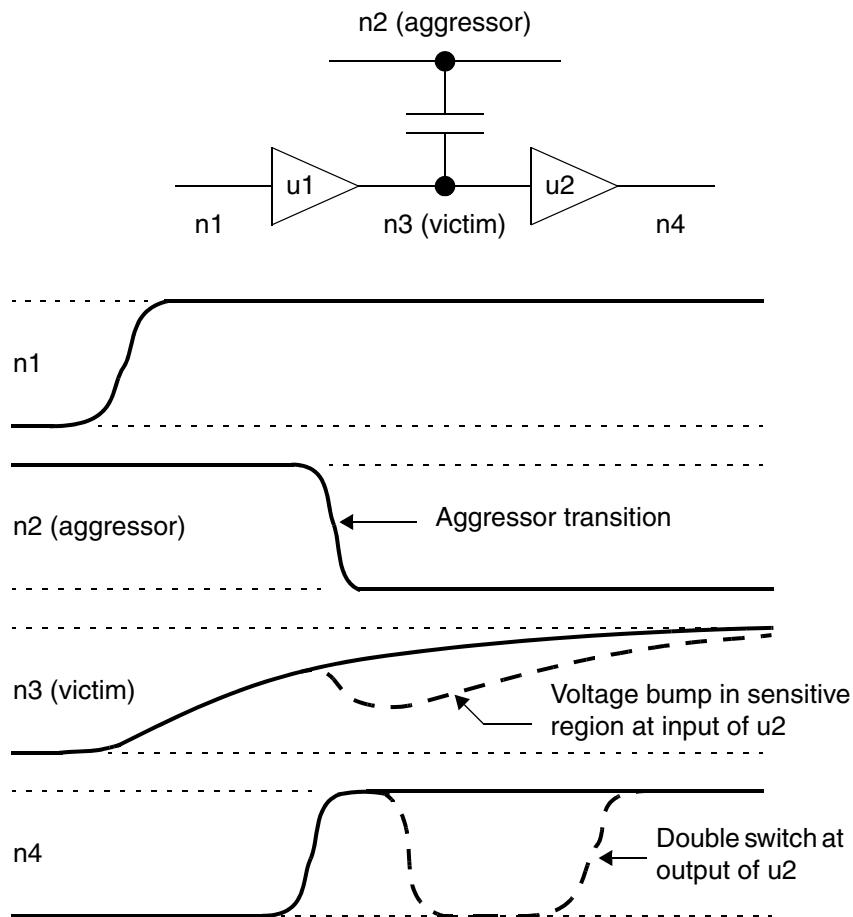
The `report_si_aggressor_exclusion` command reports the crosstalk settings made with the `set_si_aggressor_exclusion` command. Net-specific reporting for the `report_si_aggressor_exclusion` command reports all exclusive groups to which the specific net belongs. To find out which of the aggressors were considered active and which of them were screened as quiet, you can use the `report_delay_calculation` and `report_noise_calculation` commands, respectively, for crosstalk delay and noise analysis. Note that the different aggressor nets from the same exclusive group can be active during different types of analysis based on their coupling information and worst-case alignment.

Double-Switching Detection

When you use the `update_timing` command, PrimeTime SI detects timing violations resulting from the effects of crosstalk on transitions occurring on victim nets. The slowdown or speedup of a transition can trigger a setup or hold timing violation. When you use the `update_noise` command, PrimeTime SI detects functional errors resulting from the effects of crosstalk on steady-state nets. A large noise bump on a steady-state net can cause an incorrect logic value to be propagated.

In addition to crosstalk timing errors and steady-state functional errors, PrimeTime SI can also detect functional errors resulting from crosstalk effects on switching victim net. These types of errors are called double-switching errors. An example is shown in [Figure 14-25](#).

Figure 14-25 Double-Switching Error Example

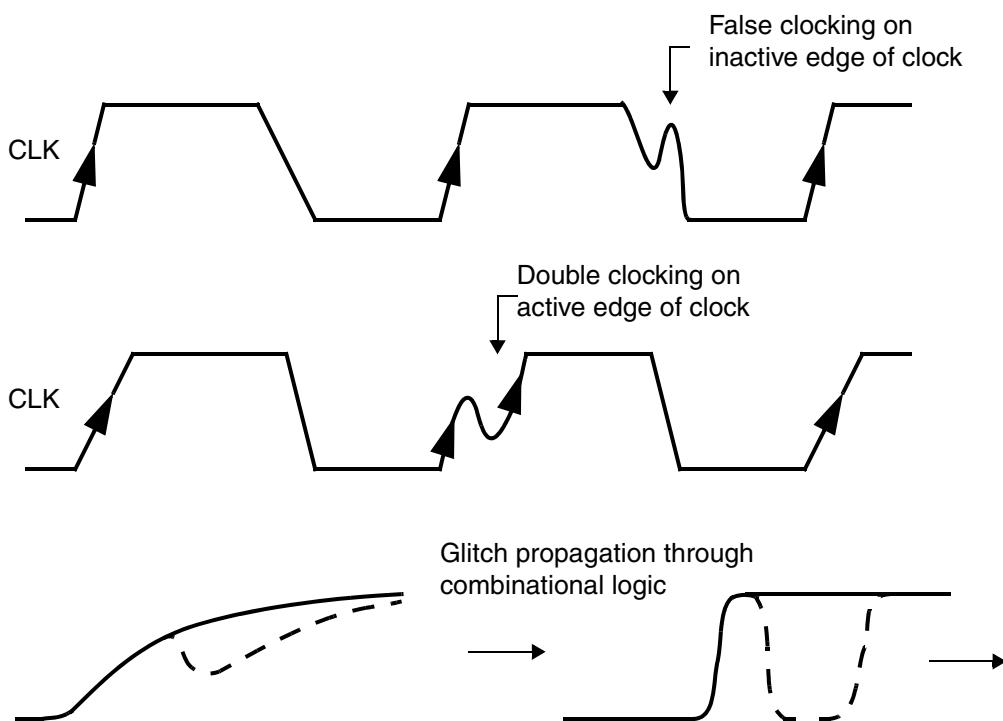


In this example, a rising transition on net n1 is propagated through buffers u1 and u2 to nets n3 and n4. Because of the low drive of buffer u1 and the capacitive load of net n3, the transition on n3 is relatively slow. In the presence of crosstalk, which is indicated by the

dashed lines in [Figure 14-25](#), an aggressor transition causes a voltage bump in the sensitive voltage area at the input of buffer u2. This causes the output of the buffer to switch twice.

Double-switching errors such as this can cause incorrect circuit operation by *false clocking* on the inactive edge of a clock signal, by *double clocking* on the active edge of a clock signal, or glitch propagation through combinational logic. These effects are shown in [Figure 14-26](#). The false clocking and double clocking examples cause undesired clocking of rising-edge-sensitive registers.

Figure 14-26 Undesirable Effects of Double-Switching Errors



Double-switching errors are caused by crosstalk coupling capacitance between a strong aggressor transition acting on a sensitive victim. These are the same conditions that cause static noise errors on steady-state nets, so most double-switching cases are detected by static noise analysis with the `update_noise` command. However, there might be cases where crosstalk effects are large enough to cause double-switching errors, but not large enough to cause steady-state functional failures. These double-switching cases are not detected by the `update_noise` command.

Invoking Double-Switching Error Detection

To enable double-switching error detection, set the `si_xtalk_double_switching_mode` variable to either `clock_network` or `full_design`. Setting the variable to `clock_network` checks for false clocking, double clocking, and double switching in the clock network only. Setting the value to `full_design` checks these same conditions in the data paths as well as in the clock network.

This is the recommended procedure for double-switching error detection:

1. Perform static timing analysis without crosstalk. Ensure that there are no timing violations and maximum transition violations.
2. Perform static timing analysis and static noise analysis with crosstalk. Fix any reported crosstalk timing and noise violations.
3. Enable double-switching crosstalk analysis and perform static timing analysis. Fix any reported double-switching errors.

Remember that most double-switching error conditions are detected by ordinary static noise analysis.

PrimeTime SI detects double-switching during the `update_timing` command and marks the net attributes of the nets that have double-switching. To get the attribute information you can use:

```
pt_shell> get_attribute [get_nets clk_net] \
    si_has_double_switching
```

You can use this attribute in a script to find and fix the double-switching conditions detected.

How Double-Switching Is Detected

There are several different conditions that affect the determination of double-switching, including the cross-capacitance value, the aggressor and victim drive characteristics, the aggressor arrival time and direction (rise or fall) with respect to the victim transition, the load coupling capacitance of the victim net, and the input sensitivity of the cells driven by the victim net.

PrimeTime SI uses the CCS noise model to propagate the coupled waveform for each stage and to check for potential double-switching. Therefore, to perform double-switching error detection, the design must use a cell library with CCS noise models. Double-switching detection is done for first and subsequent iterations of crosstalk analysis.

For each detected double-switching functional failure, PrimeTime SI sets the `si_has_double_switching` net attribute to `true`. It also measures the severity of the double-switching error and reports it as double-switching slack. A victim with a higher risk of double-switching is reported to have a more negative slack. In an engineering change order (ECO), first fix the cases with the worst double-switching slack.

The slack value is stored in the `si_double_switching_slack` net attribute. For nets without any double-switching error, this attribute is set to `POSITIVE`. If there is no CCS noise model available in the library, the switching bump is unconstrained and the slack attribute is `INFINITY`.

Reporting Double-Switching Violations

After a timing update with crosstalk analysis and double-switching error detection enabled, you can report the presence of double-switching errors detected. Use the `report_si_double_switching` command to report all the victim nets that have double-switching. Note that many of these victims could also cause noise violations.

The `-clock_network` option of the `report_si_double_switching` command reports the double-switching violations for the clock network only. This option is independent of the `si_xtalk_double_switching_mode` variable setting being either `clock_network` or `full_design`.

Use the `-rise` option of the `report_si_double_switching` command to report double-switching violations for rising victims only, or the `-fall` option to report falling victims only. Specify a list of nets to check only those nets. Otherwise, the whole clock network or the full design is checked, depending on the `si_xtalk_double_switching_mode` variable.

Here is an example of a double-switching report:

```
pt_shell> report_si_double_switching -nosplit
...
Victim Switching Actual Required Double Switching
Net    Direction Bump Height Bump Height Slack
~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~
I2      max_rise  0.69     0.42      -0.26 (Violating)
I1      max_fall  0.50     0.30      -0.20 (Violating)
```

This design has two potential double-switching errors. Net I2 has higher chance of having a double-switching error and should have a higher priority for fixing.

The `report_delay_calculation -crosstalk` command shows whether there are double-switching violations on a victim net, if the `si_xtalk_double_switching_mode` variable is set to either the `clock_network` or `full_design` value.

Fixing Double-Switching Violations

There are several ways to fix double-switching violations. For example, you can decrease the cross-capacitance by spacing or shielding the adjacent wires, or enlarge the victim net driver to reduce the transition time.

The `fix_eco_timing` command can be used to fix timing violations. For more information, see [Fixing Setup and Hold Timing Violations](#).

Static Noise Analysis

The PrimeTime SI tool can analyze designs for logic failure due to crosstalk noise on steady-state nets. To learn how to perform static noise analysis, see

- [Static Noise Analysis Overview](#)
 - [PrimeTime SI Noise Analysis Flow](#)
 - [Noise Analysis Commands](#)
 - [Noise Modeling With CCS Noise Data](#)
 - [Noise Modeling With Nonlinear Delay Models](#)
-

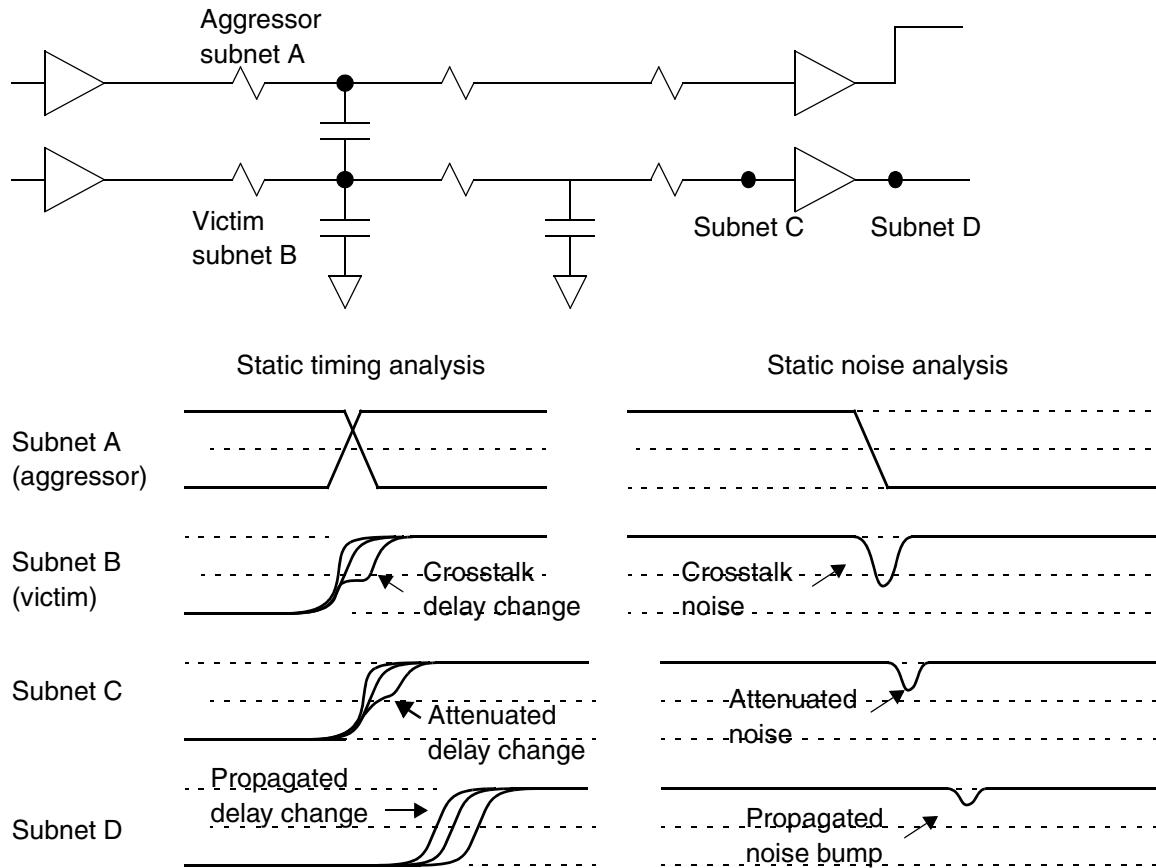
Static Noise Analysis Overview

Static noise analysis is a technique for finding the worst noise effects in a design so that they can be reduced or eliminated, thereby maximizing the reliability of the finished product.

Static noise analysis, like static timing analysis, does not rely on test vectors or circuit simulation to determine circuit behavior. Instead, it considers the cross-coupling capacitance between aggressor nets and the victim net, the arrival windows of aggressor transitions, the drive characteristics of the aggressor nets, the steady-state load characteristics of the victim net driver, and the propagation of noise through cells. Using this information, PrimeTime SI determines the worst-case noise effects and reports conditions that could lead to logic failure.

[Figure 14-27](#) compares static timing analysis and static noise analysis done by PrimeTime SI. For either type of analysis, the tool considers the cross-coupling between aggressor nets and victim nets. For static timing analysis, the tool determines the worst-case change in delay on the victim net caused by crosstalk. For static noise analysis, it determines the worst-case noise bump or glitch on a steady-state victim net. (Steady-state means that the net is constant at logic 1 or logic 0.) A noise bump on a steady-state net can be propagated down the timing path and could be captured as incorrect data at the path endpoint.

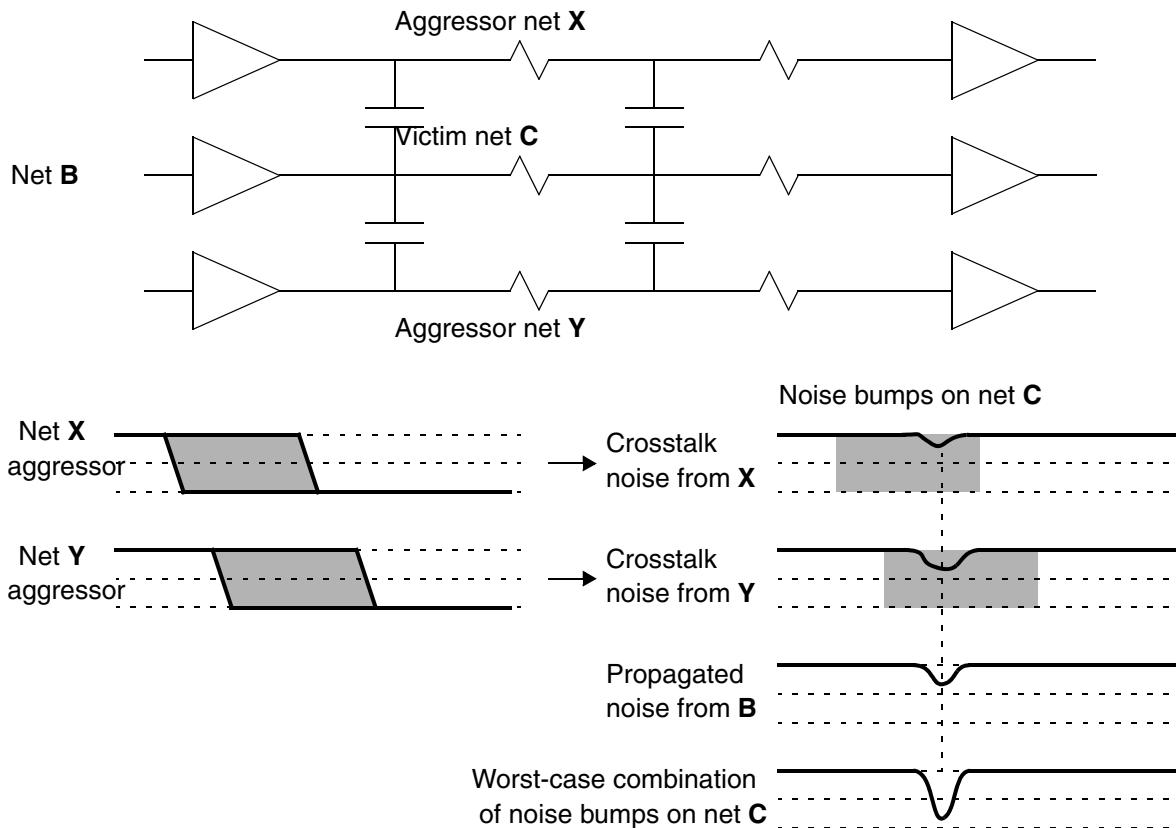
Figure 14-27 Crosstalk Effects on Timing and Steady-State Voltage



The main commands for noise analysis are the `check_noise`, `update_noise`, and `report_noise` commands, which operate in a manner similar to the `check_timing`, `update_timing`, and `report_timing` commands for static timing analysis. The `check_noise` command checks the design for the presence and validity of noise models at driver and load pins. The `update_noise` command performs a noise analysis and updates the design with noise bump information. The `report_noise` command reports worst-case noise effects, including width, height, and noise slack.

Figure 14-28 shows how the tool combines the effects from multiple aggressors and propagated noise, taking the aggressor timing windows into account, to determine the worst-case noise bump on the victim net. It propagates the worst-case noise bump through the subnets of the victim net, resulting in a composite noise bump on each load pin of the victim net.

Figure 14-28 Combined Effects of Crosstalk and Propagated Noise

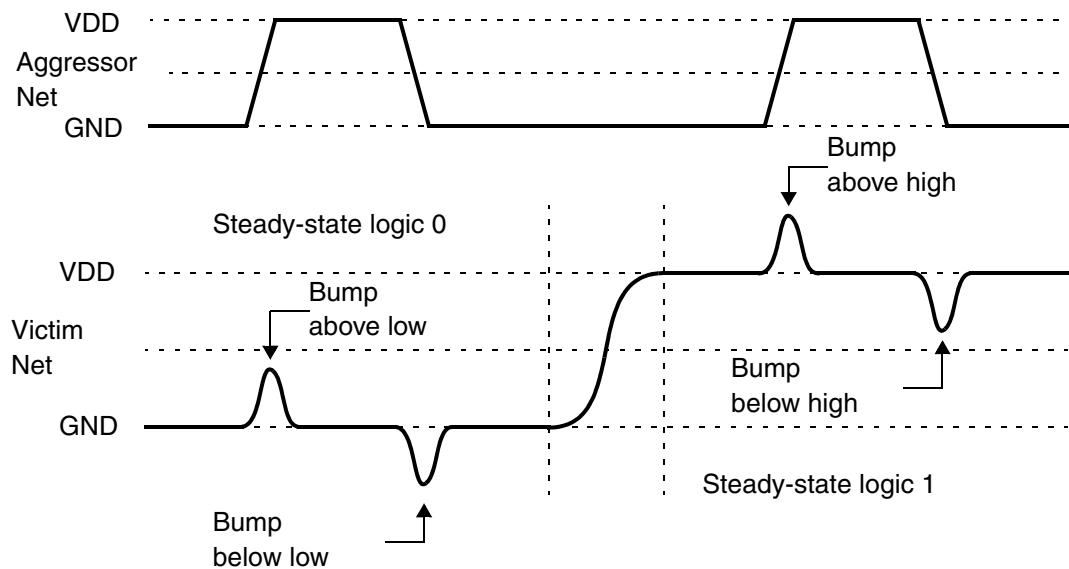


Noise Bump Characteristics

The term “noise” in electronic design generally means any undesirable deviation in voltage of a net that ought to have a constant voltage, such as a power supply or ground line. In CMOS circuits, this includes data signals being held constant at logic 1 or logic 0.

There are many different causes of noise such as charge storage effects at p-n junctions, power supply noise, and substrate noise. However, the dominant noise effect in deep-submicron CMOS circuits is crosstalk noise between physically adjacent logic nets. For this reason, the tool concentrates on the analysis of crosstalk noise between these nets and the propagation of the resulting crosstalk bumps from cell input to cell output.

Noise effects, when plotted as voltage versus time, can have many different forms: bumps, ripples, random noise, and so on. PrimeTime SI concentrates on the four types of noise bumps typically caused by aggressor net transitions: above low, below low, above high, and below high, as shown in [Figure 14-29](#).

Figure 14-29 Noise Bump Types

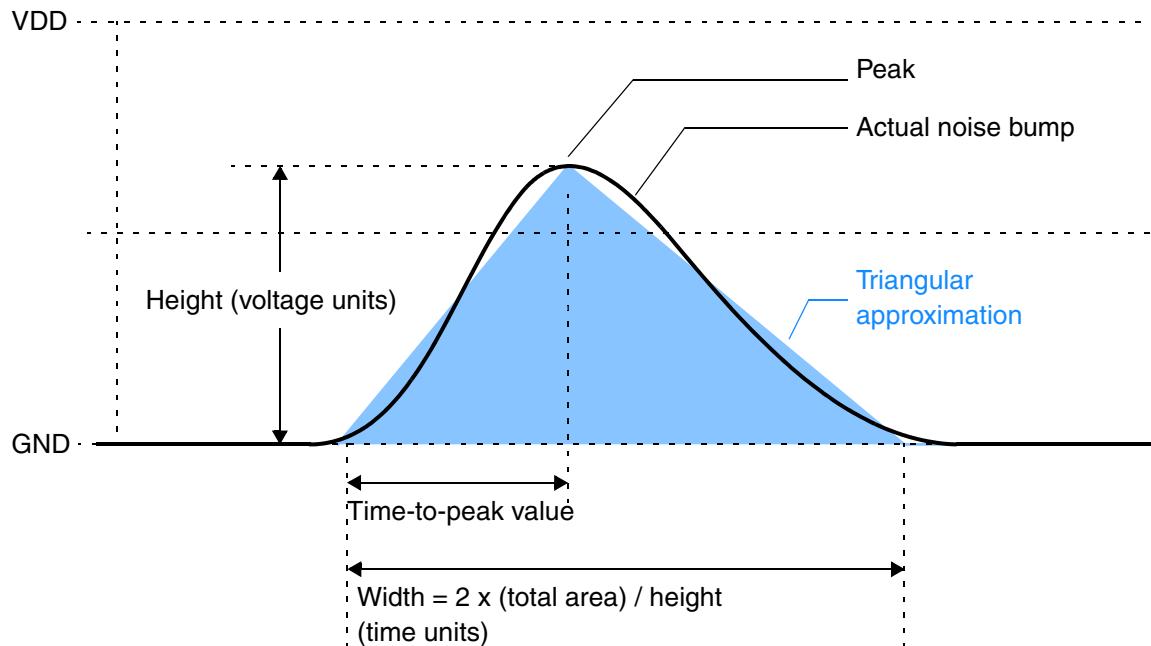
Bumps between the two rail voltages (above low and below high) can cause logic failure due if they exceed the logic thresholds of the technology. Bumps outside of the range between the two rail voltages (below low and above high) are also important because they can forward-bias pass gates at the inputs of flip-flops and latches, allowing incorrect values to be latched.

The following numbers specify the characteristics of a noise bump:

- Height in voltage units
- Total width in time units
- Time-to-peak ratio, which is the time-to-peak divided by the total bump width; a time-peak-ratio of 0.5 indicates a symmetrical bump with equal rising and falling times

To determine the width and time-to-peak value of a noise bump, the tool uses a triangular approximation based on the location of the peak and the area under the curve, as shown in [Figure 14-30](#), where

- Width of the noise bump = $2 \times (\text{area under curve}) / \text{height}$
- Time-to-peak value = $2 \times (\text{area under the curve to the left of the peak}) / \text{height}$

Figure 14-30 Noise Bump Characteristics

Noise Bump Calculation

The tool calculates the cumulative effect of noise from different sources: capacitive cross-coupling, propagation through logic cells and through subnets, and user-defined noise bumps.

Crosstalk Noise

To calculate noise bumps caused by signal crosstalk, the tool considers the cross-coupling capacitance between the aggressor nets and the victim net, the arrival windows of aggressor transitions, the drive characteristics of the aggressor nets, and the steady-state resistance characteristics of the victim net. This calculation is similar to what is done for crosstalk delay analysis, except that it uses the steady-state load characteristics (not the driver characteristics) of the driver on the victim net.

Aggressor nets that contribute to the worst-case noise bump on the victim net are called active aggressors. The remaining aggressor nets do not contribute to the worst-case bump because their timing windows are outside of the worst-case region, their bumps are too small to be significant, or they have been eliminated by you or by logical correlation.

Propagated Noise

Propagated noise on a victim net is caused by noise at an input of the cell that is driving the victim net. The tool can calculate propagated noise at a cell output, given the noise bump at the cell input and the load on the cell output.

The propagation of a noise bump from input to output can either amplify or attenuate the noise bump. In other words, the output noise bump can be either larger or smaller than the input noise bump, depending on the size of the input noise bump and the operating characteristics of the cell.

User-Defined Noise

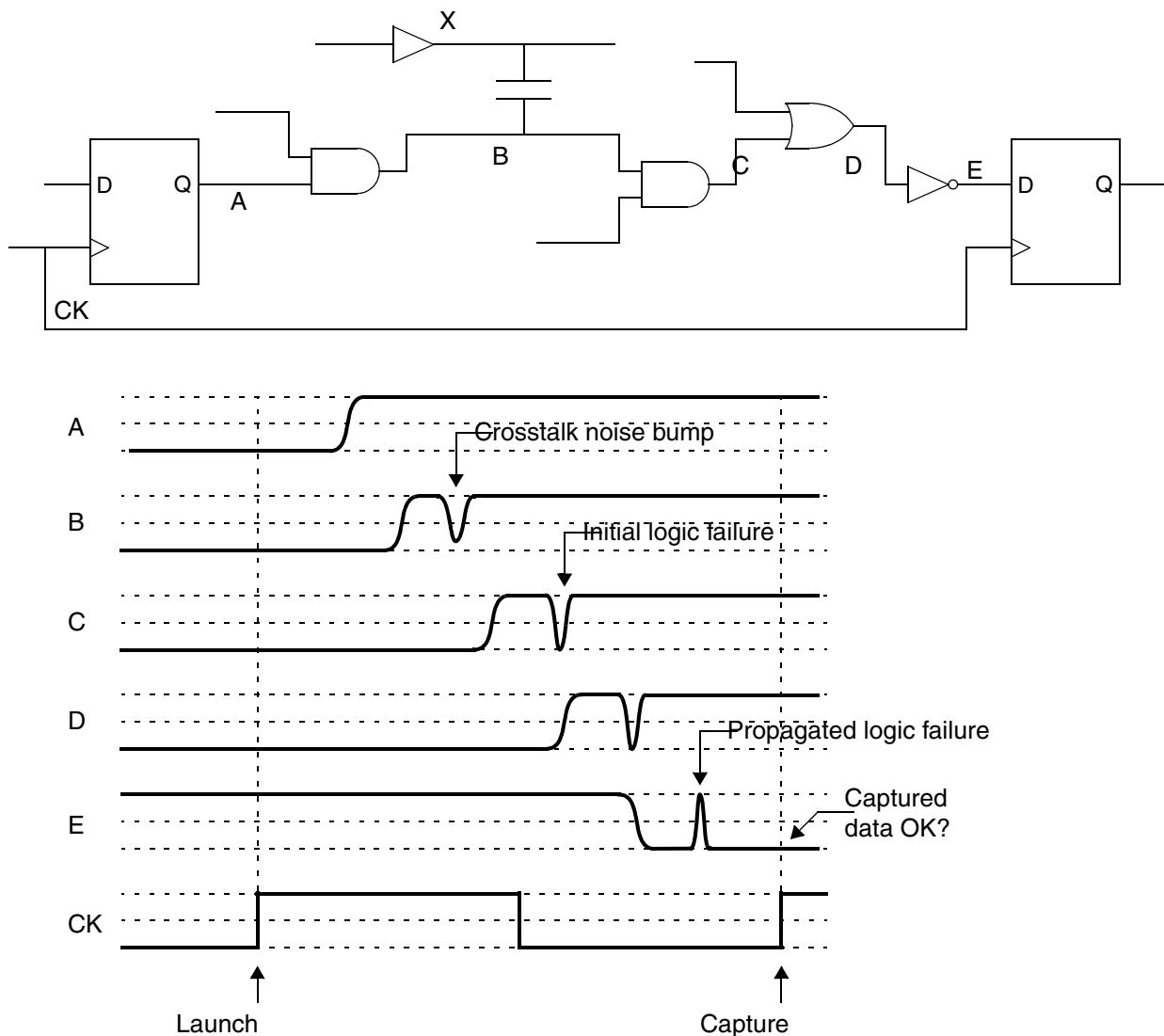
In some cases, propagated noise exists but cannot be calculated. For example if the cell is an extracted timing model or a black-box model, you can specify a noise bump explicitly on any pin or port by using the `set_input_noise` command. This is called *user-defined noise*. User-defined noise can either override or add to any propagated noise for the applicable pin or port.

Noise-Related Logic Failures

In static noise analysis, a logic failure is an incorrect logic value on a net resulting from the propagation of a noise bump from an input to an output of a cell.

A logic failure does not necessarily result in functional failure of the device. For example, consider the circuit shown in [Figure 14-31](#). A rising edge of clock CK launches data through a combinational path from net A to net E. Crosstalk from net X causes a large noise bump on net B, causing a logic failure on net C. The logic failure is propagated down the path to the endpoint at net E.

Figure 14-31 Propagated Logic Failure



If the logic failure is present on net E when the capture edge occurs, an error is latched, resulting in functional failure of the device. On the other hand, if the incorrect value becomes correct again before the capture edge occurs, the device works properly.

There are two possible strategies for repairing logic failures:

- Fix logic failures that are latched and ignore logic glitches that are not latched. The latched violations are reported in the “report at endpoint” mode.

- Fix all logic failures at the source, whether or not they appear to be latched. All logic violations caused by noise are reported in the “report at source” mode. This is the default mode.

The reporting mode is controlled by the `set_noise_parameters` command. In the default (report at source) mode, when the tool detects a logic failure, it does not propagate that failure forward through the path, based on the assumption that the failure is fixed at the source. Instead, it reduces the size of the input noise bump to a fraction of the failure level (0.75 by default) so that noise analysis can continue for cells and nets in the fanout of the failure.

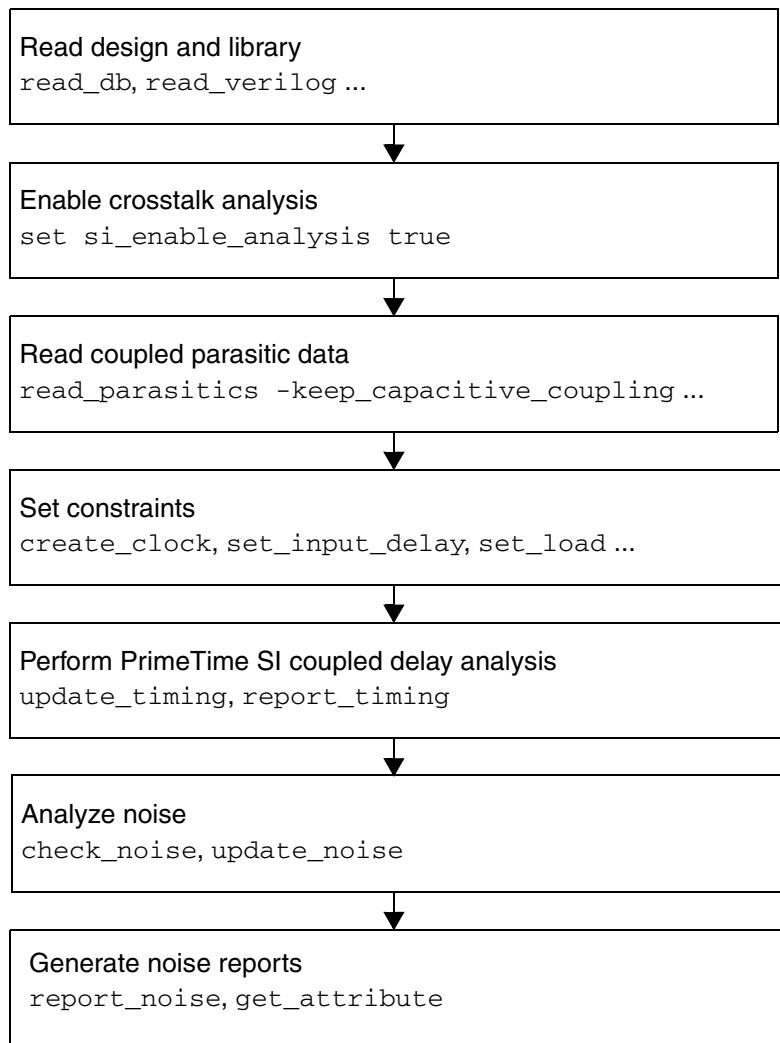
There are several ways to fix a crosstalk noise problem, such as changing the physical routing to avoid cross-coupling, inserting a buffer at the victim net, or increasing the strength of the victim net driver. To confirm the fix and to check for any new crosstalk problems, analyze the fixed design again.

PrimeTime SI Noise Analysis Flow

Static noise analysis requires a PrimeTime SI license. Crosstalk analysis must be enabled by setting the `si_enable_analysis` variable to `true`.

The typical analysis flow is the same as for an PrimeTime SI static timing analysis, with the addition of the `check_noise`, `update_noise`, and `report_noise` commands at the end of the flow. The `check_noise` command checks for the presence of noise models at the driver and load pins the design. The `update_noise` command performs static noise analysis using the aggressor timing windows previously determined by timing analysis. The `report_noise` command generates a noise report showing the locations and values of the worst-case noise bumps. The noise analysis flow is summarized in [Figure 14-32](#).

Figure 14-32 Noise Analysis Flows



The tool supports two types of library noise models, called CCS (composite current source) noise and NLDM (nonlinear delay model). CCS noise is the newer, more advanced technology that provides greater accuracy by calculating the actual response for each cell and each noise bump with SPICE-like accuracy, but with the speed of static timing analysis. CCS noise also offers very fast library characterization. NLDM is the older technology that provides good results for technologies above 65 nm.

Noise Analysis Commands

To invoke noise analysis, generate noise reports, specify noise bumps, and specify cell noise response characteristics, use the commands listed in [Table 14-4](#).

Table 14-4 Noise Analysis Commands

Command	Purpose
check_noise	Checks for the presence and validity of noise models on the driver and load pins, and reports any pins that are not constrained for noise.
get_attribute si_noise_*	For a specified pin, returns a collection of active aggressor nets or returns a string that lists the aggressor nets and their corresponding coupled bump height and width values.
get_noiseViolationSources	Creates a collection of noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
remove_input_noise	Removes noise bump annotations previously set on ports or pins with the <code>set_input_noise</code> command.
remove_noiseImmunityCurve	Removes noise immunity information previously set on ports or pins with the <code>set_noise_immune_curve</code> command.
remove_noiseLibPin	Reverses the effects of the <code>set_noise_lib_pin</code> command.
remove_noiseMargin	Removes noise margins previously set on ports or pins with the <code>set_noise_margin</code> command.
remove_si_noise_analysis	Reverses the effects of the <code>set_si_noise_analysis</code> command.
remove_si_noise_disable_statistical	Reverses the effects of the <code>set_si_noise_disable_statistical</code> command.
remove_steady_state_resistance	Removes steady-state resistance information previously set on cells with the <code>set_noise_margin</code> command.
report_noise	Reports noise effects, including the width, height, and slack of worst-case noise bumps.

Table 14-4 Noise Analysis Commands (Continued)

Command	Purpose
<code>report_noise_calculation</code>	Reports the calculation of noise effects for a specified net or pin and noise bump type. The report shows the reasons for that individual aggressors are active or inactive, the noise contribution from individual sources (aggressors, propagated noise, user-specified noise), and other information used for noise bump calculation.
<code>report_noise_parameters</code>	Reports the settings made with the <code>set_noise_parameters</code> command.
<code>report_noiseViolation_sources</code>	Reports the noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
<code>reset_noise_parameters</code>	Resets all the <code>set_noise_parameters</code> parameters to their default settings.
<code>set_input_noise</code>	Annotates a noise bump with specified width and height characteristics on a port or pin, either overriding or adding to any propagated noise at that location.
<code>set_noise_derate</code>	Derates (modifies) the calculated noise height or width by a specified fraction or absolute amount.
<code>set_noise_immune_curve</code>	Specifies the three coefficient values that determine the noise immunity curve at an input pin of a library cell or output port of the design. The tool uses this information to determine whether a noise bump of a given height and width causes a logical failure.
<code>set_noise_lib_pin</code>	Sets the noise characteristics of an input pin or output pin to match the noise characteristics of a specified library pin.
<code>set_noise_margin</code>	Specifies the bump-height noise margins for an input pin of a library cell or output port of the design. The tool uses this information to determine whether a noise bump of a given height at a cell input causes a logical failure at the cell output.
<code>set_noise_parameters</code>	Specifies options for noise analysis: effort level, aggressor arrival times, beyond-rail analysis, noise propagation, and source/endpoint noise reporting.

Table 14-4 Noise Analysis Commands (Continued)

Command	Purpose
<code>set_si_noise_analysis</code>	Includes or excludes specified nets for crosstalk noise analysis.
<code>set_si_noise_disable_statistical</code>	Disables statistical analysis for the specified nets when using composite aggressor analysis.
<code>set_steady_state_resistance</code>	Specifies the steady-state drive resistance for an output pin of a library cell or input port of the design. The tool uses this information to calculate crosstalk noise bump characteristics.
<code>update_noise</code>	Performs a noise analysis using the parameters set with the <code>set_noise_parameters</code> command. Performs a timing update (<code>update_timing</code>) if needed to get arrival window data.

For more information, see the following topics:

- [Performing Noise Analysis](#)
- [Reporting Noise Analysis Results](#)
- [Setting Noise Bumps](#)
- [Performing Noise Analysis with Incomplete Library Data](#)

Performing Noise Analysis

To perform noise analysis, set the `si_enable_analysis` variable to `true`, and use the `check_noise`, `update_noise`, and `report_noise` commands. These commands operate like `check_timing`, `update_timing` and `report_timing` commands, but for noise analysis rather than timing analysis.

The `update_noise` command performs crosstalk noise analysis of the current design. It invokes a full timing update (like using the `update_timing` command) if a timing update has not already been done. After completion of the noise analysis, you can report the results with the `report_noise` command.

Set the parameters for noise analysis with the `set_noise_parameters` command. To view the current settings, use the `report_noise_parameters` command. To return all noise parameters to their default settings, use the `reset_noise_parameters` command.

To explicitly exclude specific nets from crosstalk noise analysis, use the `set_si_noise_analysis` command. This command works very much like the `set_si_delay_analysis` command, except that it applies to crosstalk noise analysis rather than crosstalk delay analysis. To reverse the effects of the command, use the `remove_si_noise_analysis` command.

check_noise Command

The `check_noise` command can be executed before the `update_noise` command to validate the correctness of a design with respect to noise analysis. It checks for the presence and validity of noise models on all load pins and driver pins, and reports any pins that are not constrained for noise analysis. Running the `check_noise` command can quickly detect many types of noise modeling problems before you spend time using the `update_noise` command.

The `check_noise` command options are:

- The `-include` option specifies which types of noise model checking to perform, noise immunity on load pins or noise models on driver pins, or both. By default, only noise immunity checking is performed.
- The `-beyond_rail` option causes checking for beyond-rail as well as between-rail noise analysis. By default, only between-rail noise checking is performed.
- The `-verbose` option reports individual pins as well as generating a summary report.
- The `-nosplit` option prevents the splitting of long lines in the report.
- The `si_noise_immunity_default_height_ratio` variable controls the default margin value. The default of this variable is 0.4 (40% of VDD). If the variable is set to 1.0, no default noise immunity is used (backward compatibility).

By default, the `check_noise` command report summarizes the number of driver and load pins with each type of noise constraint, as shown in the following example:

Noise driver pin check:

Noise driver type	above_low	below_high
CCS noise	11	11
library IV curve	0	0
library resistance	0	0
equivalent library pin	4	4
user resistance	1	1
none	2	2

Noise load pin immunity check:

Noise immunity type	above_low	below_high
user hyperbolic curve	1	1
user margin	2	2
library immunity table	0	0
library hyperbolic curve	0	0
library CCS noise immunity	14	14
library DC noise margin	0	0
default margin	5	5
none	0	0

To ensure detection of noise violations throughout the design, verify that all pins are constrained for noise analysis. The pins reported in the “default margin” column are not constrained, so the default margin value is used. The number of pins reported in the “none” row of the report must be zero. For information about specifying noise immunity, see [Noise Immunity](#).

The `check_noise` command is typically used as follows:

1. After crosstalk timing analysis, but before noise analysis, run the `check_noise` command to check all driver and load pins for noise constraints. If any pins are found without noise constraints, run the `check_noise` command with the `-verbose` option to get a detailed list of pins that lack constraints.
2. Constrain the pins for noise as needed, using the `set_noise_lib_pin`, `set_noise_margin`, and `set_noise_immunity_curve` commands.
3. Run the `check_noise` command again to verify that all pins are constrained for noise analysis.

When the `check_noise` command confirms that all pins are constrained for noise, you can proceed to noise analysis with the `update_noise` command.

Aggressor Arrival Times

If you use the `-ignore_arrival` option of the `set_noise_parameters` command, the tool ignores aggressor arrival windows and assumes that all aggressor nets switch at the same time for each victim net throughout the design, resulting in a conservative analysis. If noise violations are reported, you can run another analysis using arrival windows to see if the violations are eliminated.

Beyond-Rail Analysis

By default, the `update_noise` command analysis only considers between-the-rails noise bumps, which are typically the more significant ones to consider. However, beyond-rail noise bumps (above high and below low) can also cause incorrect data to be latched due to forward-biasing of pass transistors at flip-flop and latch inputs. To have the tool consider beyond-rail noise conditions at the cost of additional runtime, use the `set_noise_parameters` command with the `-include_beyond_rails` option.

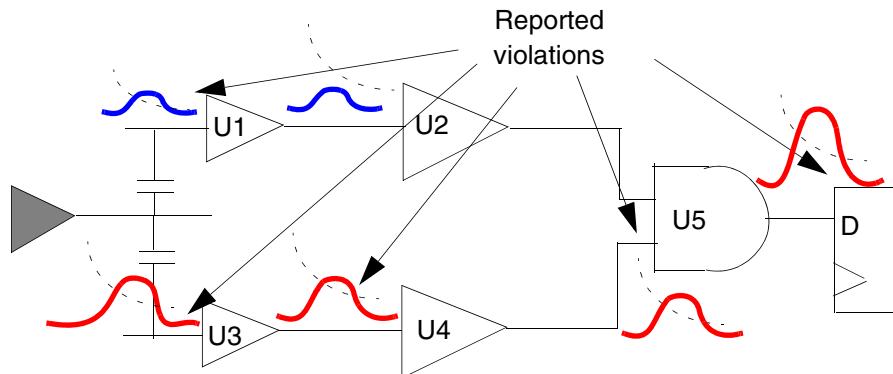
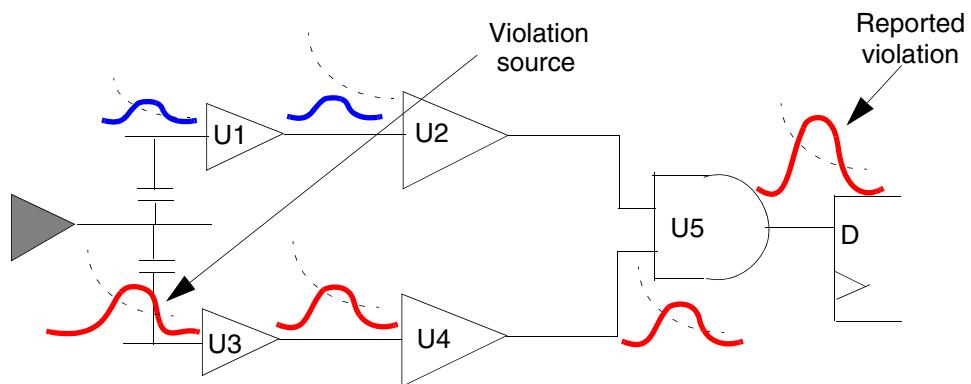
Noise Propagation

By default, the tool considers only crosstalk noise and user-specified noise, not propagated noise. This default mode is suitable for an initial analysis, when you want to quickly find the worst violations. For a more accurate noise analysis at the cost of additional runtime, enable noise propagation analysis by using the `-enable_propagation` option of the `set_noise_parameters` command.

Reporting Noise at Source or Endpoint

The tool offers two analysis reporting modes, called “report at source” and “report at endpoint.” In the “report at source” mode, the tool reports violations at the source of each noise violation. In the “report at endpoint” mode, the tool reports violations only at endpoints where noise propagation stops, such as sequential cells.

Figure 14-33 and Figure 14-34 show the two noise reporting modes.

Figure 14-33 Report at Source*Figure 14-34 Report at Endpoint*

In both figures, a violation occurs when the noise bump crosses the dotted noise immunity curve. The noise injected at the input of U3 is propagated through U4 and U5 and reaches D. The noise injected at U1, however, is not propagated beyond U2 because of the noise immunity of U2.

As shown in [Figure 14-33](#), the input pins of U1, U3, U4, U5, and D in the “report at source” mode are reported as violations because they each have negative slack. In the “report at endpoint” mode, as shown in [Figure 14-34](#), only the input pin of D, the noise propagation endpoint, is reported as a violation.

A noise startpoint can be:

- An output pin of a flip-flop
- An output pin of a level-sensitive latch
- An input port

- An output pin of a multistage cell, which is a cell with multiple levels of transistor stages, such as a flip-flop or macro

A noise endpoint can be:

- A data, clock, or asynchronous pin of a flip-flop
- An input pin of a level-sensitive latch
- An output port
- Any combinational logic pin where the noise exceeds a threshold of 75 percent of VDD, which can be controlled by setting the `si_noise_endpoint_height_threshold_ratio` variable.
- An input pin of a multistage cell

The “report at endpoint” mode produces a more concise report because it includes only noise violations that are latched as incorrect data. The “report at source” mode produces a more complete report that includes all noise immunity violations, whether or not they are latched. The default mode is “report at source.”

To set the reporting mode to “report at endpoint,” use this command:

```
pt_shell> set_noise_parameters -analysis_mode report_at_endpoint
```

To set the reporting mode back to the default, “report at source”, use this command:

```
pt_shell> set_noise_parameters -analysis_mode report_at_source
```

Changing the mode requires a full timing update, so to save time, set the desired mode before you use the `update_noise` command.

In the “report at endpoint” mode, if you get a report of endpoint violations and you want identify the sources that caused the violations, you can use the `report_noiseViolationSources` or `get_noiseViolationSources` command.

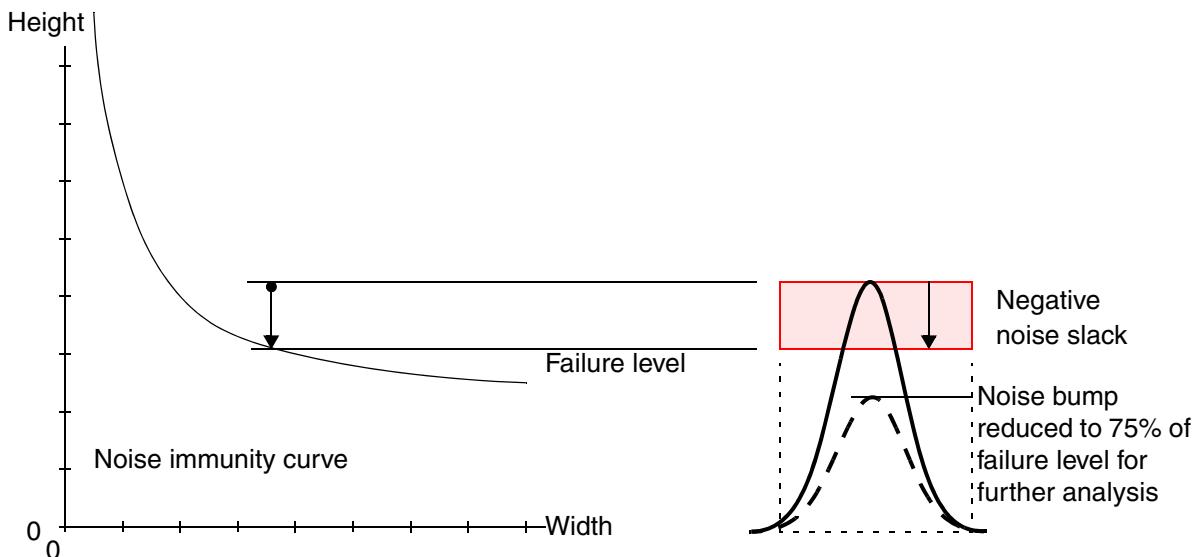
Derating Noise Results

The `set_noise_derate` command modifies the calculated noise bump sizes by a specified factor or absolute amount. It works like the `set_timing_derate` command, except that it modifies noise bump sizes rather than path delays. The derating factors affect the reported bump sizes and noise slacks. In the `set_noise_derate` command, you specify the types of noise bumps (above/below high/low), the parameters to be modified (height offset, height factor, and width factor), and the factor or absolute amount of modification.

Noise Failure Propagation Limit

In the “report at source” mode, when the tool detects a functional failure caused by a noise bump, it does not propagate the incorrect logic value forward through the path. Instead, it reduces the size of the input noise bump to a fraction of the failure level so that noise analysis can continue for cells and nets in the fanout of the failure. The default factor is 0.75, as shown in [Figure 14-35](#).

Figure 14-35 Input Noise Bump Reduction for Propagation Analysis



To specify a different reduction factor, set the `si_noise_limit_propagation_ratio` variable to the desired ratio, which must be a value between 0.0 and 1.0. Note that in the “report at endpoint” mode, the entire noise bump is propagated, so this variable has no effect.

Irrespective of this variable setting, the height of any propagated noise bump is limited to VDD (the rail-to-rail voltage swing).

Noise Analysis with Composite Aggressors

Some complex designs require an efficient method of analyzing multiple aggressors to a single victim net. In general, such designs have the following characteristics:

- A large number of aggressors per victim net
- Little or no filtering of aggressors
- Noise calculations are performed in high effort mode

If your design has these characteristics, you can use composite aggressor analysis. Composite aggressor analysis aggregates the effects of multiple aggressors into a single

virtual aggressor, thereby reducing the computational complexity cost and improving the runtime and memory utilization without affecting accuracy.

By default, composite aggressor mode for noise analysis is disabled. To enable it, set the `si_noise_composite_aggr_mode` variable to `statistical`. This mode applies statistical analysis to the composite aggressor to reduce its overall effect on the victim net.

If you want to use statistical mode, but have certain nets in your design for which you do not want to statistically reduce the aggressor effect, you can disable statistical analysis for just those nets by using the following commands. These commands take a list of nets as an argument, and have no effect if a net is not part of the composite aggressor group:

- `set_si_noise_disable_statistical`
- `remove_si_noise_disable_statistical`

To see which aggressors are part of a composite aggressor, use the `report_noise_calculation` command. Aggressors that are part of a composite aggressor are labeled with a “C” in the report.

For more information, see [Crosstalk Analysis with Composite Aggressors](#).

Incremental Noise Analysis

After a full noise update with the `update_noise` command, the tool can sometimes perform a fast “incremental” noise update to analyze the effects of certain design changes. An incremental update takes just a fraction of the time needed for a full noise update, because only the portions of the design affected by the changes are analyzed.

Depending on previous usage of the `update_noise` command and the types of changes performed on the design, the `update_noise` command might invoke an incremental noise update, a full noise update, an incremental timing update with a full noise update, or a full timing update with a full noise update. The tool generally uses the fastest possible type of update that gives accurate results.

You can force a complete noise update, irrespective of the changes made to the design, by using the `-full` option of the `update_noise` command. In that case, if a timing update is necessary due to a change such as the `size_cell` command, the tool performs a full (not incremental) timing update as well.

Reporting Noise Analysis Results

To report noise analysis results, use these commands:

- [report_noise](#)
- [report_noise_calculation](#)
- [get_attribute](#) and [report_attribute](#)

report_noise

The `report_noise` command provides information about the worst-case noise bumps on one or more nets. The command options let you specify the types of noise reported (above/below high/low); the pins, ports, or nets of the design to be reported; and the type of information included in the report.

If a noise update has not been done already, using the `report_noise` command invokes the `update_noise` command to generate the timing window information needed for noise analysis.

By default, the `report_noise` command by itself, without any options, reports the bump characteristics and noise slack for the pin with the smallest (or most negative) slack for each type of noise bump. For example,

```
pt_shell> report_noise
...
analysis_mode report_at_source
slack type: area

noise_region: above_low
pin name (net name)      width    height    slack
-----
tmp2f_reg/D (buf2_1f)     0.07     0.12     0.12

noise_region: below_high
pin name (net name)      width    height    slack
-----
U3/A (buf0_2f)           0.22     0.03     0.37
...
```

Width values are in library time units such as nanoseconds, height values are in library voltage units such as volts, and noise slack area values are in library voltage-time units such as volt-nsec.

To restrict the scope of the report to certain types of noise bumps, use `-above` or `-below` and `-high` or `-low`. For example,

```
pt_shell> report_noise -above -low
...
noise_region: above_low
pin name (net name)      width    height    slack
-----
tmp2f_reg/D (buf2_1f)     0.07     0.12     0.12
```

To report multiple pins having the worst noise slack, use the `-nworst` option. For example,

```
pt_shell> report_noise -above -low -nworst_pins 4
...
noise_region: above_low
pin name (net name)      width    height    slack
-----
tmp2f_reg/D (buf2_1f)    0.07     0.12     0.12
U1/A (tmp2f)             0.24     0.01     0.39
U2/A (tmp2f)             0.24     0.01     0.39
U3/A (buf0_2f)           0.23     0.01     0.40
```

The `-slack_type` option specifies the method used for measuring noise slack: `height`, `area`, or `area_percent`, as described in [Noise Slack](#). The default is `height`. The slack `height` is a more direct representation of the noise violation on a pin. The `-slack_lesser_than` option restricts the report to pins that have noise slack worse than a specified amount. The `-all_violators` option restricts the report to pins that have negative noise slack.

To restrict the scope of the report to specific pins, ports, or nets, specify a list of objects in the command. If the specified object is a pin, the command reports the noise on that pin. If the specified object is a net, the command reports the noise on all load pins in the net. For example, to restrict the scope of the report to load pins in net1 and net2:

```
pt_shell> report_noise -above -low {net1 net2}
```

To restrict the scope of the report to just the data pins, clock pins, or asynchronous pins of all registers, use the option `-data_pins`, `-clock_pins`, or `-asynch_pins`. For example,

```
pt_shell> report_noise -data_pins
```

To get a more detailed report showing separately the noise contribution of different aggressor nets and noise propagation, use the `-verbose` option. For example,

```
pt_shell> report_noise -verbose -above -low
...
```

```
noise_region: above_low
pin name (net name)      width    height    slack
-----
tmp2f_reg/D (buf2_1f)
Aggressors:
CK                      0.07     0.12
net2                    0.04     0.02
Total:                  0.07     0.12     0.12
```

Information about the calculation of these noise bumps is available by using the `report_noise_calculation` command.

Noise bump on the data pin of the flip-flop is reported only if the noise bump overlaps with the setup-hold window of the data pin. If the noise bump is induced away from the setup-hold window, then the noise bump is not reported. In such cases, the `report_noise_calculation` or `report_noise -verbose` command reports the aggressors and their contribution, but the total noise bump is reported as 0. This not only applies to the data pin of the flip-flop, but also to the data pin of any latch device that has setup-hold window.

When CCS noise models are used, the `report_noise` command reports slack as “positive” for small bumps where the exact slack number is not important. If you want to see exact numbers or noise information about non-endpoint pins under these conditions, use the `report_noise_calculation` command.

report_noise_calculation

The `report_noise_calculation` command reports the calculation of the noise bump on a net arc. In the command, you specify the type of noise bump to be reported (above/below high/low), the startpoint of the net arc, and endpoint of the net arc. The startpoint is the driver pin or driver port of a victim net and the endpoint is a load pin or load port on the same net.

Here is an example of a report generated by the `report_noise_calculation` command:

```
pt_shell> report_noise_calculation -below -high \
           -from buf2/ZN -to buf5/I
Units: 1ns 1pF 1kOhm

Analysis mode : report_at_source
Region : below_high
Victim driver pin : buf2/ZN
Victim driver library cell : mylib/INVD3
Victim net : I2
Victim driver effective capacitance : 0.200827

Steady state resistance source : library set CCS
noise iv curve
Driver voltage swing : 1.080000
Noise derate height offset : 0.000000
Noise derate height scale factor : 1.000000
Noise derate width scale factor : 1.000000
Noise effort threshold : 0.000000
Noise composite aggressor mode : disabled
```

Noise calculations:

Attributes:

A	- aggressor is active
C	- aggressor is a composite aggressor
D	- aggressor is analyzed with detailed engine
E	- aggressor is screened due to user exclusion
G	- aggressor is analyzed with gate level simulator
I	- aggressor has infinite window
L	- aggressor is screened due to logical correlation
S	- aggressor is screened due to small bump height
X	- aggressor is screened due to aggressor exclusion

	Height	Width	Area	Aggressor Attr.
<hr/>				
Aggressors:				
I1	0.300604	0.786466	0.118207	A I D
I3	0.300604	0.786466	0.118207	A I D
Total:	0.497124	0.919737	0.228612	G

Noise slack calculation:

Constraint type: library CCS noise immunity

	Height	Area
<hr/>		
Required	0.581570	(0.581570 * 0.919737)
Actual	0.497124	(0.497124 * 0.919737)
<hr/>		
Slack	0.084445	0.077668

The command uses the noise analysis settings set by the `set_noise_parameters` command. It invokes the `report_noise` command if a noise update has not been performed already.

The report first identifies the victim driver net and pin. It also shows any derating factors set by the `set_noise_derate` command. It shows the noise bumps resulting from each aggressor net and from propagation of noise from the previous stage of the driver.

A column labeled “Aggressor Attributes” shows additional information about each effective aggressor. (Aggressors that have been removed by filtering are not included in this report.) The letters in this column indicate the following conditions:

- A: The aggressor net is active. It contributes to the worst-case noise bump, taking into consideration the possible overlap times of all aggressor nets.
- C: The aggressor is a composite aggressor composed of two or more smaller aggressors working together. See [Crosstalk Analysis with Composite Aggressors](#).
- D: The effects of the aggressor net were calculated with the detailed (high-effort) algorithm.

- E: The aggressor net is not active because it has been excluded from consideration by the `set_si_noise_analysis` command (see [Performing Noise Analysis](#)).
- G: The noise was analyzed by gate-level simulation using CCS noise models. See [Noise Modeling With CCS Noise Data](#).
- I: The aggressor net uses an infinite timing window; it has no fixed timing relationship with the victim net. This happens when the `-ignore_arrival` mode has been set with the `set_noise_ parameters` command or when the victim net and aggressor nets are in different clock domains.
- L: The aggressor net is not active due to logical correlation with the victim net or with other aggressors. For more information, see [Logical Correlation](#).
- S: The aggressor net is not active because it has been screened out. An internal prescreening analysis has determined that the aggressor bump is too small to be significant.
- X: The aggressor net is not active because it has been excluded from consideration by the `set_si_aggressor_exclusion` command. For more information, see [Excluding Aggressor-to-Aggressor Nets](#).

get_attribute and report_attribute

Crosstalk information is stored in attributes associated with pins, nets, ports, timing points, timing arcs, library pins, and library timing arcs. To view this information, use the `get_attribute` or `report_attribute` command.

By default, if a noise update has not been done already, the `get_attribute` or `report_attribute` command automatically invokes the `update_noise` command to determine the attribute value.

To control automatic noise updates during the `report_attribute` command, set the `si_noise_skip_update_for_report_attribute` variable, as described in [Table 14-5](#).

Table 14-5 Attribute Reporting Behavior Depends on the si_noise_skip_update_for_report_attribute Variable Setting

	<code>si_noise_skip_update_for_report_attribute = false (default)</code>	<code>si_noise_skip_update_for_report_attribute = true</code>
If a noise update has been done...	The <code>report_attribute</code> command reports attribute values immediately.	The <code>report_attribute</code> command reports attribute values immediately.
If a noise update has not been done...	The <code>report_attribute</code> command triggers an implicit noise update and then reports attribute values.	The <code>report_attribute</code> command does not trigger an implicit noise update and does not report attribute values.

If you change a noise constraint, the `report_attribute` command behaves as described in [Table 14-5](#) based on the variable setting.

Note:

The `si_noise_skip_update_for_report_attribute` variable does not affect the `get_attribute` command.

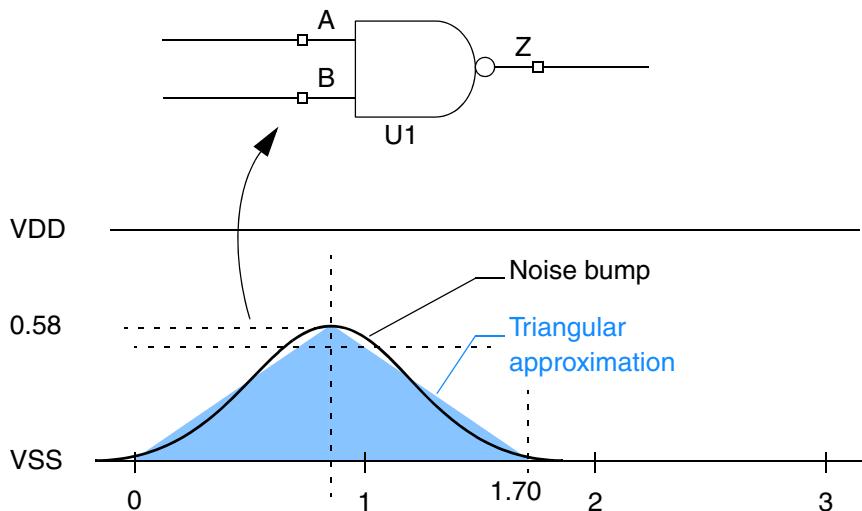
Setting Noise Bumps

Rather than allow the tool to calculate propagated noise bumps, you can explicitly specify a noise bump at any port or pin. To do so, use the `set_input_noise` command. In the command, you specify the port or pin on which to apply the noise and the characteristics of the noise bump: the type (above/below high/low), the width in library time units, and the height in library voltage units (always entered as a positive number). For example,

```
pt_shell> set_input_noise -above -low \
    -width 1.70 -height 0.58 [get_pins U1/B]
```

This creates a noise bump on pin U1/B with the waveform characteristics shown in [Figure 14-36](#).

Figure 14-36 Noise Bump Created With the `set_input_noise` Command



The tool treats this injected noise as propagated noise from the driver of the net connected to the pin. By default, this noise bump overrides any propagated noise that would otherwise be calculated from the driver of the net. However, if you use the `-add_noise` option of the `set_input_noise` command, the noise is added to any existing propagated noise or previously added noise.

You can specify propagated noise on an output pin rather than a load pin. In that case, unless you use the `-add_noise` option, the specified noise bump overrides any propagated

noise that would otherwise be calculated from the driver. The specified noise bump is propagated through the subnets and attenuated by the parasitic capacitance and resistance specified for the net, until the propagated noise reaches each of load pin on the net.

When extracted timing models or other hierarchical timing models are used in a design, the specified noise bump can be used to model the worst-case noise that can occur at each output port of the timing model. The `extract_model` command includes a set of `set_input_noise` commands in the output constraint script to model the propagated noise at the output pins of the model.

Performing Noise Analysis with Incomplete Library Data

To specify cell noise response characteristics in the absence of library-specified characteristics, or to override the library-specified characteristics at specified points, use these commands:

- `set_noise_lib_pin`
- `set_noise_immunity_curve`
- `set_noise_margin`
- `set_steady_state_resistance`

To remove the user-specified noise response characteristics, use the following corresponding commands:

- `remove_noise_lib_pin`
- `remove_noise_immunity_curve`
- `remove_noise_margin`
- `remove_steady_state_resistance`

`set_noise_lib_pin`

To specify that the noise characteristics of a pin in the design are the same as the noise characteristics of a library pin, use the `set_noise_lib_pin` command. This command is useful when some library cells have noise information, but other cells do not. In the following example, the macro_cell/A input pin inherits the noise characteristics of the ccs_noise_lib/inv/I library cell input pin. The tool uses this information to calculate the noise immunity of the pin macro_cell/A.

```
pt_shell> set_noise_lib_pin [get_pins macro_cell/A] ccs_noise_lib/inv/I
```

Similarly, in the following example, the macro_cell/Z output pin inherits the noise characteristics of the ccs_noise_lib/buf/Z library cell output pin. The tool uses this information to calculate the noise bump induced on the net driven by macro_cell/Z.

```
pt_shell> set_noise_lib_pin [get_pins macro_cell/Z] ccs_noise_lib/buf/z
```

set_noise_immunity_curve

The `set_noise_immunity_curve` command specifies the noise immunity characteristics at an input of a library cell or at an output port of the design. The tool uses this information to determine whether a noise bump at the input causes a logic failure. To learn about logic failures, see [Noise Immunity](#).

A noise immunity curve specifies the maximum allowable bump height as a function of the bump width. This function is defined by three positive coefficients, as described in [Noise Immunity Curves](#).

With the `set_noise_immunity_curve` command, you specify

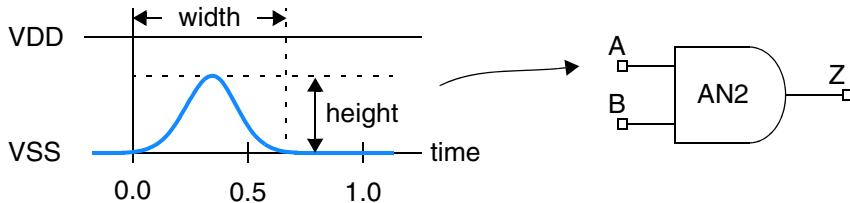
- The type of noise bump: above-high, below-high, above-low, or below-low. To fully specify the noise immunity characteristics of a cell or design, you need one `set_noise_immunity_curve` command for each type of noise bump.
- The three positive coefficient values that define the curve
- The input pin of a library cell or the output port of the design to which the curve applies

For example,

```
pt_shell> set_noise_immunity_curve -above -low \
    -width 0.00 -height 0.58 -area 0.0064 lib_name/AN2/A
```

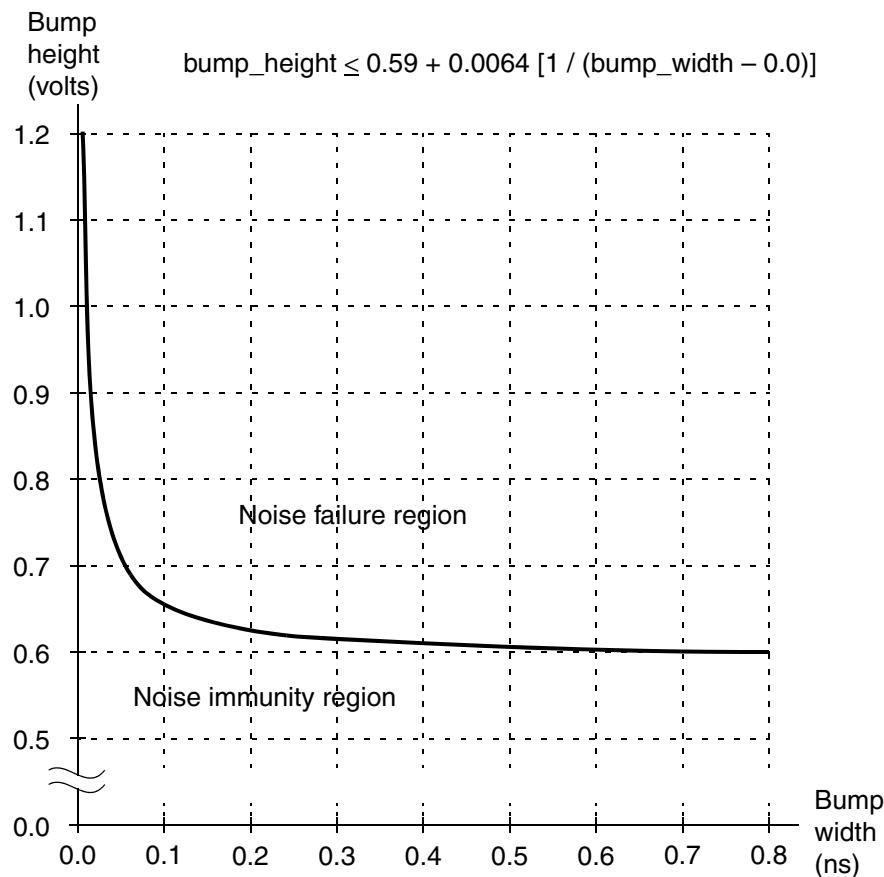
This command defines the above-low noise immunity curve for input A of library cell AN2, as shown in [Figure 14-37](#). [Figure 14-38](#) shows the resulting noise immunity curve.

Figure 14-37 User-Defined Noise Immunity Curve



```
set_noise_immunity_curve -above -low \
    -width 0.0 -height 0.58 -area 0.0064 lib_name/AN2/A
```

Figure 14-38 Plot of User-Defined Noise Immunity Curve



For more information, see [Noise Immunity Curves](#).

set_noise_margin

Where there is no noise immunity curve specified by the library or by the `set_noise_immunity_curve` command, the tool uses noise margins based on bump heights, as described in [Bump Height Noise Margins](#).

The `set_noise_margin` command specifies the noise margin at an input of a library cell or at an output port of the design. For a library cell, this information overrides the library-specified voltage noise margins.

With the `set_noise_margin` command, you specify the type of noise bump (above/below high/low), the bump height failure threshold for the input, and the input pin of a library cell or input port the design to which the setting applies. For example,

```
pt_shell> set_noise_margin -above -low 0.4 lib_name/AN2/A
```

All noise margin values are entered as positive numbers, including those for below-high and below-low noise bumps.

set_steady_state_resistance

The `set_steady_state_resistance` command specifies the drive resistance at an output of a library cell or at an input port of design. The tool uses this information to determine the size of voltage bumps in the presence of crosstalk noise.

With the `set_steady_state_resistance` command, you specify the type of noise bump (above/below high/low), the drive resistance in library resistance units such as ohms or Kohms, and the output pin of a library cell or the output port the design to which the setting applies. For example,

```
pt_shell> set_steady_state_resistance -above -low 0.042 \
           lib_name/AN2/Z
```

All resistance values are entered as positive numbers.

Noise Modeling With CCS Noise Data

CCS noise uses an advanced, current-based driver model that performs noise analysis with SPICE-like accuracy. Using an adaptive algorithm for analysis, the tool precisely calculates not only injected crosstalk noise bumps, but also propagated noise bumps and driver weakening effects. The current-based CCS noise model provides the accuracy needed for process technologies at 65 nm and below.

With CCS noise models, the tool computes nonlinear noise bump waveforms on-the-fly with excellent correlation with SPICE simulations at speeds capable of handling designs containing millions of gates. Unlike NLDM models that use static noise immunity curves, CCS noise models allow the tool to calculate noise immunity dynamically, including the effects of nonmonotonic noise waveforms and noise propagation.

When CCS noise models are present, a victim net is first analyzed with a fast macro model engine. The noise bumps calculated with this engine are generally conservative. If the magnitude of the calculated noise bump is smaller than the transistor threshold voltage of the receiver pins, no further analysis is done because such a small noise bump does not affect the output of the receiver cell. In such cases, noise slack is simply reported as “POSITIVE”. On the other hand, if the noise bump calculated with the macro model engine exceeds the receiver transistor threshold voltage, the noise bumps are further calculated using a more accurate CCS noise gate-level simulation engine. This tiered approach provides a good tradeoff between accuracy and performance. The nets that have potential noise violations are analyzed with an accurate engine while the nets with smaller noise bumps are analyzed with a fast engine.

Noise Immunity

The tool uses the following order of precedence when choosing which noise immunity information to use:

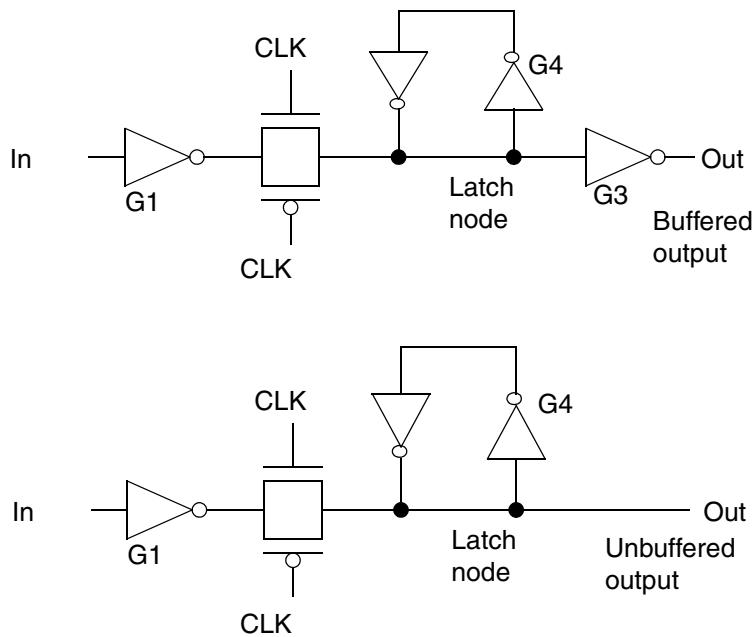
1. Static noise immunity curve annotated using the `set_noise_immunity_curve` command
2. DC noise margin annotated using the `set_noise_margin` command
3. Arc-specific noise immunity curve from library
4. Pin-specific noise immunity curve from library
5. CCS noise model from library
6. DC noise margin from library

For example, if you specify noise immunity curve information using the `set_noise_immunity_curve` or `set_noise_margin` command, the tool uses that information for noise slack calculation, even if the library has CCS noise information.

CCS Noise Analysis for Unbuffered-Output Latches

Level-sensitive latches are often used in high-performance designs because they have smaller data-to-output and clock-to-output delays than flip-flops. [Figure 14-39](#) shows two possible ways to build a level-sensitive latch, with and without an output buffer. Both of these examples use an input buffer, a pass gate, and an inverter loop that holds the latched value on a latch node.

Figure 14-39 Latch Circuits With Buffered and Unbuffered Outputs



The latch with an output buffer is resistant to noise at the output because the buffer isolates the latch node from the output node. The latch circuit without an output buffer is faster and uses less power but is very sensitive to noise at the output.

Noise analysis is performed for an output pin if its attribute `is_unbuffered` is set to `true` in the library and the pin is characterized for noise. The `is_unbuffered` pin attribute is supported by Liberty CCS modeling syntax and Library Compiler, as described in the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

You can specify the name of an unbuffered cell output pin in the `report_noise` command, producing a report similar to what you get when you specify an input pin or bidirectional pin. For example,

```
pt_shell> report_noise [get_pins I2/Z]
...
analysis mode: report_at_source
slack type: area

noise_region: above_low
pin name (net name)      width    height    slack
-----
I2/Z (P4)                1.03     2.03     -1.99

noise_region: below_high
pin name (net name)      width    height    slack
-----
I2/Z (P4)                1.04     2.07     -2.04
```

Noise Modeling With Nonlinear Delay Models

When nonlinear delay models (NLDM) are used for modeling noise, the noise response characteristics of the cells must be specified either in the Synopsys .lib logic library or by using PrimeTime SI commands. These are the types of noise response information used in noise analysis:

- The steady-state I-V (current-voltage) characteristics of the cell outputs. This information is needed to determine the size of noise bumps resulting from crosstalk
- The noise immunity characteristics of the cell inputs, either in terms of allowable noise bump heights and widths (noise immunity curves) or in terms of bump heights alone. This information is needed to determine whether a noise bump of a given size at the input causes a logic failure at the output
- The width and height of propagated noise bumps at the cell outputs, given the width and height of the noise bumps at the cell inputs and the load on the output. This information is needed to determine the size of noise bumps resulting from propagation

It is better to specify the noise response characteristics in the library. However, if the library lacks noise response information, or if you want to override the library-specified information, you can use PrimeTime SI commands to specify the steady-state I-V characteristics, noise immunity characteristics, or both.

If noise propagation information is not available in the library, you can use PrimeTime SI commands to specify explicitly the noise bumps at any ports or pins in the design.

The library syntax offers more specification features and flexibility than PrimeTime SI commands. In libraries you can specify piecewise linear models, polynomial models, and

noise propagation models of various types, in addition to the methods supported by PrimeTime SI commands.

In the absence of cell characterization data obtained in the laboratory, you can use a circuit simulator such as SPICE to get theoretical noise response characteristics, and use that information in the library or in PrimeTime SI commands that specify noise response characteristics. This process can provide detailed static noise analysis results.

If cell noise characteristics are not specified in the library and not specified by PrimeTime SI commands, the tool still performs noise analysis by estimating noise characteristics from the library-specified cell timing and slew characteristics. However, noise analysis under these conditions cannot detect logic failures and cannot calculate noise propagation effects.

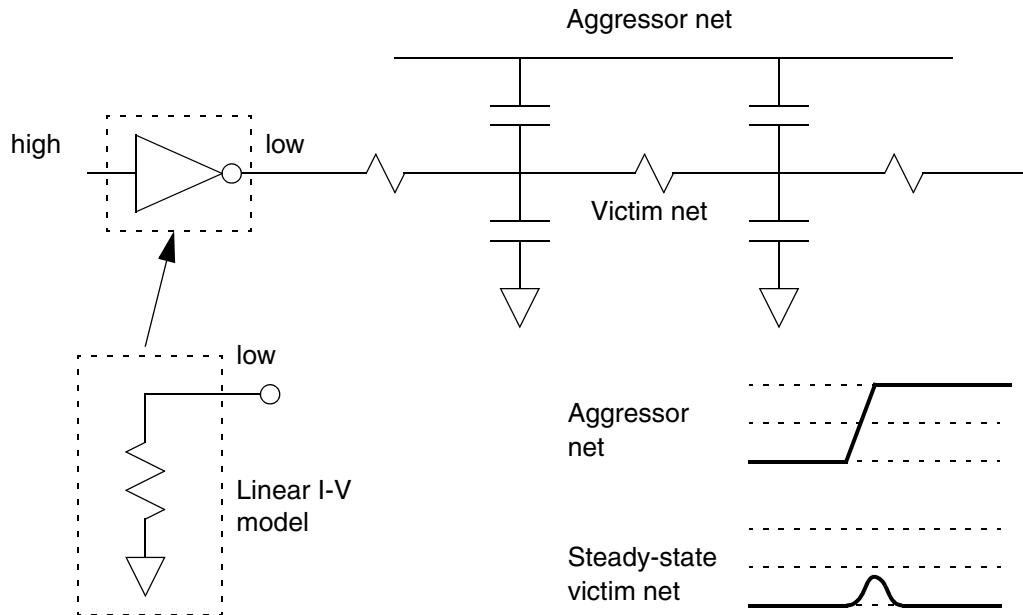
To learn more, see

- Synopsys Library Compiler documentation
- [Steady-State I-V Characteristics](#)
- [Noise Immunity](#)
- [Propagated Noise Characteristics](#)

Steady-State I-V Characteristics

A steady-state driver of a net affects the size of crosstalk bumps on the net due to its loading effects on the net. The tool needs the steady-state I-V characteristics of the driver output to accurately calculate the characteristics of crosstalk noise bumps. For example, consider the crosstalk noise analysis in [Figure 14-40](#). The driver of the victim net is steady at logic 0. When a rising transition occurs on the aggressor net, it causes an above-low noise bump on the victim net. The steady-state I-V characteristics of the driver affect the size of the size bump. The simplest model that can be used is a linear I-V curve, which is the same as a resistor, like the model shown in the diagram.

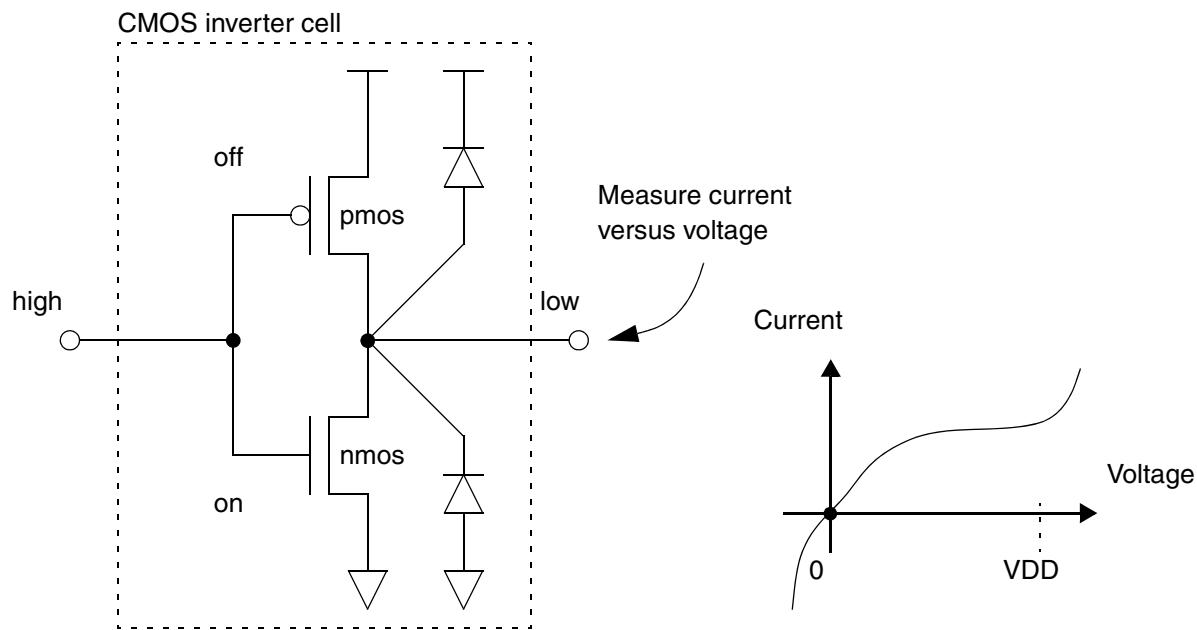
Figure 14-40 Linear I-V Model for a Cell With Steady-State Low Output



The I-V characteristics of a cell can be measured in the laboratory by placing the cell output into either the low or high state, and then measuring the current at different voltages at the output, as shown for the CMOS inverter in [Figure 14-41](#). The circuit diagram includes the diodes at the output that exist because of the p-n isolation junctions of the pulldown and pullup transistors. These diodes affect the I-V characteristics at voltages below zero and above VDD.

A typical CMOS output at logic 0 has a steady-state operating point at (0.0, 0.0). In the small region surrounding this operating point, the output behaves like a resistor and the I-V plot looks like a straight line. However, at larger positive voltages, the I-V plot becomes nonlinear due to the NMOS transistor shutting off and because of forward-biasing of the PMOS junction diode. At voltages well below zero, the plot become nonlinear because of the forward-biasing of the NMOS junction diode.

Figure 14-41 I-V Characterization of a Steady-State Low Output

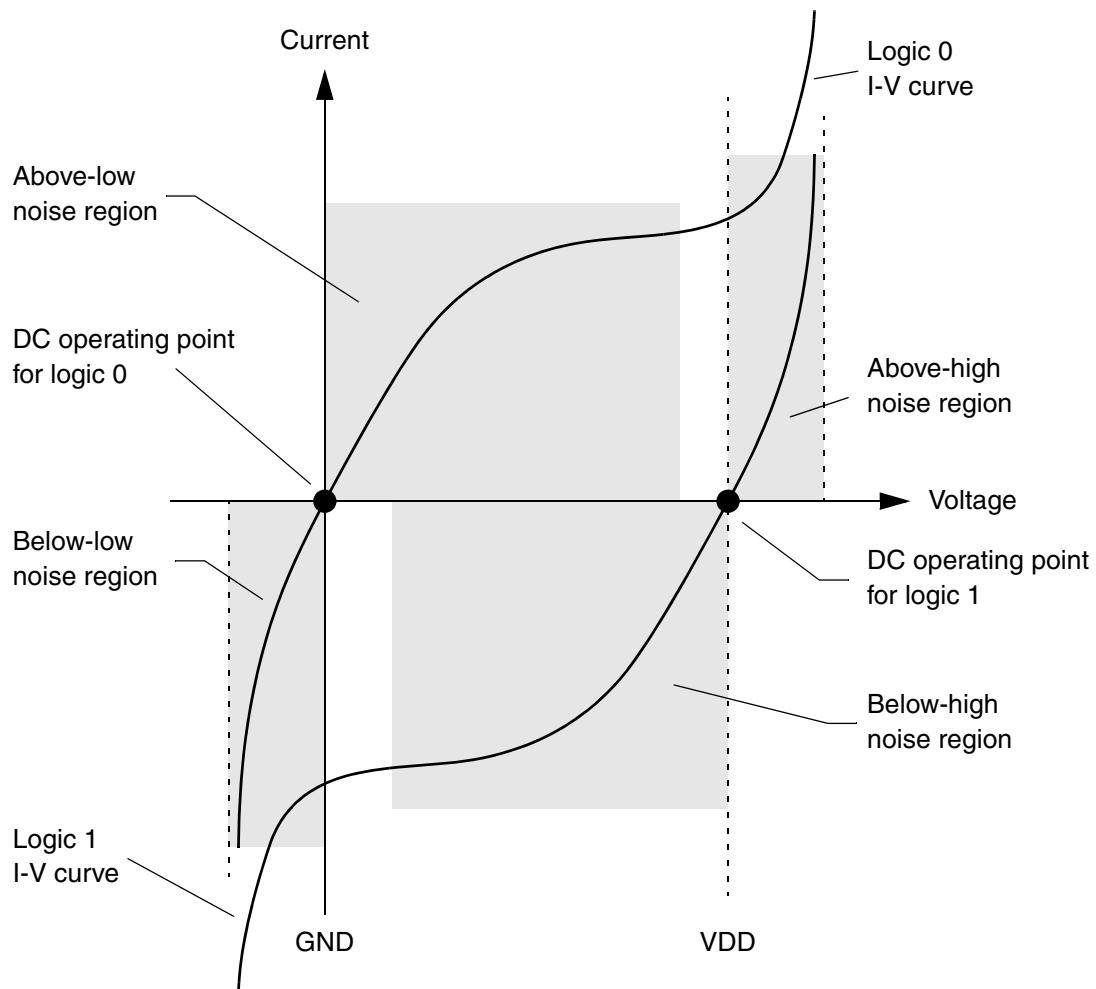


A similar I-V plot can be obtained by placing the CMOS output at logic 1. Its steady-state operating point is at (VDD, 0.0).

[Figure 14-42](#) shows a typical plot of both the logic 0 and logic 1 I-V curves on the same graph. For a process technology that is symmetrical between NMOS and PMOS transistor characteristics, the logic-one I-V curve is the same as the logic-zero curve rotated 180 degrees and shifted to the right by the amount VDD.

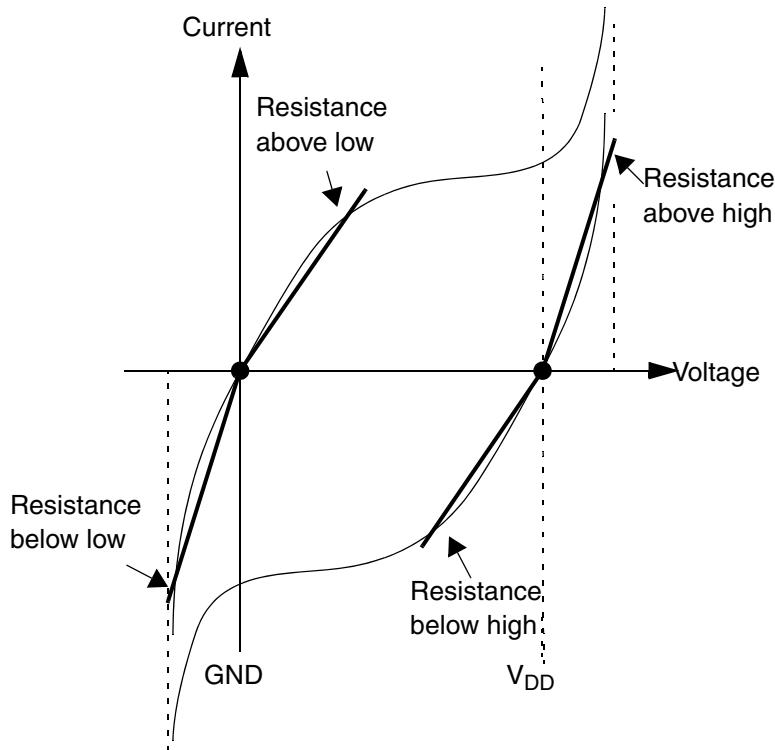
The four shaded portions indicate the operating regions when noise bumps occur: below low, above low, below high, and above high. The library syntax allows the I-V characteristics to be specified as two resistors (one each for above low and below low), as a piecewise linear model, or as a polynomial function.

Figure 14-42 Steady-State I-V Characteristics at a CMOS Output



For example, to approximate the I-V curve using two resistors, you could draw the two lines shown in [Figure 14-43](#). Each resistance value is the inverse of the slope of the line (voltage divided by current). You can enter the four steady-state resistance values into the cell library description or specify them with the `set_steady_state_resistance` command.

Figure 14-43 Resistance Approximation of Steady-State I-V Characteristics



For even better accuracy, you can specify the I-V characteristics using a piecewise linear model or a polynomial model. These more accurate models can be specified only in the library, not with PrimeTime SI commands.

In the absence of library-specified or command-specified I-V characteristics, the tool uses an estimated linear resistance calculated from the output delay, output slew, and NMOS and PMOS transistor threshold voltages. The output delay and slew are specified in the library. The tool assumes an NMOS and PMOS threshold voltage of 0.2 times the rail-to-rail voltage.

In case of conflict between different methods, the tool uses steady-state I-V specifications in the following order:

- PrimeTime SI command-specified steady-state resistance (the `set_steady_state_resistance` command)
- Library-specified per-arc I-V polynomials or tables
- Library-specified per-pin steady-state resistance
- PrimeTime SI estimation from output delay, output slew, and NMOS and PMOS transistor threshold voltages

Table 14-6 lists and briefly describes the methods that can be used to specify cell output steady-state I-V characteristics in the Synopsys .lib logic library. For more information about library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 14-6 Library Specification Methods for Output I-V Characteristics

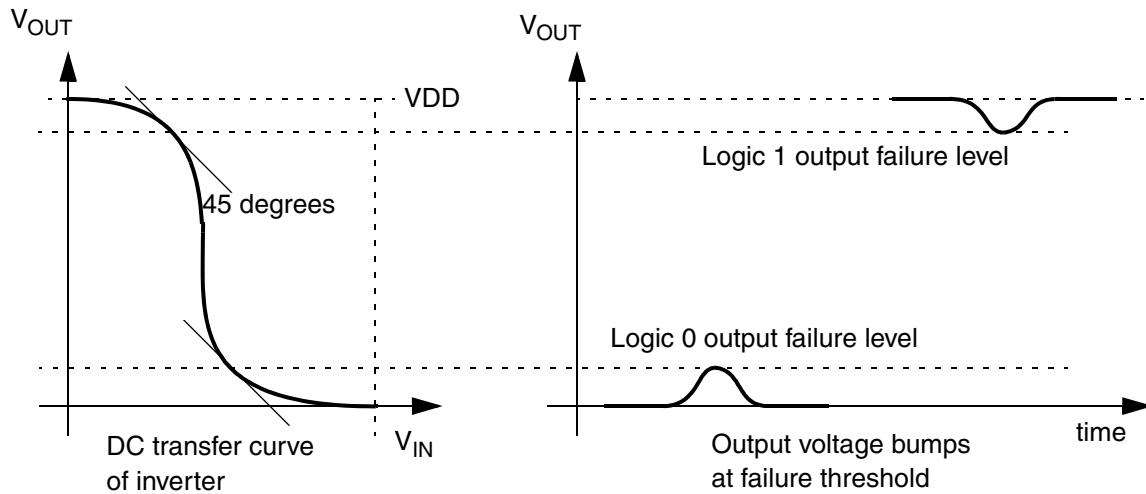
Specification method	Library type	How specified in library
Polynomials	SPDM	Two polynomials per timing arc for steady-state low and steady-state high. Polynomials describe current as a function of voltage.
Lookup tables	NLDM	Two lookup tables per timing arc for steady-state low and steady-state high. Tables describe current as a function of voltage.
Steady-state resistance	NLDM or SPDM	Four floating-point resistance values per timing arc for above low, below low, above high, and below high. Output is modeled as a resistor connected to ground for logic 0 or connected to VDD for logic 1. Can also be specified per output with the <code>set_steady_state_resistance</code> command.

Noise Immunity

Each cell input can tolerate a certain amount of noise without causing a failure at the cell output. This characteristic is called *noise immunity*. The tool uses the library-specified or command-specified noise immunity at cell inputs to determine whether noise failures occur and the amount of noise slack at each cell input.

The recommended definition of “logic failure” is established by the points in the DC transfer curve at which the slope of the curve reaches 45 degrees. This definition is shown for the case of an inverter in [Figure 14-44](#). However, the creator of the library might use some other failure criteria.

Figure 14-44 Noise Immunity Failure Definition



Noise immunity can be specified either in terms of allowable noise bump heights and widths at the cell inputs (specified as noise immunity curves, polynomials, or tables) or in terms of noise margins that consider only the bump heights at the cell inputs.

In case of conflict between different methods, the tool uses noise immunity specifications in the following order:

- PrimeTime SI command-specified noise immunity curves (the `set_noise_immunity_curve` command).
- PrimeTime SI command-specified bump height noise margins (the `set_noise_margin` command)
- Library-specified per-arc noise immunity polynomials or tables
- Library-specified per-pin noise immunity curves
- Library-specified DC noise margins (V_{OL} , V_{OH} , V_{IL} , V_{IH})

Table 14-7 lists and briefly describes the methods that can be used to specify cell noise immunity characteristics in the Synopsys .lib logic library. For more information about library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 14-7 Library Specification Methods for Noise Immunity

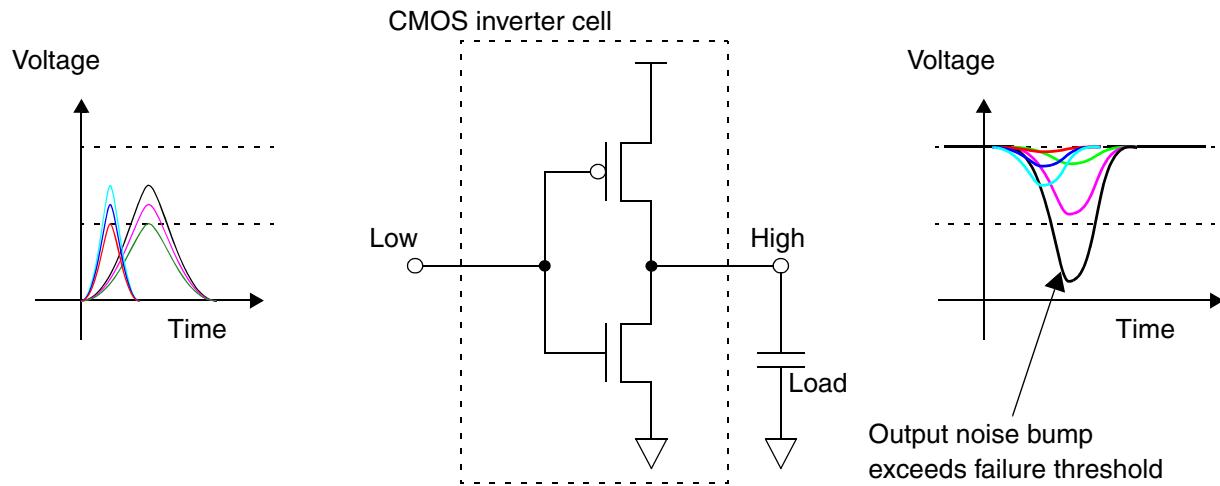
Specification method	Library type	How specified in library
Hyperbolic noise immunity curves (per input)	NLDM or SPDM	Four hyperbolas per input specify the maximum noise bump height as a function of noise bump width: one hyperbola each for input noise bumps above low, below low, above high, and below high. Each hyperbola is specified with three floating-point coefficients. Can also be specified with the <code>set_noise_immunity_curve</code> command.
Noise immunity polynomials (per timing arc)	SPDM	Four polynomials per timing arc specify maximum noise height as a function of noise width and output load: one polynomial each for input noise bumps above low, below low, above high, and below high.
Noise immunity lookup tables (per timing arc)	NLDM	Four lookup tables per timing arc specify maximum noise height as a function of noise width and output load: one table each for input noise bumps above low, below low, above high, and below high.
Noise margins based on bump height only (per input)	NLDM or SPDM	Four floating-point values specify the maximum and minimum allowable voltages for logic 0 and logic 1 for each input pin. Can also be specified with the <code>set_noise_margin</code> command.

Noise Immunity Curves

When you use a noise immunity curve, the tool considers the width and height of noise bumps occurring at cell inputs. Because of the capacitance at the cell input, a very brief noise bump can be tolerated, even if it is relatively high.

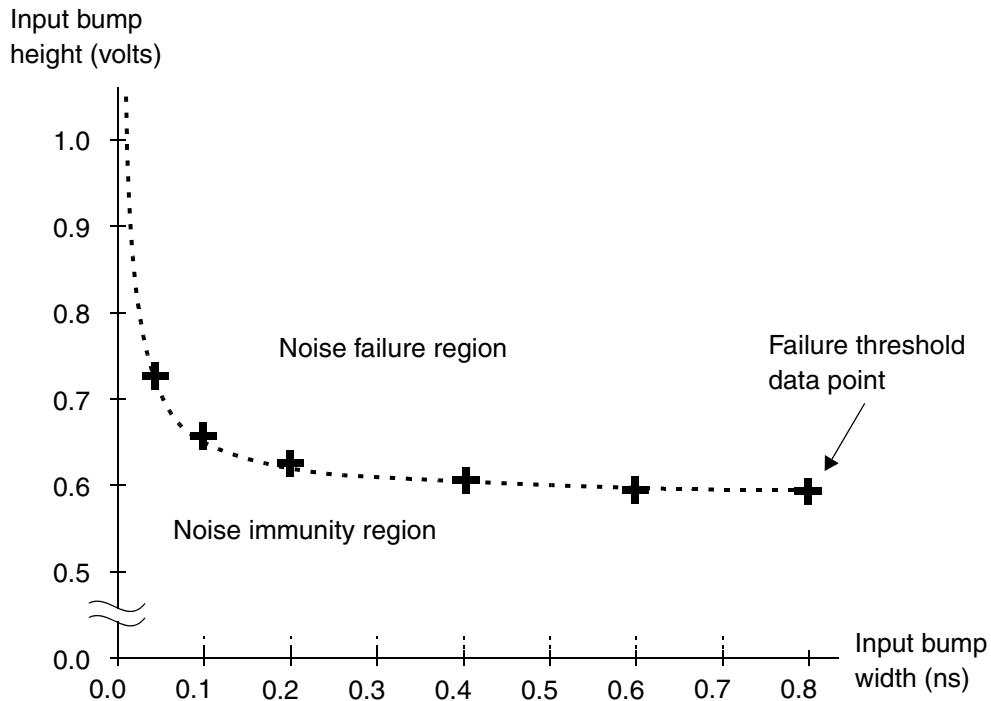
The noise immunity of a cell input can be measured in the laboratory by applying noise bumps of varying heights and widths to the input, as shown for the CMOS inverter in **Figure 14-45**. Bumps at the input that are small in terms of width and height does not cause any change at the output. However, bumps that are wide and high causes the output to have a bump or even change logic state entirely.

Figure 14-45 Noise Immunity Characterization of Input



If the cell has multiple outputs, there can be multiple timing arcs with different immunity characteristics for the same input. For each input, the values obtained from the worst-case input-to-output timing arc should be used in the library. Because noise immunity is sensitive to output load, characterization should be done with the worst-case (smallest) capacitive load.

Under the worst-case conditions, if you select several combinations of height and width at which logic failures just begin to appear and plot them on a graph, you get a curve similar to the one shown in [Figure 14-46](#). Given this information for each cell input and the size of the input noise bump, the tool can determine whether a logic failure occurs at the cell output.

Figure 14-46 Input Bump Width Versus Height at Failure Thresholds

The library syntax allows a noise immunity curve to be specified as a hyperbolic curve, a piecewise linear model, or a polynomial. For a hyperbolic curve, three coefficients fully specify the hyperbola:

$$y = c_1 + \frac{c_2}{(x - c_3)}$$

where

y = height

x = width of the input voltage bump at the threshold of logic failure

c_1 = voltage offset equal to the DC noise immunity value

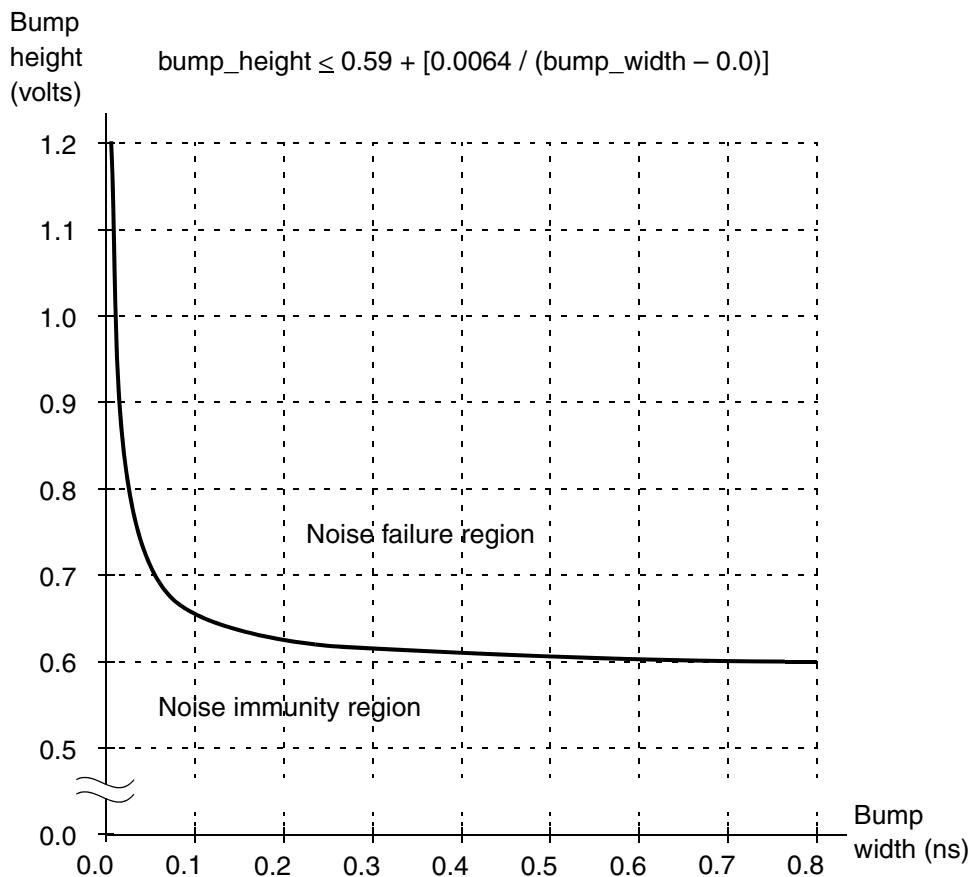
c_2 = size parameter for the hyperbolic curve

c_3 = time-value offset

The three coefficients should be chosen to match the hyperbolic curve to the data points obtained by laboratory characterization.

For example, [Figure 14-47](#) shows a plot of a noise immunity curve with $c_1 = 0.59$, $c_2 = 0.0064$, and $c_3 = 0.0$. Combinations of bump height and width below the curve are in the region of noise immunity, while those above the curve are in the region of noise failure.

Figure 14-47 Hyperbolic Noise Immunity Curve



In Library Compiler syntax, the coefficients c_1 , c_2 , and c_3 are called `height_coefficient`, `area_coefficient`, and `width_coefficient`.

Noise immunity characteristics can vary for different noise bump types, so there can be four different noise immunity curves associated with each input: below low, above low, below high, and above high. All coefficients are specified as positive numbers for all four types of noise bumps.

In the absence of library-specified noise immunity characteristics, or to override the library-specified characteristics, you can use the `set_noise_immunity_curve` command, which lets you set the three hyperbolic coefficients for specified ports or cell input pins. The coefficients c_1 , c_2 , and c_3 are set with the options `-height`, `-area`, and `-width`.

You can display noise immunity curves in the PrimeTime GUI. For more information, see [Examining Noise Immunity Curves](#).

Noise Immunity Polynomials and Tables

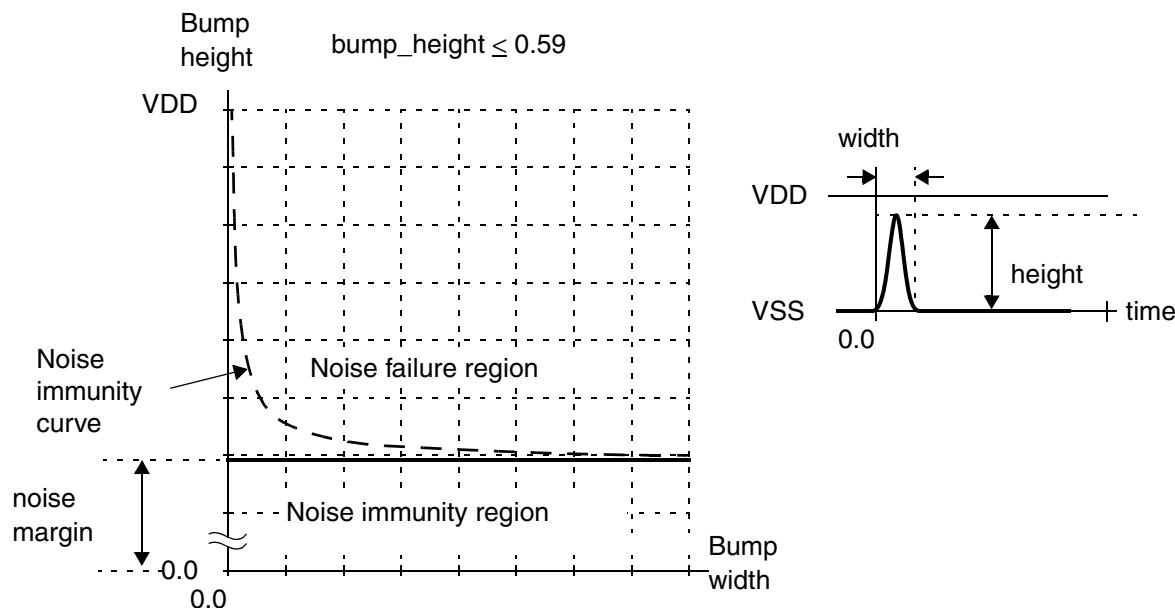
In cases where the noise immunity characteristics vary significantly due to different output loads or different paths through the cell, the library can use polynomials or lookup tables instead of hyperbolic noise immunity curves. Using polynomials or lookup tables, the library specifies the maximum input bump height as a function of input bump width and output capacitive load. Each noise immunity specification applies to an individual input-to-output timing arc rather than all arcs through a given input.

Per-arc specification allows for state-dependent variation in noise immunity. For example, for an XOR gate with inputs A and B and output Z, the A-to-Z and B-to-Z arcs might have different noise immunity due to the different paths taken through the transistors in the cell.

Bump Height Noise Margins

Instead of using noise immunity specifications that consider input bump width, input bump height, and output load, you can use noise margins that consider only the input bump height. Using height-only noise margins is simpler, faster, and more conservative than the other methods. [Figure 14-48](#) compares noise immunity curves and bump height noise margins.

Figure 14-48 Height-only Noise Margin Versus Noise Immunity Curve



For high, narrow noise bumps, using height-only noise margins is pessimistic because it treats some bumps as noise failures that would otherwise pass with the immunity curve model. However, for wide noise bumps, using noise margins gives the same results as using noise immunity curves.

There are four different noise margin values associated with each input: below low, above low, below high, and above high. These values are specified as positive numbers for all four types of noise bumps.

In the absence of library-specified noise immunity specifications, or to override the library-specified specifications, you can use the `set_noise_margin` command to set the height-only noise margins for specified ports or cell input pins.

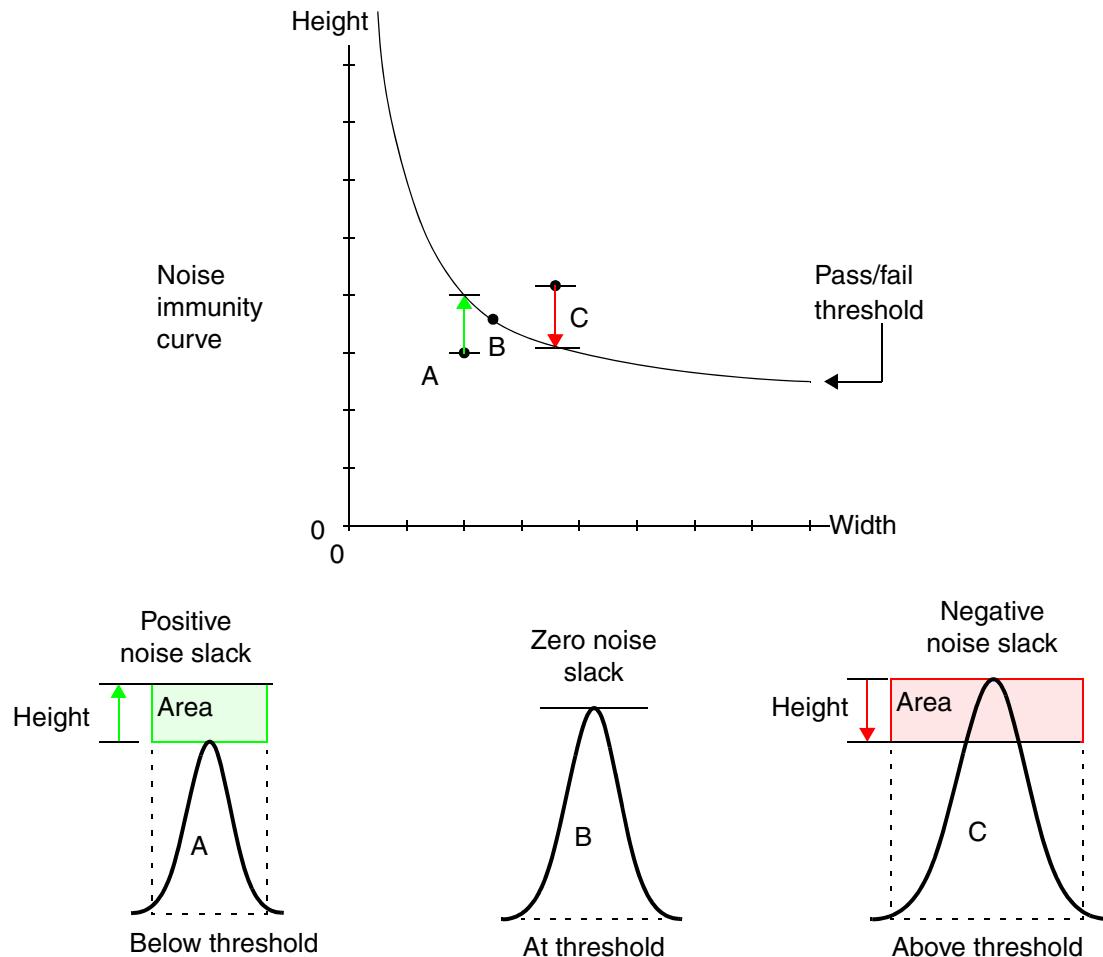
In the absence of command-specified or library-specified noise immunity data, the tool calculates the maximum allowable noise bump heights based on DC noise margins of the driver and receiver, as defined in the .lib logic library by the input/output logic-level parameters V_{IL} , V_{IH} , V_{OL} , and V_{OH} .

Noise Slack

In static timing analysis, *slack* is the amount of time by which a timing constraint is met. It is the difference between the required time and the arrival time of a signal transition. It is in library units of time, such as nanoseconds. Negative slack indicates a timing failure.

In static noise analysis, you can choose the following noise slack reporting methods: “area”, “height”, and “area_percent” methods. The default method is “height”, where noise slack is determined by the amount of voltage by which a noise constraint is met, given that the noise bump width remains unchanged.

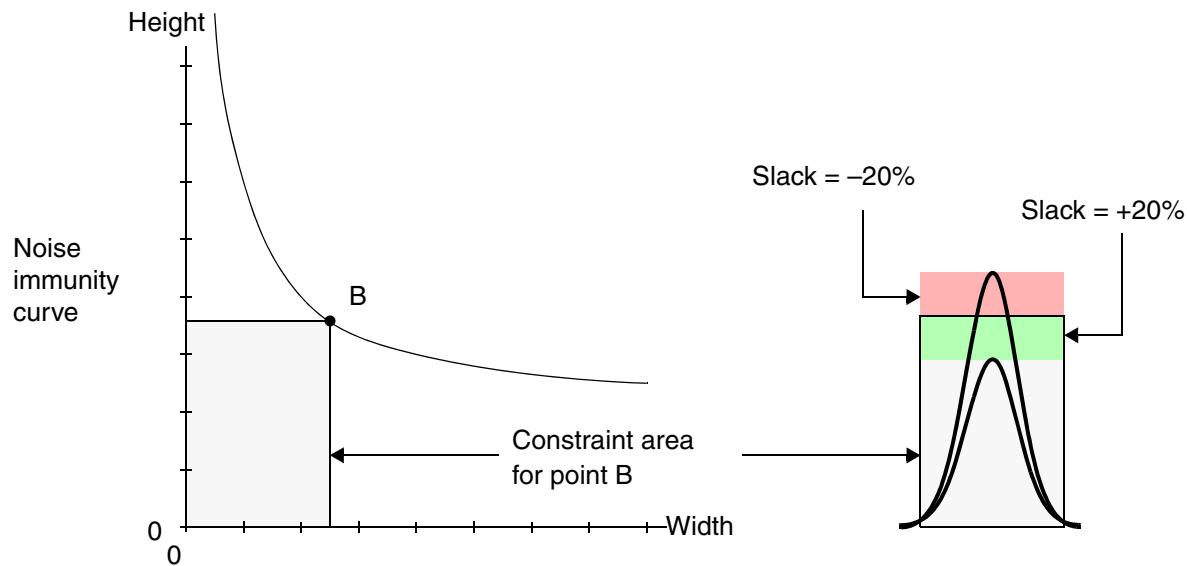
The calculation of noise slack is shown in [Figure 14-49](#). The figure shows three noise bumps: one below the threshold of noise failure, one just at the threshold of failure, and one above the threshold of failure. The width and height of each noise bump is plotted as a black dot on the noise immunity curve for the cell input.

Figure 14-49 Noise Slack Calculation

The “area” type noise slack is the voltage margin (height of the curve above the data point) multiplied by the noise bump width, as indicated by the shaded rectangles in [Figure 14-49](#). For a noise bump below the failure threshold, the slack is positive, or for a noise bump above the failure threshold, the slack is negative. The units for “area” noise slack are library units of voltage multiplied by library units of time, such as millivolt-nanoseconds.

Using the height method, the noise slack is defined as the voltage margin, in library voltage units, for a given noise bump height. For noise immunity specified by immunity curves, tables, or polynomials, different input noise bump widths yield different slack height values. The slack height is a more direct representation of the noise violation on a pin.

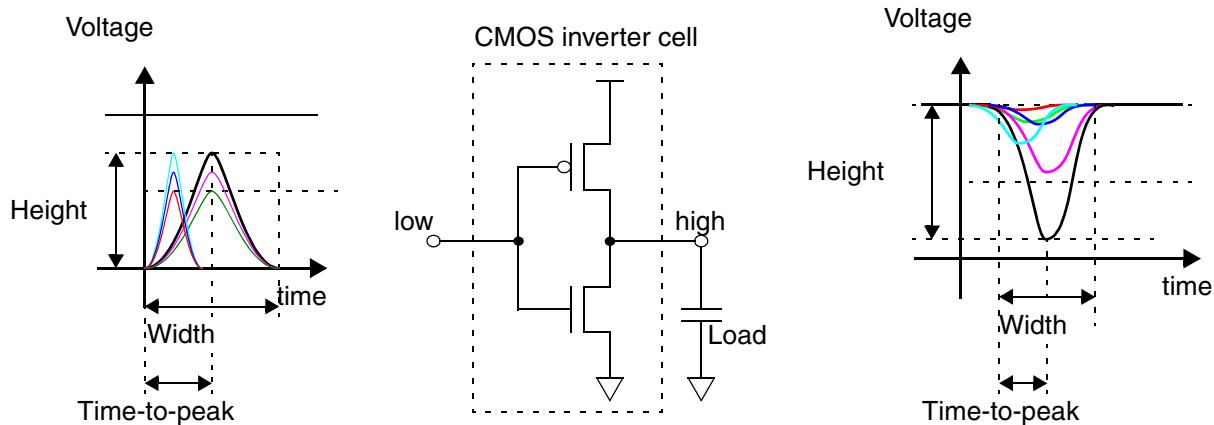
Using the “area_percent” method, the noise slack is defined as the area slack divided by the total “constraint area.” The constraint area is the bump width multiplied by the allowed height, equal to the area of the rectangle bounded by the data point in the noise immunity curve, as shown in [Figure 14-50](#).

Figure 14-50 Area Percent Slack Calculation

Propagated Noise Characteristics

A noise bump at a cell input, if large enough in terms of height and width, causes a noise bump at the cell output. This effect is called noise propagation.

The noise propagation for an input-to-output timing arc can be measured in the laboratory by applying noise bumps of varying heights and widths to the input and measuring the resulting noise bumps at the output, as shown for the CMOS inverter in [Figure 14-51](#). The output noise bump characteristics depend on the width and height of the input bump and the capacitive load on the output, and to a lesser extent, the time-peak-ratio of the input bump.

Figure 14-51 Noise Propagation Characterization

After the noise propagation effects have been characterized, the information can be entered into the library. The library syntax supports two ways to specify propagation effects: polynomials and lookup tables.

With polynomials, the library specifies the height, width, and time-peak-ratio of the output bump as a function of the input bump height, input bump width, input bump time-peak-ratio, and output load. There are 12 such polynomials for each timing arc because there are three functions (height, width, and time-peak-ratio) for each of four output bump types: below low, above low, below high, and above high.

The time-peak-ratio is the time-to-peak value divided by the width of the noise bump. The time-peak-ratio can be floating-point value between 0.0 and 1.0. For a symmetrical noise bump, the time-peak-ratio is 0.5. Including time-peak-ratio characteristics makes a more accurate noise propagation model, but requires more work for characterization.

With lookup tables, the library specifies the height and width of the output bump as a function of the input bump height, input bump width, and output load. There are eight such lookup tables for each timing arc because there are two functions (height and width) for each of four output bump types: below low, above low, below high, and above high. The lookup tables do not include time-peak-ratio information, but the tool still calculates the time-peak-ratio characteristics of propagated noise bumps based on the cell delay and slew information in the library.

[Table 14-8](#) describes the methods that can be used to specify propagated noise characteristics in the Synopsys .lib logic library. For more information about library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 14-8 Library Specification Methods for Propagated Noise

Specification method	Library type	How specified in library
Polynomials	SPDM	Four sets of three polynomials (12 polynomials) per timing arc that specify the output bump height, output bump width, and output peak-time-ratio as a function of input bump height, input bump width, input peak-time-ratio, and output load; one set of three polynomials each for output noise bumps above low, below low, above high, and below high.
Lookup tables	NLDM	Four pairs of lookup tables (eight tables) per timing arc that specify the output bump height and output bump width as a function of input bump height, input bump width, and output load; one pair of tables each for output noise bumps above low, below low, above high, and below high.

15

Advanced Analysis Techniques

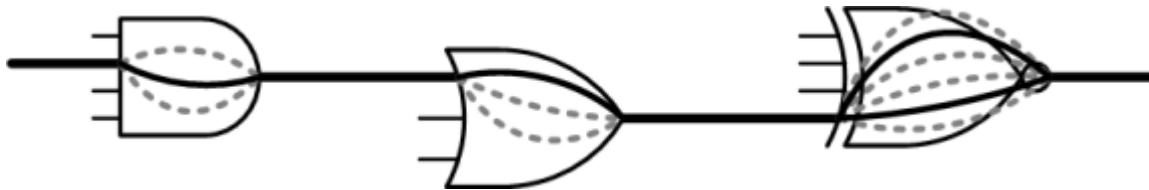
To learn about advanced timing analysis techniques that you can use in PrimeTime, see

- [Parallel Arc Path Tracing](#)
- [Support for Retain Arcs](#)
- [Asynchronous Logic Analysis](#)
- [Three-State Bus Analysis](#)
- [Data-to-Data Checking](#)
- [Interdependent Setup and Hold Pessimism Reduction](#)
- [Time Borrowing in Latch-Based Designs](#)
- [PrimeTime ADV Advanced Latch Analysis](#)

Parallel Arc Path Tracing

To ensure that the `report_timing` command uses only the worst arc in each sense set of parallel arcs for path tracing, set the `timing_report_use_worst_parallel_cell_arc` variable to `true`. This results in the path tracing shown in [Figure 15-1](#).

Figure 15-1 Worst Arc Used for Path Tracing



Only paths with the worst arc behaviors through the logic with a given rise and fall edge sequence are returned. The worst arc is the fastest arc for minimum delay tracing and it is the slowest arc for maximum delay tracing. Subcritical timing paths through other combinations of timing arcs with the same edge direction sequence are not returned. However, since timing arcs with different senses are still considered unique, different paths with unique rise and fall edge sequences through nonunate gates are still returned.

This might cut down substantially on the number of paths returned by large `-nworst` values, improving analysis efficiency. If the `-nworst` limit is not being reached, such as when reporting paths to a single endpoint, a lesser number of timing paths are returned. If `-nworst` has reached its limit, such as when reporting across an entire path group, the paths that are returned up to the limit might have a more varied topological exploration of the logic. This can help improve the efficiency of reporting scripts, bottleneck analysis, ECO scripts that are driven by `timing_path` collections.

This feature affects both the `report_timing` and `get_timing_paths` commands when you specify an `-nworst` value greater than 1. Only parallel arcs of the same sense are affected. If there are different sense arcs in parallel, such as `positive_unate` and `negative_unate` across an XOR gate, they are still considered unique. This feature affects path tracing behavior at reporting time only. It does not affect the behavior of the `update_timing` command. You can change the value of the variable at any time without incurring a timing update penalty.

Support for Retain Arcs

PrimeTime can load retain arcs for timing models from library files, annotate the retain arcs from SDF input files, and report these arcs. Retain arcs are similar to hold-check arcs and are typically used for modeling random access memory (RAM). They are defined between a clock pin and the data output of a RAM, and they are always defined in parallel with the

parent arc, which is the ordinary or default delay arc between the same two pins. A retain arc does not generate an actual timing check during timing analysis, but is treated as another delay arc that is connected in parallel with its parent arc.

Clock-to-output retain arcs guarantee that the RAM output does not change for a specific interval of time after the clock edge. When the retain arc delay is less than its parent arc, the retain arc appears in a timing report for only the minimum delay paths. When the retain arc delay is longer than its parent arc, the retain arc can also appear in the maximum delay path report with no error messages or warnings.

To report all of the timing arcs for cells in a logic library, including retain arcs, use the `-timing_arcs` option with the `report_lib` command. Use the `check_timing retain` command to check if the retain arc has a delay greater than its parent arc.

The `read_sdf` command supports retain arcs, but the default behavior of the `write_sdf` command does not support those arcs. To write out retain arcs, use SDF version 3.0 format, not the default version 2.1 format, by specifying the following syntax:

```
pt_shell> write_sdf -version 3.0 file
```

To map retaining information for arcs, use these functions:

- `min_rise_retain_delay`
- `min_fall_retain_delay`
- `max_rise_retain_delay`
- `max_fall_retain_delay`

Asynchronous Logic Analysis

To simplify the timing verification designs with asynchronous logic, isolate the asynchronous logic into separate blocks, then disable the timing of these blocks.

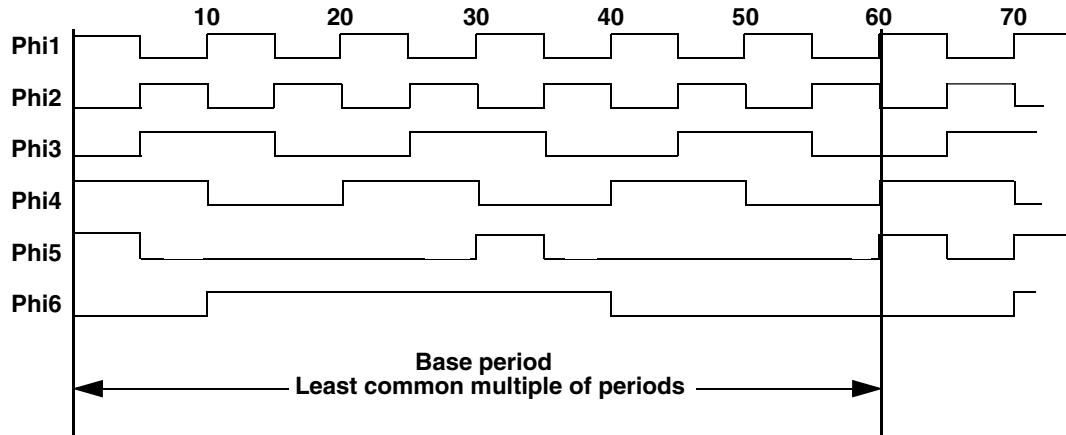
PrimeTime does not support self-timed asynchronous logic where no global clock is used. Isolate this type of logic in a level of hierarchy and then use full-timing gate-level simulation to verify valid timing and functional capability.

PrimeTime can analyze designs:

- With no combinational feedback loops; loops containing flip-flops or latches are adequate (combinational feedback loops are automatically broken)
- Without unclocked memory elements, such as RS latches
- With a single clock or multiple clocks fanning in to each register clock pin

- With known and fixed phase relationship between the clocks at the start and end registers of every path (Interacting clocks must have a single base period over which all clock waveforms repeat—see [Figure 15-2](#).)

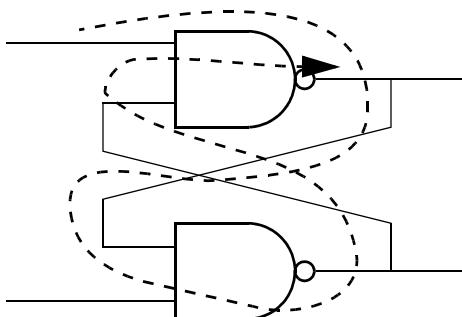
Figure 15-2 Base Period of Clocks



Combinational Feedback Loop Breaking

A combinational feedback loop is a path that can be traced through combinational logic back to its starting point. [Figure 15-3](#) shows an example. To analyze such a path, PrimeTime must break the loop (stop tracing the path) at some point within the loop. Check a design for the presence of combinational feedback loops with the `check_timing -include_loops` command.

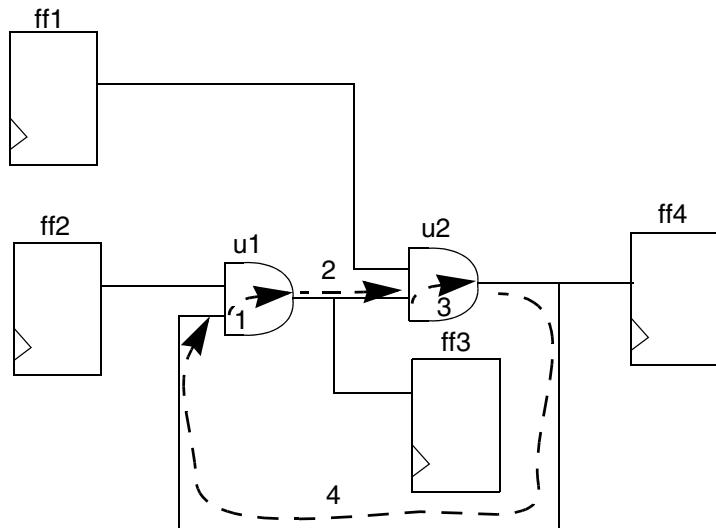
Figure 15-3 Combinational Feedback Loop



By default, PrimeTime identifies each feedback loop and disables one of the timing arcs of the loop, such as the timing arc from one input to one output of a NAND gate in the loop. In some cases, this approach can result in some real paths not being reported because the paths are broken by the disabled arcs.

Figure 15-4 shows an example where no timing arc of the combinational loop can be broken without a valid path of the design also being broken. Arcs #1 and #4 cannot be broken because breaking them would break the valid path ff1 – u2 – u1 – ff3. Arcs #2 and #3 cannot be broken because breaking them would break the valid path ff2 – u1 – u2 – ff4.

Figure 15-4 Loop Breaking Example



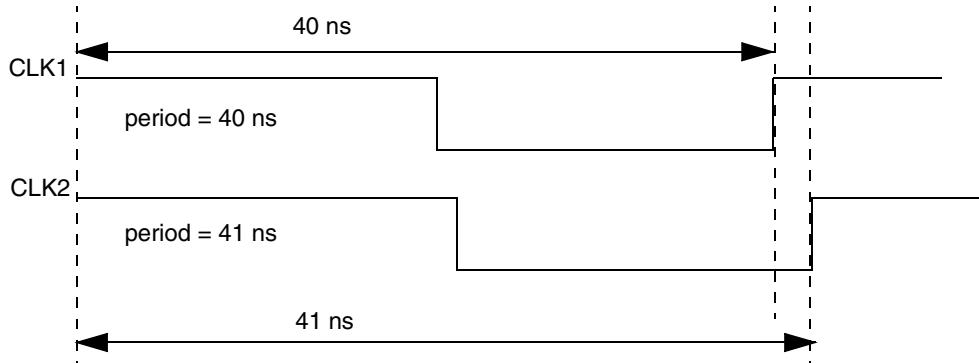
There are multiple places in a feedback loop that could be broken. If you want a feedback loop to be broken at a different timing arc from the one selected by default, use the `set_disable_timing` command to explicitly break the loop at the desired point.

Design Compiler also uses loop breaking to analyze timing. The points at which Design Compiler and PrimeTime break a loop can be different, possibly leading to different timing results. If you want to ensure consistent loop breaking between the two tools, set the `timing_keep_loop_breaking_disabled_arcs` variable to `true` in PrimeTime. In that case, PrimeTime inherits the loop-breaking choices from the .ddc or .db file generated by Design Compiler. By default, this variable is set to `false`. After changing this variable setting, a timing update is necessary to see the change.

The report generated by `report_disable_timing` distinguishes between arcs disabled by PrimeTime and those disabled by inheritance from the .ddc or .db file.

Unrelated Clocks

Sometimes a design has paths between unrelated clocks. Unrelated clocks have different frequencies that do not have a reasonable base period. PrimeTime attempts to find a base period and phase relationship anyway, which typically is not useful (see [Figure 15-5](#)).

Figure 15-5 Unrelated Clock Waveforms

The clocks shown in [Figure 15-5](#) do not expand to a reasonable base period, and the resulting setup requirement is quite restrictive. In this case, you might want to verify the timing with dynamic simulation rather than use PrimeTime.

To exclude asynchronous paths from static timing analysis to improve runtimes and avoid false violations, enter the following command:

```
pt_shell> set_false_path -from [get_clocks clk40] \
           -to [get_clocks clk41]
```

In some cases, you might know the required minimum and maximum path delays for the combinational logic between two registers of unrelated clocks. In these cases, specify the path delay constraint using the `set_max_delay` and `set_min_delay` commands for that path.

Three-State Bus Analysis

By default, PrimeTime checks for setup and hold violations in three-state bus designs. It checks the worst delay path to ensure that the latched signals are stable and are not unknown (X) values. This ensures proper timing checks for transient bus contention and transient floating bus conditions.

PrimeTime considers that a disabling transition on a three-state cell can cause either a 1 or 0 delay on the output. By default, PrimeTime considers `three_state_disable` and `three_state_enable` arcs (as defined in the library) during path tracing. Although this is different from propagating an X value, the effect for static timing is the same as for propagating an X value.

Limitations of the Checks

The three-state bus checks have these limitations:

- PrimeTime checks the potential for setup or hold errors due to bus contention or float conditions. PrimeTime does not check logical and power violations for bus contention or float conditions.
 - The analysis is pessimistic; some reported violations might never happen because of state dependencies.
 - In some simulators, Z does not propagate to X until after the charge decay time. PrimeTime does not model this effect; it uses the gate delay equations to propagate Z and X. This might be pessimistic compared to some simulators.
-

Disabling the Checks

If you know that bus contention or floating buses do not occur in your design, disable these checks by setting these variables to `true`:

`timing_disable_bus_contention_check`

When you set this variable to `true`, propagation of maximum delay along `three_state_disable` timing arcs and minimum delay along `three_state_enable` arcs is disabled. These checks are valid only during transient bus contention. The default is `false`.

`timing_disable_floating_bus_check`

When you set this variable to `true`, propagation of minimum delay along `three_state_disable` timing arcs and maximum delay along `three_state_enable` arcs is disabled. These checks are valid only during floating bus conditions. The default is `false`.

Bus Contention

Some designs rely on a bus configuration in which many three-state drivers control the bus. In most designs, no two drivers with different logical outputs can be simultaneously enabled at the steady state (when the enable pins of the three-state drivers assume their steady-state values for any clock cycle). When multiple drivers drive the same bus, it is called bus contention.

Although you can design a circuit so that no steady-state bus contention occurs, a design might contain transient bus contention conditions. Transient bus contention conditions occur during the transition of the bus control from one driver to another. During this short transient period, the logical value for the bus is unknown (X value) if the drivers contending for control

of the bus are imposing conflicting logical values. For static timing analysis, setup checks ensure that this X value, assumed during transient bus contention, is not latched. The setup check is measured from the time the bus becomes stable.

Floating Buses

A floating bus condition can arise for three-state bus configuration designs. A floating bus condition occurs when no driver is enabled. In this case, the bus immediately gets an unknown (X) value. For static timing analysis, hold checks ensure that this X value, assumed when the bus switches to the floating mode, is not latched. The hold checks are measured to the time the bus becomes floating.

Three-State Buffers

Two timing arc types are used to describe timing behavior of three-state buffers. These timing arc types are defined in the library.

`three_state_disable` timing arc

Specifies the time the three-state pin takes to go from a high or low state to the high-impedance state.

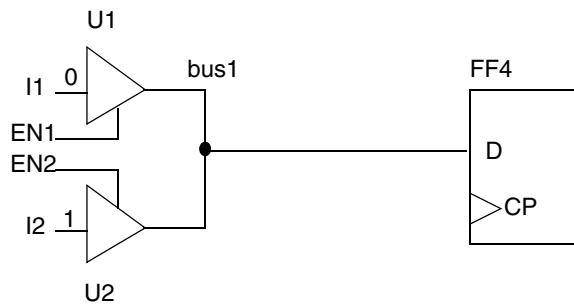
`three_state_enable` timing arc

Specifies the time a three-state pin takes to go from the high-impedance state to a high state or low state.

Performing Transient Bus Contention Checks

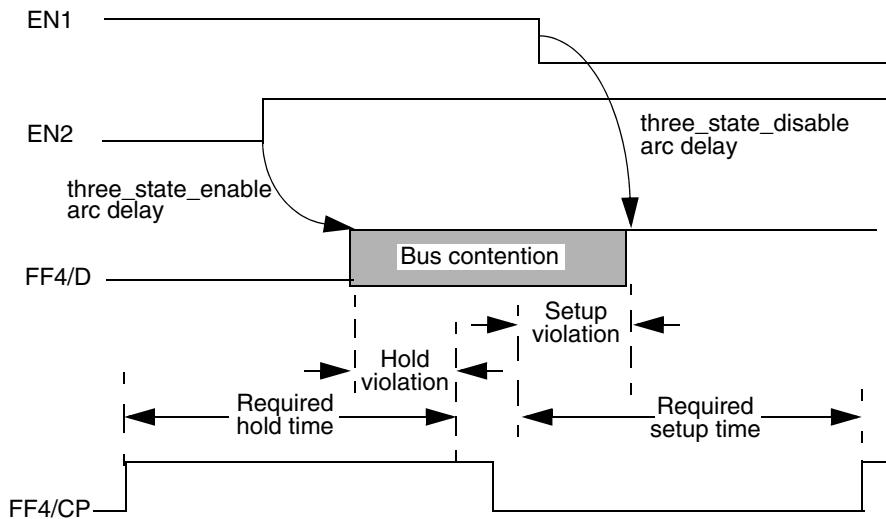
[Figure 15-6](#) shows a circuit used to describe how PrimeTime performs transient bus contention checks and floating bus checks.

Figure 15-6 Circuit Example



[Figure 15-7](#) shows how PrimeTime performs transient bus contention checks.

Figure 15-7 Transient Bus Contention Check



In Figure 15-6, if EN1 turns off after EN2 turns on, there is a potential bus contention for some time. The signal arriving at FF4/D is X state until the contention is over and the new bus value propagates along the path. If I1=0 and I2=1, the waveforms for the circuit in Figure 15-6 are as shown in Figure 15-7.

The setup violation for FF4/CP is seen only if the three_state_disable arc delay is considered. A transition to the Z state must be propagated as a possible transition to logic 0 or logic 1 to find all possible cases. A similar situation occurs for the hold check. The hold violation is seen only if the three_state_enable arc delay is considered.

This bus contention region is bounded by the minimum three_state_enable arc delay of any bus driver from one side and by the maximum three_state_disable arc delay from the other side. If you know that such bus contention regions can never occur, disable checking for both setup and hold violations that occur due to this bus contention region.

Disable the checks by setting the `timing_disable_bus_contention_check` variable to `true`, causing PrimeTime to ignore the three_state_enable arc delay for hold violation checking and the three_state_disable arc delay for setup violation checking.

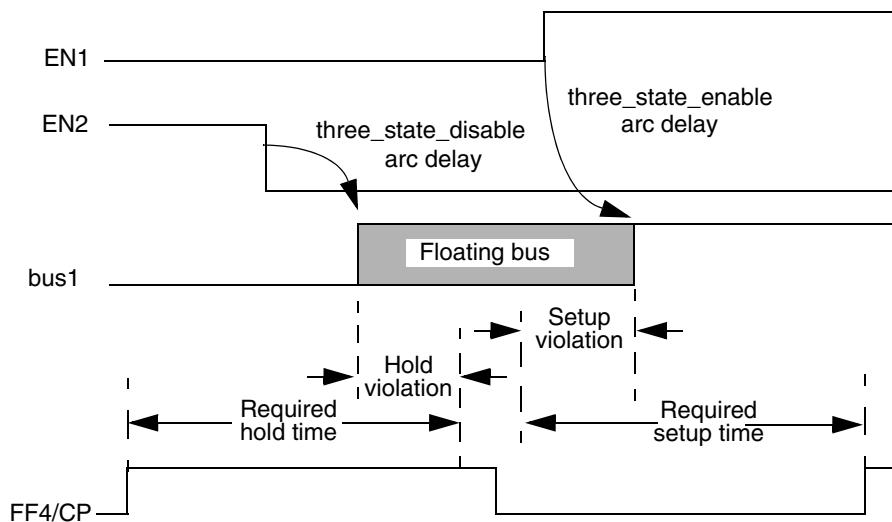
Even when you set this variable to `true`, PrimeTime considers the three_state_enable arc delay for setup violation checking and the three_state_disable arc delay for hold violation checking. This occurs during a floating bus region, which is described in [Performing Floating Bus Checks](#).

Performing Floating Bus Checks

A floating bus occurs when the bus is at a valid logic value and then begins to float. Timing analysis ensures that the float does not propagate an X to the register data pin until after the hold check. PrimeTime assumes the same setup and hold relationships as for any other data signals.

[Figure 15-8](#) shows another situation that can arise that yields a transient floating bus condition.

Figure 15-8 Floating Bus Contention Check



In [Figure 15-8](#), the hold violation for FF4/CP is seen only if the `three_state_disable` arc delay is considered. A transition to the Z state must be propagated as a possible transition to logic 0 or logic 1 to find all possible cases. A similar situation happens for the setup check. The setup violation is seen only if the `three_state_enable` arc delay is considered.

Because the `three_state_enable` arc delays are considered, by default PrimeTime checks for all setup and hold violations that occur due to the floating bus region. The floating bus region is bounded by the minimum `three_state_disable` arc delay of any bus driver from one side and by the maximum `three_state_enable` arc delay from the other side. PrimeTime ignores charge decay here (it assumes that the logical value for the bus immediately becomes unknown—(X)—when the bus is floating).

If you know that such floating bus regions can never occur, disable checking for both setup and hold violations that occur due to this bus contention region. To disable the checks, set the `timing_disable_floating_bus_check` variable to `true`. In this case, PrimeTime ignores the `three_state_disable` arc delay for hold violations checking and the `three_state_enable` arc delay for setup violations checking. Even when you set this variable to `true`, the `three_state_enable` arc delay is considered for setup violation checking and

the `three_state_disable` arc delay is considered for hold violation checking. This occurs during a bus contention region, which is described in [Performing Transient Bus Contention Checks](#).

Data-to-Data Checking

PrimeTime can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins. This feature can be useful for checking the following types of timing constraints:

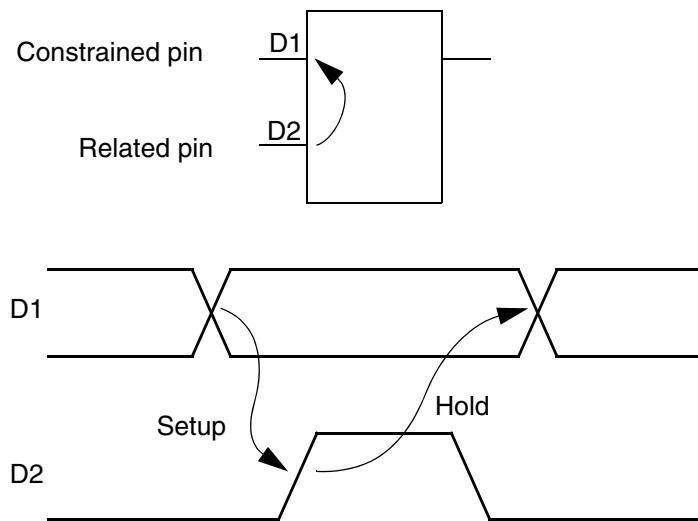
- Constraints on handshaking interface logic
- Constraints on asynchronous or self-timed circuit interfaces
- Constraints on signals with unusual clock waveforms that cannot be easily specified with the `create_clock` command
- Constraints on skew between bus lines
- Recovery and removal constraints between asynchronous preset and clear input pins

A timing constraint between two data (nonclock) signals is called a nonsequential constraint. You can define such checks in PrimeTime by using the `set_data_check` command, or define them for a library cell by setting nonsequential constraints for the cell in Library Compiler. Use data checks only in situations such as those described earlier. Do not consider data checks as a replacement for standard sequential checking.

Data Check Examples

[Figure 15-9](#) shows a simple example of a cell that has a nonsequential constraint. The cell has two data inputs, D1 and D2. The rising edge of D2 is the active edge that might be used to latch data at D1. Pin D1 is called the constrained pin and Pin D2 is called the related pin. In a sequential setup or hold check, pin D2 is considered as the clock pin. However, for any of a number of reasons, it might be desirable to consider the signal at D2 a data signal, and not define it to be a clock.

Figure 15-9 Simple Data Check Example



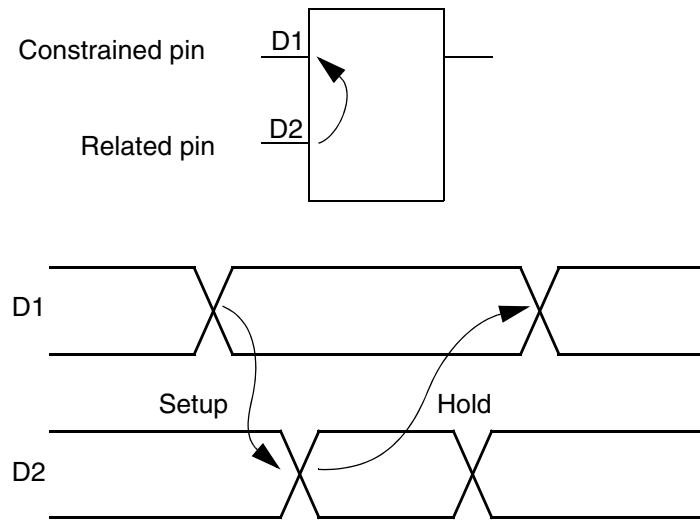
In this example, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge. If these nonsequential constraints are not already defined for the library cell, you can define them in PrimeTime. To do so, use commands similar to the following:

```
pt_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
pt_shell> set_data_check -rise_from D2 -to D1 -hold 6.0
```

The “from” pin is the related pin and the “to” pin is the constrained pin. If the data checks apply to both rising and falling edges on the related pin, use `-from` instead of `-rise_from` or `-fall_from`, as shown in the following example:

```
pt_shell> set_data_check -from D2 -to D1 -setup 3.5
pt_shell> set_data_check -from D2 -to D1 -hold 6.0
```

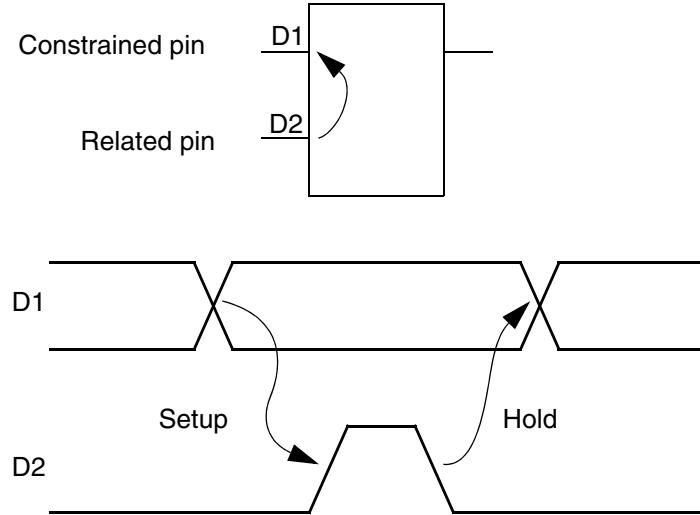
The resulting timing checks are shown in [Figure 15-10](#).

Figure 15-10 Data Checks on Rising and Falling Edges

Define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge:

```
pt_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
pt_shell> set_data_check -fall_from D2 -to D1 -hold 3.0
```

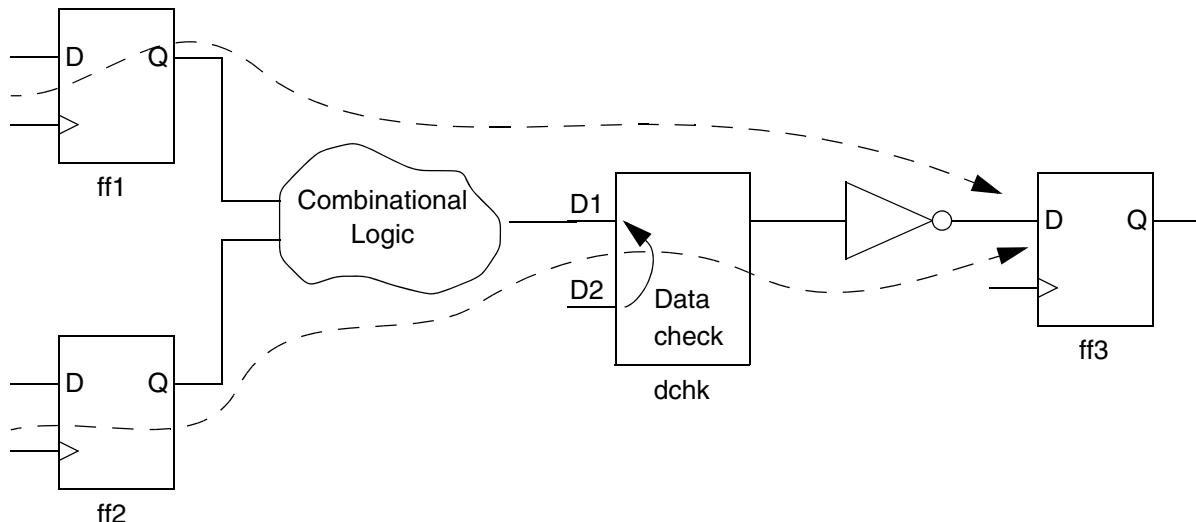
PrimeTime interprets this as a no-change check on a positive-going pulse. The resulting timing check is shown in [Figure 15-11](#).

Figure 15-11 No-Change Data Check

Data checks are nonsequential, so they do not break timing paths. For example, in [Figure 15-12](#), the data check between D1 and D2 does not interrupt the timing paths shown

by the dashed-line arrows. If you define the signal at D2 to be a clock, the check is sequential, and the paths are terminated at D1.

Figure 15-12 Timing Paths Not Broken by Data Checks



You can specify a data check pin as a path endpoint for the `report_timing` command. In that case, PrimeTime reports the data checks that apply to the pin. For example, for the circuit shown in [Figure 15-12](#), `report_timing -to dchk/D1` generates a data check report, whereas `report_timing -through dchk/D1` reports the timing on standard paths that pass through the specified pin.

To remove data checks set with the `set_data_check` command, use the `remove_data_check` command.

Data Checks and Clock Domains

In a data check, signals arriving at a constrained pin or related pin can come from different clock domains. PrimeTime checks the signal paths separately and puts them into different clock groups, just like standard sequential checks.

If the related pin has signals from multiple clock domains, you might want to specify which clock domain to analyze at that pin for the data check. To specify a clock domain to analyze, either use the `-clock clock_name` option of the `set_data_check` command, or disable all clocks other than the clock of interest.

Library-Based Data Checks

PrimeTime performs data checking for any cell that has nonsequential timing constraints defined in the library cell, if the signal at the related pin is not defined to be a clock in PrimeTime. If the signal is defined to be a clock, PrimeTime converts the nonsequential checks to sequential checks and does not block this clock signal from further propagation. If a combinational arc from the related pin exists (such as with an integrated clock-gating cell), the clock is free to continue propagation down this arc.

In Library Compiler, you define nonsequential constraints on a cell by specifying a related pin and by assigning the following `timing_type` attributes to the constrained pin:

```
non_seq_setup_rising  
non_seq_setup_falling  
non_seq_hold_rising  
non_seq_hold_falling
```

For more information about defining nonsequential constraints in Library Compiler, see the Library Compiler user guides.

Defining nonsequential constraints in the library cell results in a more accurate analysis than using the `set_data_check` command because the setup and hold times can be made sensitive to slew of the constrained pin and the related pin. The `set_data_check` command is not sensitive to slew.

To specify which clock domain to use at the related pin for data checks defined in library cells, use the `set_data_check ... -clock clock_name` command. The `remove_data_check` command does not remove data checks defined in library cells.

Interdependent Setup and Hold Pessimism Reduction

Setup and hold pessimism reduction (SHPR) allows you to reduce pessimism in slack computation in path-based analysis. This is done by using the information in a library that supports interdependent setup and hold characterization data.

In a traditional library, the setup and hold constraint arcs for a sequential cell have a single fixed behavior. Setup might be characterized conservatively to allow hold constraint arcs to be characterized aggressively or vice versa. In some libraries, both the setup and hold constraints might be characterized conservatively (large setup/hold requirements) to avoid any possibility of failure during operation. In other libraries, both constraints might be characterized aggressively (small setup/hold requirements) to achieve maximum performance. If both setup and hold are characterized aggressively, failures can result unless care is taken to avoid data pulse widths that are too narrow to be reliably captured by the clock edge.

In reality, the magnitude of the setup and hold constraint requirements depend on each other. In a library with SHPR data, PrimeTime can understand this interdependence between setup and hold and trade off the setup and hold checks against each other to adapt to the needs of each sequential cell to the upstream logic during path-based analysis. By using the SHPR data, PrimeTime can use aggressive setup/hold constraints while still ensuring the minimum data pulse width requirements for reliable capture are met.

To learn more about SHPR, see

- [Use Model for SHPR](#)
 - [SHPR Optimization Mechanism](#)
 - [SHPR User Interface](#)
 - [SHPR Examples](#)
 - [Liberty Format Extension](#)
-

Use Model for SHPR

PrimeTime supports three modes for SHPR. Some user-controlled constraints of SHPR optimization are also available.

- [Setup-Preferred Slack Improvement](#)
- [Hold-Preferred Slack Improvement](#)
- [Total Negative Slack Improvements](#)
- [SHPR Optimization Constraints](#)

Setup-Preferred Slack Improvement

For some designs that prefer fast clock frequency (and assuming the hold time violation is easy to fix), the setup-preferred slack improvement mode can be desirable. This mode optimizes only total negative setup slacks of `max_rise` and `max_fall` paths ending at a specific flip-flop, with the trade off of corresponding hold slacks.

Hold-Preferred Slack Improvement

For some designs in which the setup time violation can be easily fixed by lowering clock frequency, hold time violation fixing becomes the primary consideration. The hold-preferred slack improvement mode is designed for such a case. This mode optimizes only total negative hold slacks of `min_rise` and `min_fall` paths ending at a specific flip-flop, with the trade off of corresponding setup slacks.

Total Negative Slack Improvements

The total negative slack mode minimizes the sum of negative slacks of all four types of timing paths ending at a particular flip-flop. By default, this mode allows trade off between the positive slack of one type of timing path and the negative slack of another type of timing path, without incurring new timing violations. This mode gives the same priorities on both negative setup slack and negative hold slack. This is the default mode of SHPR.

SHPR Optimization Constraints

For the three modes just described, user controls are provided to influence the slack trade-off. This is known as the SHPR optimization constraint, which defines the limit of trade off of the positive setup or hold slack. When one type of slack is sacrificed to improve the other type of slack, specify the slack limit for the sacrificial slack trade off.

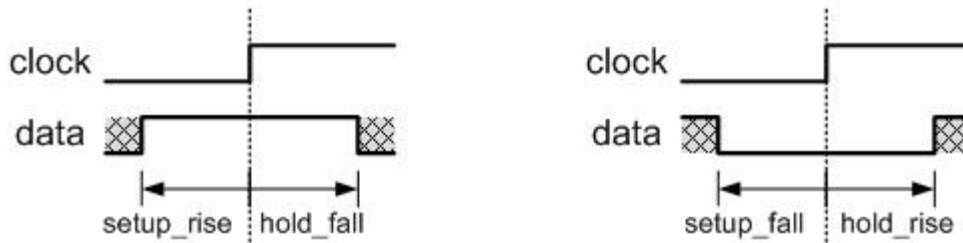
SHPR Optimization Mechanism

In SHPR, a sequential capturing endpoint can be traded off in only two independent scenarios:

- `setup_rise` against `hold_fall`
- `setup_fall` against `hold_rise`

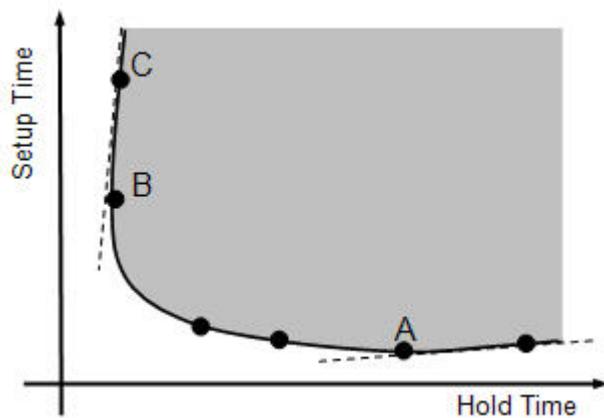
[Figure 15-13](#) shows these two scenarios.

Figure 15-13 Two Setup and Hold Trade off Scenarios



[Figure 15-14](#) shows a typical interdependent setup and hold value curve. On the curve, any points beyond point A or B have a positive slope. Points A and B are called the turning points.

Figure 15-14 Interdependent Setup and Hold Value Curve



To be conservative, PrimeTime does not allow any extrapolation beyond the boundary points of the characterized SHPR curve.

SHPR User Interface

To set the optimization constraints for SHPR, use the `set_setup_hold_pessimism_reduction` command. The valid modes are total, setup, and hold. Use the different modes to control how setup slack is traded off for hold slack or vice versa.

In setup mode (also known as setup-preferred mode), the goal is to prefer positive setup slack at the expense of hold slack. In hold mode (also known as hold-preferred mode), the goal is to prefer positive hold slack at the expense of setup slack. In total mode (the default), the goal is to trade off setup and hold slack in a way that minimizes the sum of the negative rise/fall and setup/hold slack. This mode results in the smallest overall set of setup and hold timing violations.

It might not be desirable to introduce a large violation in the sacrificial slack type to gain only a small improvement in the preferred slack type. Use the `-setup_cutoff` and `-hold_cutoff` options of the `set_setup_hold_pessimism_reduction` command to keep the trade off reasonable. In setup-preferred mode, hold violations are made no worse than the `-hold_cutoff` value to improve setup. In hold-preferred mode, setup violations are made no worse than the `-setup_cutoff` value to improve hold. The default for both cutoff options is negative infinity, which allows any amount of slack worsening in the sacrificial slack to improve the preferred slack.

In both setup and hold mode, the preferred slack type is improved only as far as it takes to reach zero slack and no further.

To disable SHPR, use the `remove_setup_hold_pessimism_reduction` command, which has the following options:

- `-setup_cutoff` – Resets cutoff setup slack.
- `-hold_cutoff` – Resets cutoff hold slack.

If you do not add an option, the SHPR feature is completely disabled.

SHPR Examples

For examples of SHPR, see

- [Setup-Preferred Slack Improvement Example](#)
- [Total Slack Improvement Example](#)

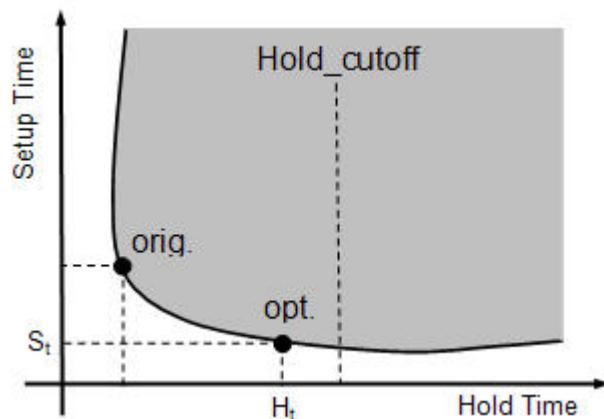
Setup-Preferred Slack Improvement Example

In the following example, the mode is setup-preferred slack improvement, and the cutoff hold slack is -100 ps.

```
pt_shell> set_setup_hold_pessimism_reduction -mode setup \
           -hold_cutoff -100
```

The targeted setup time that results in zero slack in the max timing path is determined as S_t in [Figure 15-15](#).

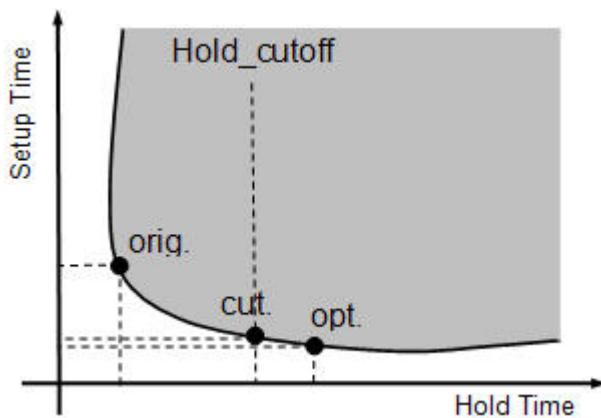
Figure 15-15 Successful Setup Slack Improvement



Using S_t as the input, obtain the corresponding hold time H_t , by using the interdependent setup and hold value curve. If H_t does not reach the `hold_cutoff` value, S_t and H_t are the optimized values of setup and hold time, as shown in [Figure 15-15](#).

Instead, if H_t is beyond the `hold_cutoff` value, as shown in [Figure 15-16](#), the corresponding setup and hold times of the cutoff point are considered as the optimized setup and hold values.

Figure 15-16 Setup Slack Improvement Stops in Cutoff Hold Slack



Total Slack Improvement Example

The total mode is a special case. It does not find the minimal negative slack of min and max timing paths along the whole interdependent setup and hold curve. Instead, one of the following two cases with the minimal total negative slack of min and max timing paths is chosen:

- Case 1 – Setup-preferred slack improvement mode, `hold_cutoff`
- Case 2 – Hold-preferred slack improvement mode, `setup_cutoff`

Liberty Format Extension

The Liberty format and Library Compiler support interdependent setup and hold tables. The `interdependence_id` tag is used to identify the interdependent setup and hold tables.

Table 15-1 shows the library format needed to support SHPR.

Table 15-1 Library Format Needed to Support SHPR

<pre> timing() { related_pin : "CP"; timing_type : setup_rising; rise_constraints (setup_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx"); } fall_constraints (setup_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx"); } } timing() { related_pin : "CP"; timing_type : setup_rising; interdependence_id : 1; rise_constraints (setup_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx"); } fall_constraints (setup_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx", \ "xxxx, xxxx, xxxx"); } } </pre>	<pre> timing() { related_pin : "CP"; timing_type : hold_rising; rise_constraints (hold_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy"); } fall_constraints (hold_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy"); } } timing() { related_pin : "CP"; timing_type : hold_rising; interdependence_id : 1; rise_constraints (hold_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy"); } fall_constraints (hold_template_3x3) { index_1 ("0.0264, 0.1728, 0.6736"); index_2 ("0.0264, 0.1728, 1.9424"); values ("yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy", \ "yyyy, yyyy, yyyy"); } } </pre>
--	--

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

Time Borrowing in Latch-Based Designs

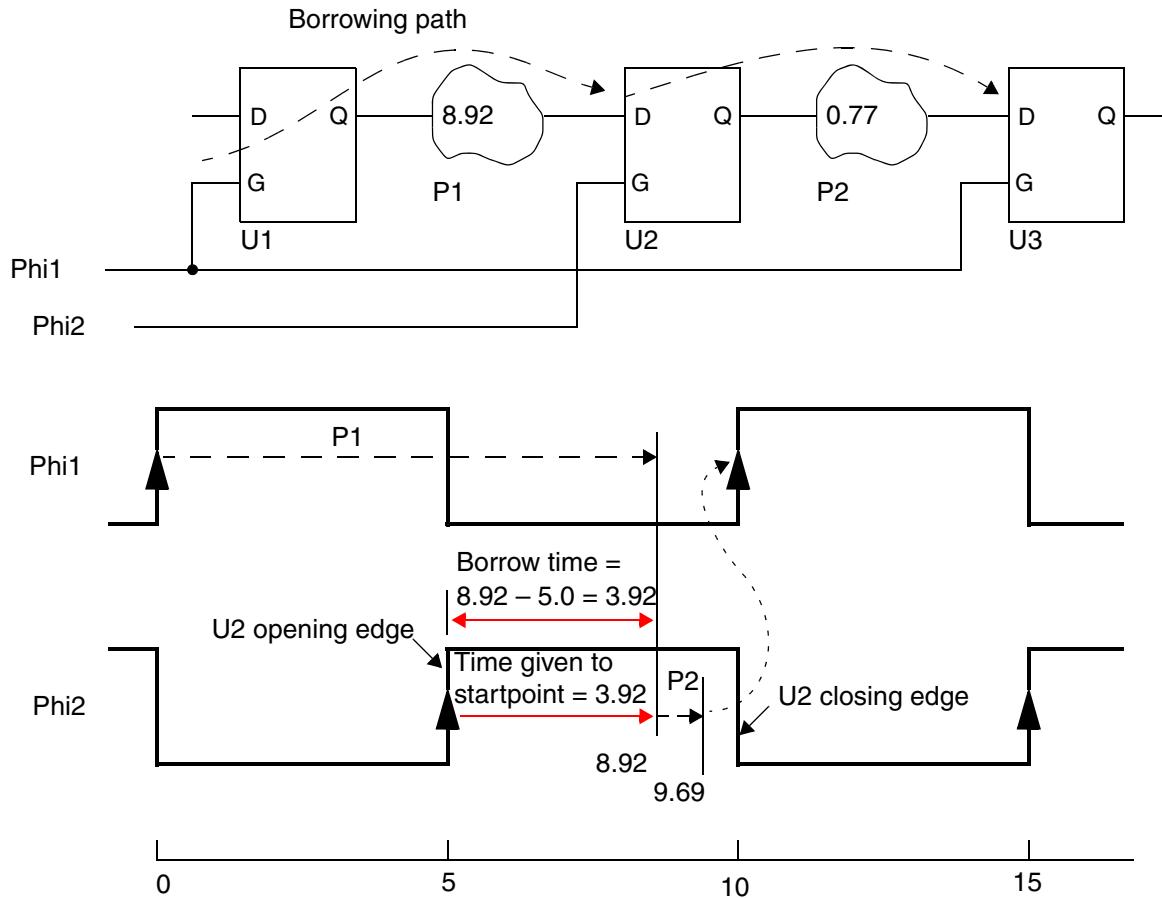
Transparent latches present unusual challenges for static timing analysis tools. A technique known as time borrowing (also known as “cycle stealing”) gives latch-based designs a distinct advantage over flip-flop based designs because a level-sensitive latch is transparent for the duration of an active clock pulse. This technique can relax the normal edge-to-edge timing requirements of synchronous designs. However, it is harder to control the timing of latch-based designs because of the multiphase clocks used and the lack of “hard” clock edges at which events must occur.

Borrowing Time From Logic Stages

In a design using level-sensitive latches, a path can borrow time from the next logic stage by taking advantage of the fact that latches are transparent while the gate input is asserted.

[Figure 15-17](#) shows latch-based stages using a simple two-phase clocking scheme.

Figure 15-17 Latch-Based Timing Paths



A design using level-sensitive latches allows a combinational logic path with a delay longer than the available cycle time if it is compensated by shorter path delays in subsequent latch-to-latch stages. For the two-phase design, the available time for latch-to-latch paths is half the clock cycle.

In [Figure 15-17](#), U1, U2, and U3 are positive-level-sensitive latches (active when G = 1), and P1 and P2 are combinational logic paths. For now, assume a library setup time of zero for the latches and zero delay from D to Q in transparent mode. For positive-level-sensitive latches, PrimeTime uses the rising (opening) edge as the reference edge.

The figure shows path delays of $P1 = 8.92$ and $P2 = 0.77$. There might appear to be a violation at U2 because the data arrives at U2 after the rising edge of phi2. However, because the U2 latch is transparent for 5 ns and P2 is less than that amount, path P1 can borrow the slack time (3.92 ns) from the path between U2 and U3. Therefore, the sum of P1 and P2 is 9.69, which is less than the required time of 10.00 at U3.

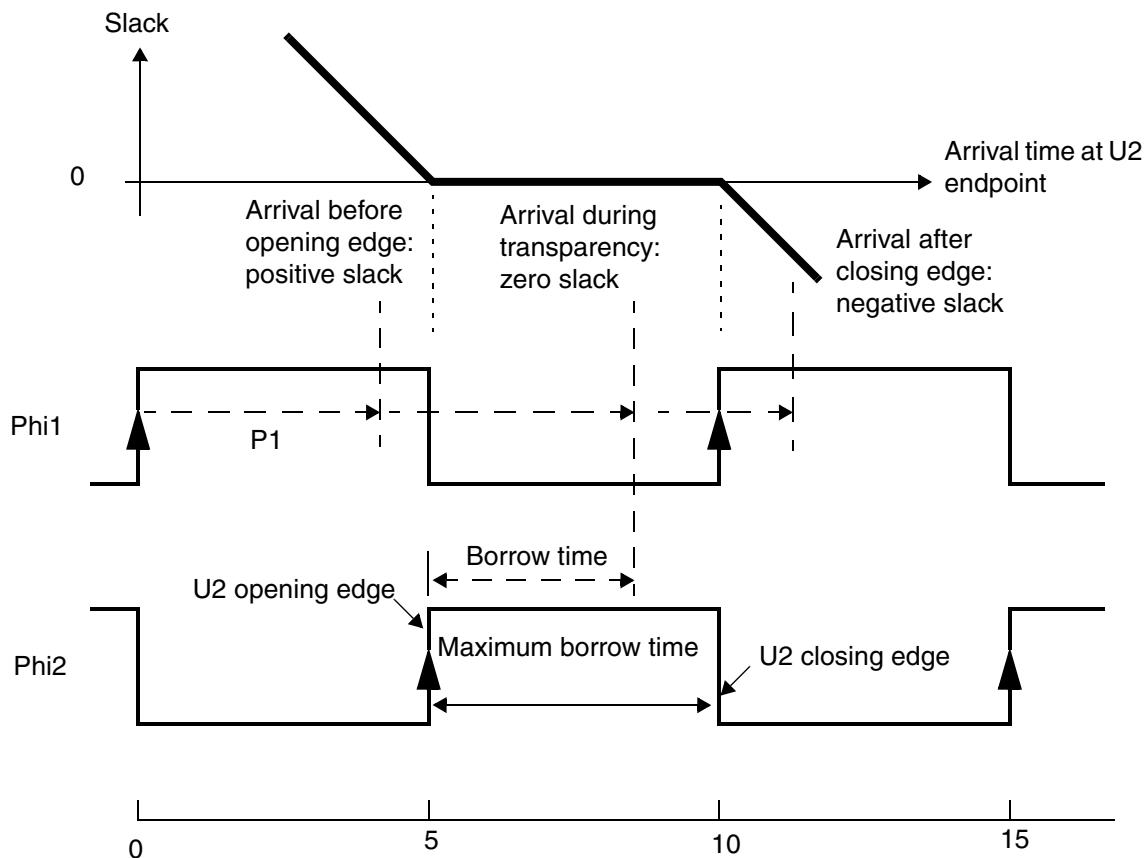
Latch Timing Reports

If the data signal arrives before the opening edge at the endpoint latch, PrimeTime models this behavior just as it would for a flip-flop. The opening edge of the clock (rising edge in this example) captures the data at the endpoint. The same clock edge launches the data at the startpoint of the next path.

On the other hand, if the data signal arrives while the latch is transparent (after the opening edge but before the closing edge at the endpoint latch), PrimeTime models the behavior at the next stage as a launch from the data pin of the latch, rather than from the clock pin. In this case, there is no timing violation and the slack is considered zero. The amount of time borrowed by the stage ending at the latch becomes the departure time for the next stage, subject to certain adjustments described in [Maximum Borrow Time Adjustments](#). A data signal arriving after the closing edge at the endpoint latch is a timing violation.

[Figure 15-18](#) shows how PrimeTime calculates and reports slack for a range of combinational delays through P1, which results in different arrival times at U1 (see the schematic in [Figure 15-17](#)). [Figure 15-18](#) does not consider the effects of latch setup time, latency, uncertainty, and clock reconvergence pessimism removal.

Figure 15-18 Latch-Based Timing Path Slack Calculation



The following example shows how the `report_timing` command reports a timing path ending at a transparent latch with time borrowing. The “time borrowed from endpoint” and “time given to startpoint” statements in this report correlate with [Figure 15-17](#).

```
*****
Report : timing
    -path short
    -delay max
    -max_paths 1
Design : time_borrow
*****
Wire Loading Model Mode: enclosed

Startpoint: U1 (positive level-sensitive latch clocked by
Phi1)
Endpoint: U2 (positive level-sensitive latch clocked by Phi2)
Path Group: Phi2
Path Type: max
```

Point	Incr	Path
clock Phi1 (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
U1/G (LATCH)	0.00	0.00 r
U1/Q (LATCH)	0.57	0.57 r
...		
U2/D (LATCH)	8.35	8.92 r
data arrival time		8.92
clock Phi2 (rise edge)	5.00	5.00
clock network delay (ideal)	0.00	5.00
U2/G (LATCH)	0.00	5.00 r
time borrowed from endpoint	3.92	8.92
data required time		8.92
data required time		8.92
data arrival time		-8.92
slack (MET)		0.00
Time Borrowing Information		
Phi2 nominal pulse width	5.00	
library setup time	-0.46	
max time borrow	4.54	
actual time borrow	3.92	

Startpoint: U2 (positive level-sensitive latch clocked by Phi2)
 Endpoint: U3 (positive level-sensitive latch clocked by Phi1)
 Path Group: Phi1
 Path Type: max

Point	Incr	Path
clock Phi2 (rise edge)	5.00	5.00
clock network delay (ideal)	0.00	5.00
time given to startpoint	3.92	8.92
U2/D (LATCH)	0.00	8.92 r
U2/Q (LATCH)	0.53	9.45 r
...		
U3/D (LATCH)	0.24	9.69 f
data arrival time		9.69
clock Phi1 (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
U3/G (LATCH)	0.00	10.00 r
time borrowed from endpoint	0.00	10.00

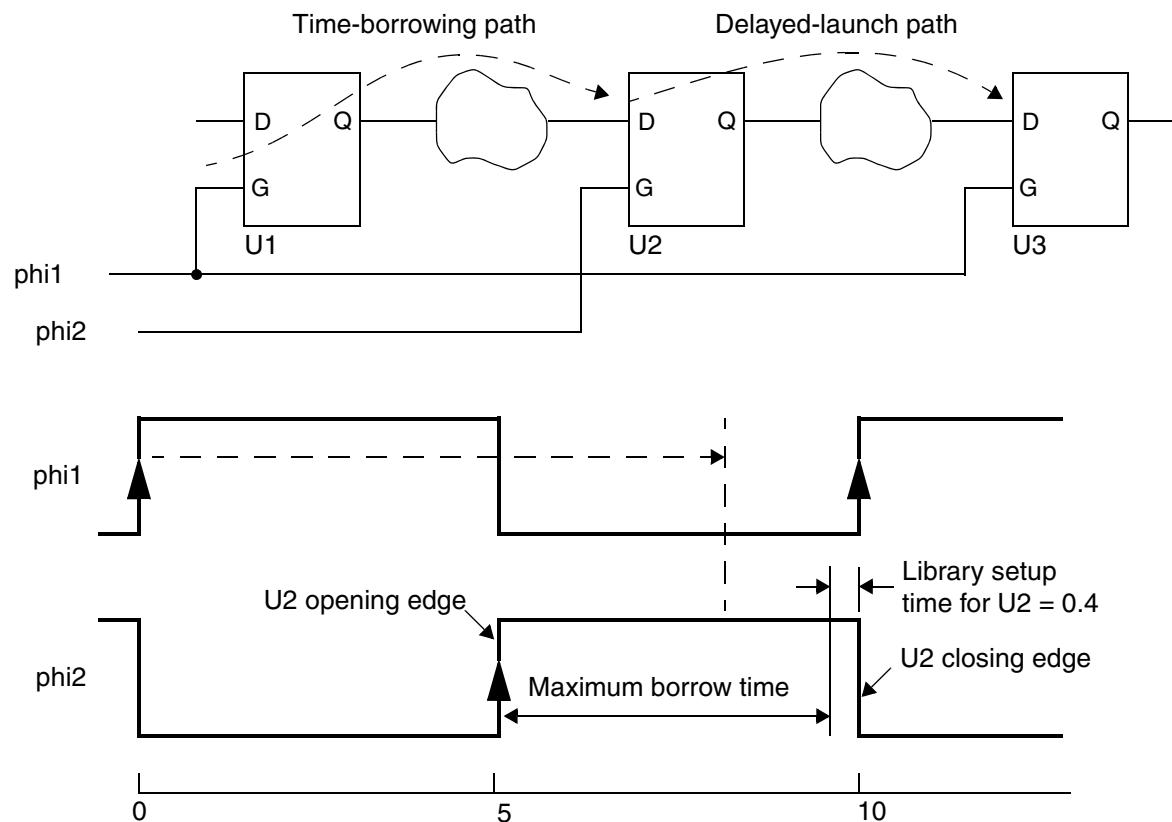
data required time	10.00
-----	-----
data required time	10.00
data arrival time	-9.69
-----	-----
slack (MET)	0.31
 Time Borrowing Information	
-----	-----
Phil nominal pulse width	5.00
library setup time	-0.49
-----	-----
max time borrow	4.51
actual time borrow	0.00
-----	-----

For the U2 to U3 path, time must be added to compensate for the time borrowed, so PrimeTime adds 3.92 ns to the launch time of U2. This is reported in the second path's timing report. Because the P2 path has enough slack, neither path is in violation.

Maximum Borrow Time Adjustments

The maximum amount of time that can be borrowed at an endpoint latch is based on the clock pulse width (the time from the opening edge to the closing edge of the gate signal) as defined by the `create_clock` command, minus the library setup time of the latch, which is shown in [Figure 15-19](#).

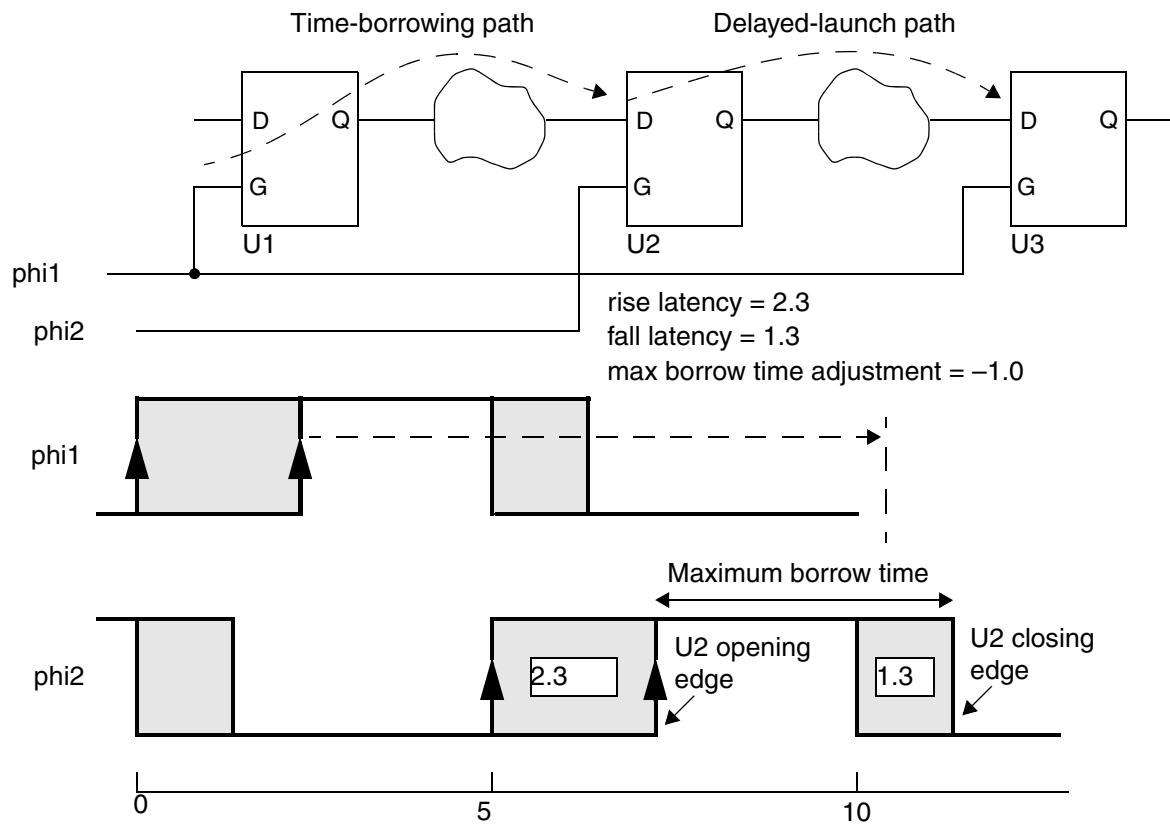
Figure 15-19 Maximum Borrow Time Reduced by Setup Requirement



For better accuracy, PrimeTime adjusts this amount further for the effects of clock latency, clock uncertainty, and clock reconvergence pessimism removal.

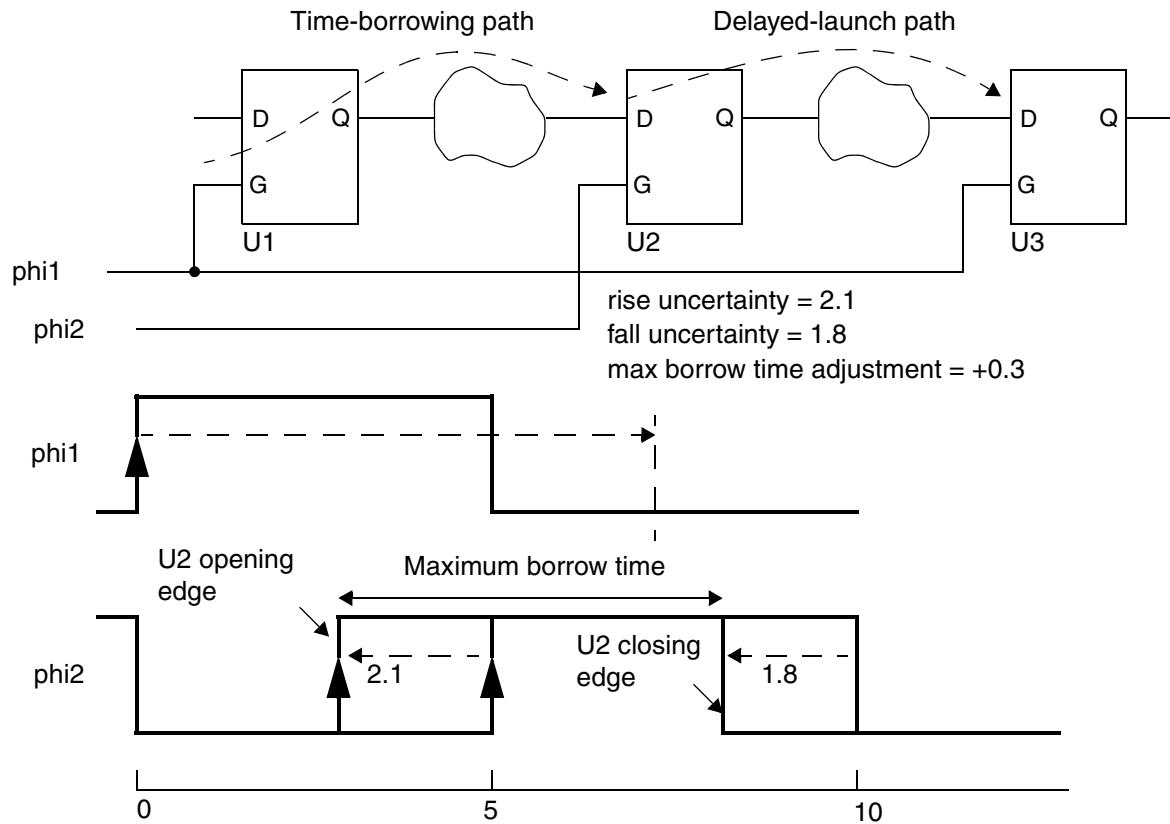
Clock latency can be defined with the `set_clock_latency` command or calculated by PrimeTime from propagated delays in the clock tree (enabled by `set_propagated_clock`). Latency values can be different for the rising and falling edges of the clock pulse, which can affect the pulse width and thus the maximum borrow time. See [Figure 15-20](#).

Figure 15-20 Maximum Borrow Time Adjustment for Latency



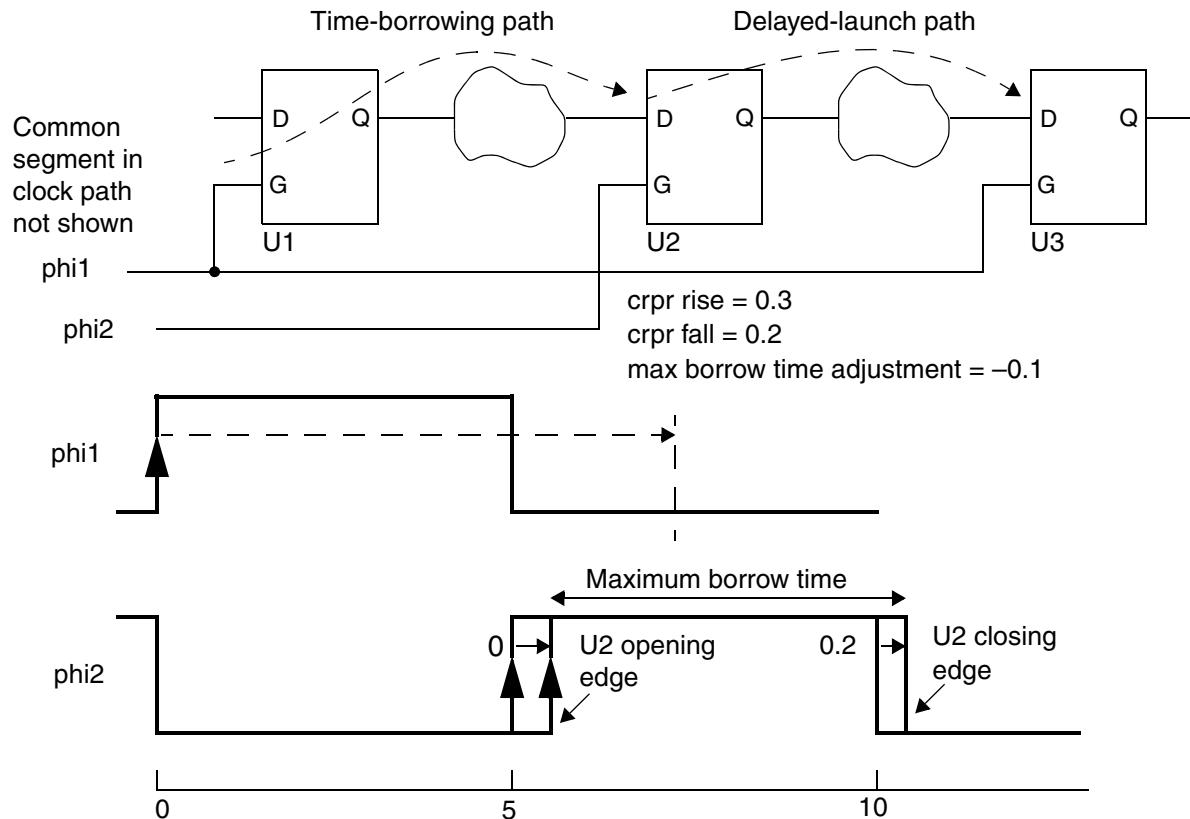
Clock uncertainty can be defined with the `set_clock_uncertainty` command. For a setup check, PrimeTime considers the earliest possible arrival of capture clock edges. Differences in uncertainty between rising and falling edges can affect the clock pulse width and maximum borrow time. See [Figure 15-21](#).

Figure 15-21 Maximum Borrow Time Adjustment for Uncertainty



Clock reconvergence pessimism removal (CRPR) is an analysis technique that corrects any inaccuracy resulting from a common segment in the launch and capture clock paths. To enable this feature, set the `timing_remove_clock_reconvergence_pessimism` variable to true.

Application of CRPR shifts the clock edge times, like clock uncertainty. However, applying clock uncertainty makes the analysis more pessimistic, whereas applying CRPR makes the analysis less pessimistic, so the direction of the edge shift is in the positive direction rather than the negative direction. Differences in CRPR between rising and falling edges can affect the clock pulse width and maximum borrow time, which is shown in [Figure 15-22](#).

Figure 15-22 Maximum Borrow Time Adjustment for CRPR

To calculate the maximum allowable borrow time, PrimeTime starts with the clock pulse width and then adjusts it for the applicable effects of clock latency, clock uncertainty, clock reconvergence pessimism removal, and library setup time of the endpoint latch. The `report_timing` command reports the clock pulse width and the adjustments as follows:

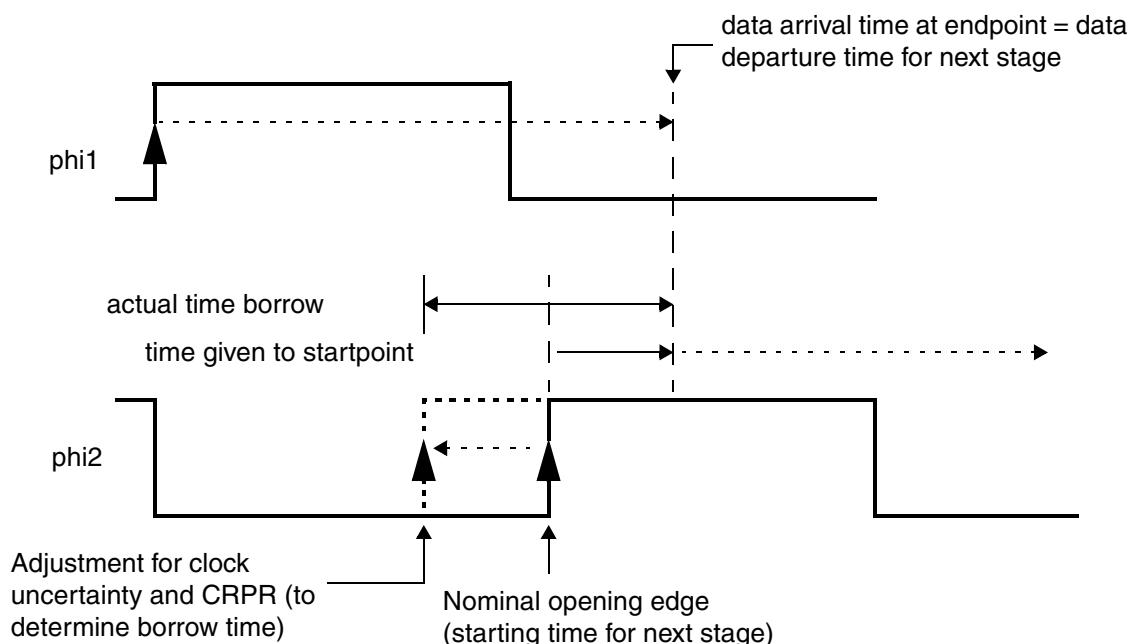
```
Time Borrowing Information
-----
CLK nominal pulse width           5.00
clock latency difference          -1.00
clock uncertainty difference       0.30
CRPR difference                  -0.10
library setup time                -0.40
-----
max time borrow                   3.80
-----
...
```

Time Borrowed and Time Given

PrimeTime calculates the amount of time borrowed in relation to the arrival time of the opening clock edge at the latch. PrimeTime first adjusts the arrival time of the opening edge to account for path-specific effects of both uncertainty and CRPR (if applicable). Then it compares the adjusted value to the signal arrival time at the data pin to determine the amount of time borrowed at the path endpoint, if any.

If uncertainty and CRPR exist for the opening edge of the latch, the time borrowed is different from the amount of time given to the startpoint of the next stage. To determine the time given to the startpoint, PrimeTime subtracts the uncertainty and CRPR adjustments. This subtraction is necessary in transparent mode to make the launch at the next stage occur precisely when the signal arrives at the data pin, as shown in [Figure 15-23](#). When the `timing_early_launch_at_borrowing_latches` variable is disabled, the data arrival and launch times are not identical, owing to the deliberate application of a late clock latency to launch the next stage. This mode is recommended when CRPR is enabled. Note that the CRPR adjustment to the time given to the startpoint of the next stage is not applied in this mode.

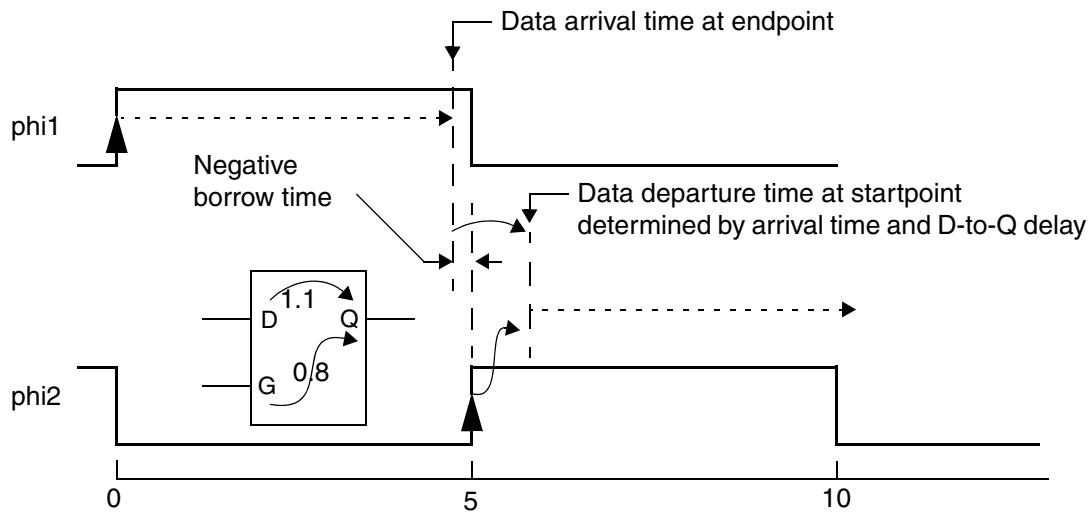
Figure 15-23 Borrow Time and Time Given to Startpoint in Transparent Mode



The `report_timing` command reports the amount of time borrowing and uncertainty and CRPR adjustments as follows:

```
Time Borrowing Information
-----
CLK nominal pulse width      5.00
clock latency difference     -1.00
clock uncertainty difference  0.30
CRPR difference              -0.10
library setup time            -0.40
-----
max time borrow               3.80
-----
actual time borrow            3.40
open edge uncertainty         -2.10
open edge CRPR                0.30
-----
time given to startpoint      1.60
-----
```

In most cases, data arrival before the opening clock edge results in no borrowing, whereas data arrival after the opening clock edge results in borrowing. When borrowing occurs, the arrival time minus the clock edge time is reported as “time borrowed from endpoint” from the perspective of the path segment ending at the latch. This same amount of time is reported as “time given to startpoint” from the perspective of the path segment starting from the latch. However, in certain cases, data arrival just before the clock edge can result in borrowing. This happens when the D-to-Q delay of the latch is more than its clock-to-Q delay, and the arrival time is very close to the clock edge, as shown in [Figure 15-24](#). Under these circumstances, the data departure time for the next path is determined by the data arrival time and the D-to-Q delay, rather than the clock-to-Q delay of the no-borrowing case.

Figure 15-24 Negative Borrow Time

PrimeTime accounts for this condition by allowing time borrowing. The arrival time minus the clock edge time is a negative number, so the amount of borrowing is negative. The borrowed amount is reported as “time given to endpoint” from the perspective of the path segment ending at the latch, and “time borrowed from startpoint” from the perspective of the path segment starting from the latch. This amount of time cannot exceed the difference between the D-to-Q delay and clock-to-Q delay of the latch.

Limiting Time Borrowing

The `set_max_time_borrow` command limits time borrowing to a specified amount for all latch endpoints within a specified scope of the design. Use this command to set a more restrictive constraint on borrowing. At the specified latch endpoints, PrimeTime limits borrowing to the specified amount or to the default determined by the adjusted pulse width, whichever is smaller.

If the `default_max_time_borrow` constraint is zero, no time borrowing is allowed and PrimeTime analyzes the latch like a flip-flop.

If the `default_max_time_borrow` constraint is negative, the data signal must be stable before the open edge of the clock. Use this feature to ensure that the enable input arrives before the clock (on a gated-clock latch, for example).

To show the `max_time_borrow` attributes, use the `report_exceptions` command. To remove the maximum time borrow limit set on specified objects using `set_max_time_borrow`, use the `remove_max_time_borrow` command.

These commands set maximum time borrowing of 2.5 on all latches clocked by clk:

```
pt_shell> create_clock -period 10 -waveform {0 5} clk
pt_shell> set_max_time_borrow 2.5 find(clock,clk)
pt_shell> report_timing -to latch1/D
```

The `user_max_time_borrow` constraint (from the `set_max_time_borrow` command) effectively reduces the pulse available for time borrowing from 5.0 to 2.5:

```
Time Borrowing Information
-----
user max_time_borrow           2.50
-----
max time borrow                2.50
actual time borrow             0.00
-----
```

The time borrowed from an endpoint is usually the same as the time given to the next startpoint when the endpoint and startpoint are both the D pin of the same latch. There can be a small difference due to uncertainty and CRPR adjustment of the opening edge of the latch. Sometimes, if the borrowing at D is small, the most critical path from the latch still comes from G; however, if you do the explicit report from D, then the time given is the same as the time borrowed.

PrimeTime ADV Advanced Latch Analysis

By default, timing violations are reported by analyzing single-segment paths between latches. Borrowing paths are introduced for borrowing (or failing) latches. The borrowing paths are single-segment paths that end at the D pin of the next latch or flip-flop. The single-segment nature of timing paths can be an obstacle to fixing timing and performing power optimization on latch designs.

You can optionally use PrimeTime ADV to analyze paths through latches without breaking the paths into segments. In that case, a transparent latch is both a throughpath and an endpoint. Each latch can have paths ending at the D pin of the latch, as well as paths passing through the latch toward another endpoint. In addition, each latch clock pin can be a startpoint of a path. Advanced latch analysis provides global visibility of timing paths through latches, which can improve power and design optimization.

The following table summarizes the commands and variables for advanced latch analysis:

Table 15-2 Commands and Variables for Advanced Latch Analysis

Command or variable	Usage
timing_enable_through_paths	Enables advanced latch analysis
set_latch_loop_breaker	Breaks latch loops at specified points, overriding the default points chosen by the tool
get_latch_loop_groups	Returns a list of collections of pins, each collection containing the data pins of a latch loop group
report_latch_loop_groups	Reports information about latch data pins involved in loops of transparent latches
timing_through_path_max_segments	Specifies the maximum number of successive latch path segments analyzed per path
timing_report_skip_early_paths_at_intermediate_latches	Prevents reporting of throughpaths that arrive early at intermediate latches
report_timing -trace_latch_borrow -trace_latch_forward	Traces paths backward or forward through loop-breaker latches

For details about running the PrimeTime ADV advanced latch analysis, see the following topics:

- [Enabling Advanced Latch Analysis](#)
- [Breaking Loops](#)
- [Latch Loop Groups](#)
- [Timing Exceptions Applied to Latch Paths](#)
- [Reporting Paths Through Latches](#)
- [Calculating the Worst and Total Negative Slack](#)
- [Normalized Slack Analysis](#)
- [Finding Recovered Paths](#)

Enabling Advanced Latch Analysis

To enable advanced latch analysis, set the `timing_enable_through_paths` variable to `true`. By default, this variable is set to `false`.

Breaking Loops

Loops present a challenge when viewing throughpaths as timing paths. Finding the worst path through a circuit with loops is only possible for very small, simple circuits.

To avoid issues with loops, selected latches are designated as loop-breaker latches. Paths do not propagate through loop-breaker latches. By default, loop-breaker latches are selected by PrimeTime. The tool tries to select a small set of loop-breaker latches, based on the connectivity of the design. The tool does not consider arrival values when selecting loop-breaker latches.

The `report_timing` command does not find paths through loop-breaker latches. For each `report_timing` command, the tool reports the worst timing path based on path specifiers that does not pass through a loop-breaker latch.

Specifying Loop-Breaker Latches

To manually specify loop-breaker latches, use this command:

```
set_latch_loop_breaker -pin pin_list
```

You should specify a particular latch as a loop breaker if

- The latch is not on a critical path
- The latch is part of a path that is not a loop, but the tool might detect a latch loop (such as a register file with a read and write port, which are never used simultaneously)
- The latch is used with a pulse clock or has only a small transparency window
- There are other latches in a latch loop that should not be treated as loop breakers

Finding Loop-Breaker Latches

To find the loop-breaker latches, query the `is_latch_loop_breaker` attribute on the pins of sequential cells. This attribute is set to `true` for the D pin of a loop-breaker latch. For example, to create a collection of loop-breaker latch pins:

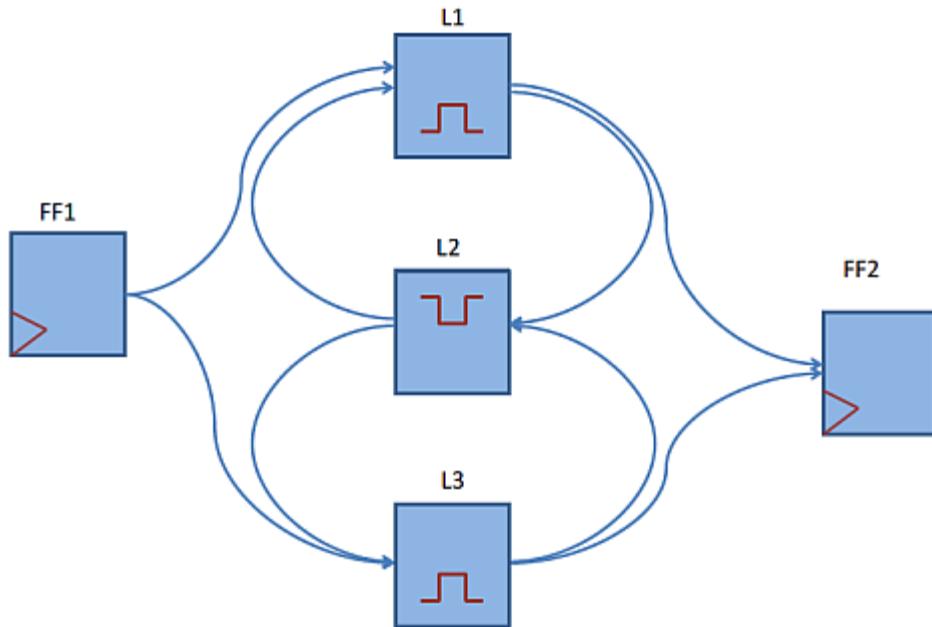
```
get_pins -hierarchical * -filter "@is_latch_loop_breaker"
```

Latch Loop Groups

A single latch can be part of multiple latch loops. The set of all intersecting latch loops is called a *latch loop group*. Every latch in the latch loop group is involved in at least one loop with every other latch in the group.

In [Figure 15-25](#), a single latch loop group contains three latches (L1, L2, and L3). Between every two latches in the latch loop group, it is possible to form a loop.

Figure 15-25 Single Latch Loop Group With Three Latches



Listing Collections of Latch Loop Groups

To list the collections that contain the latch D pins that make up a particular latch loop group, use this command:

```
get_latch_loop_groups
  [-of_objects list_of_transparent_d_pins]
  [-loop_breakers_only]
```

This command returns a Tcl list of collections. Each collection has the latch D pins that make up a particular latch loop group. By default, the command returns every latch loop group. If you use the `-of_objects` option, the tool includes only those loop groups that include the specified latch D pins. If you use the `-loop_breakers_only` option, the tools includes only loop breaker pins in the collections.

The command does not report latches outside of loops.

Reporting Latch Loop Groups

To report information about latch loop groups, use the `report_latch_loop_groups` command. [Example 15-1](#) shows the report. The report lists the latch D pins on the left. The latch loop groups are given a number listed in the second column. You can distinguish which latch D pins are in which group by the number. The attributes on the right indicate if the latch is a loop breaker, and also indicates whether you requested the D pin to be a loop breaker or to be avoided as a loop breaker.

Example 15-1 Report of Latch Loop Groups

```
pt_shell> report_latch_loop_groups
*****
Report : latch loop groups
...
*****
Attributes
b - loop breaker d pin
p - long path breaker d pin, but not in a loop
u - user requested to be a loop breaker using set_latch_loop_breaker
a - user requested to avoid with set_latch_loop_breaker -avoid
Latch Latch Loop Attributes
D pin Group
-----
DUT/Latch1/D NA pu
DUT/Latch2/D 1 b
DUT/Latch3/D 1
DUT/Latch4/D 2 bu
DUT/Latch5/D 2 a
```

If a latch D pin is not in a loop, the latch loop group column indicates “NA”. It is possible for latch D pins that are not in a loop to be path breakers. If you do not use the `-loop_breakers_only` option, the tool does not report these pins.

Specifying the Maximum Number of Latches Analyzed per Path

By default, the tool reports a maximum of five successive latches per path. The tool limits the maximum length of each path by introducing additional loop-breaker latches even where there are no loops.

To override the default behavior and consider a different maximum number latches per path, set the `timing_through_path_max_segments` variable to the desired number. Set a higher number increase the analysis accuracy for long paths. Set a lower number to decrease the runtime.

A setting of zero means no limit on the number of latches analyzed per path. If this setting causes excessively long runtimes, you should revert to the default setting or use some other reasonable setting.

Timing Exceptions Applied to Latch Paths

When using advanced latch analysis, timing exceptions must be satisfied within one path segment. All `-from`, `-through`, and `-to` option specifiers must be met in one path segment to be applied.

To learn how PrimeTime ADV applies timing exceptions when running advanced latch analysis, see

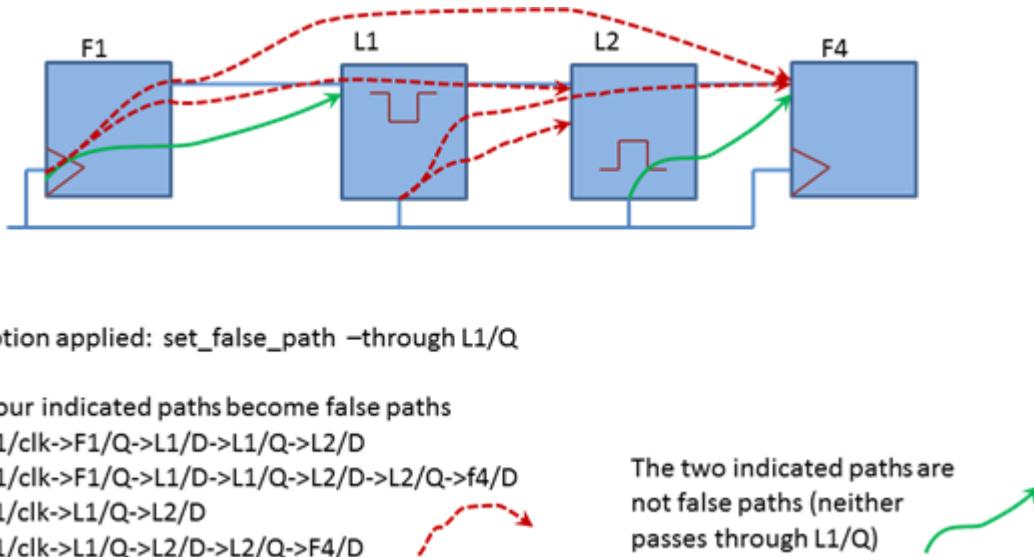
- [False Path Exceptions](#)
- [Multicycle Path Exceptions](#)
- [Maximum and Minimum Delay Exceptions](#)
- [Clock Groups](#)
- [Specification of Exceptions on Throughpaths](#)

False Path Exceptions

If a throughpath has a satisfied false path exception on any of its segments, the path becomes a false throughpath, and the tool does not check constraints at the end of the path.

[Figure 15-26](#) shows an example of a false path exception.

Figure 15-26 False Path Exceptions on Throughpaths

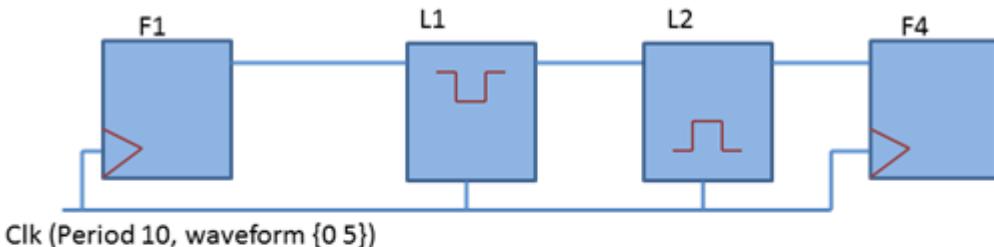


Multicycle Path Exceptions

If a throughpath has a satisfied multicycle path exception on a segment, the exception affects the expected transparency window of latches downstream of the exception, and affects the required time at the path endpoint. Different multicycle path exceptions can apply to different segments of the same throughpath.

[Figure 15-27](#) shows examples of multicycle path exceptions.

Figure 15-27 Multicycle Path Exceptions Affect the Downstream Latch Windows



Exceptions applied	F1 Launch	L1 Window	L2 Window	F4 Required
No exception	0	5 - 10	10 - 15	20
set_multicycle_path 2 –through L1/Q	0	5 - 10	20 – 25	30
set_multicycle_path 2 –through L1/Q set_multicycle_path 2 –through L2/Q	0	5 – 10	20 – 25	40

In [Figure 15-27](#), there is a throughpath from F1 to F4. When both exceptions are applied, the required time for the path is 40 (minus setup time). For the path from F1 to L2/D, the required time is 25 (minus setup time) if the exceptions are applied.

The multicycle path exceptions also affect normalized slack by changing the allowed propagation delay of paths.

Maximum and Minimum Delay Exceptions

The tool supports maximum delay exceptions for throughpaths when the endpoint of the maximum delay is the D pin of a latch. The exceptions apply a maximum delay to the segment only, overriding the normal pulse relation for the segment. Throughpaths for which the maximum delay exception is not satisfied are not affected, even if they pass through the D pin. For throughpaths that are affected by the exception, the exception affects the local constraint at the latch but does not affect path recovery or normalized slack calculations for the path.

Minimum delay exceptions ending at the D pin of a latch affect local hold time constraints for single-segment paths. Throughpaths are not affected because hold checks are not performed for throughpaths.

Clock Groups

Use clock groups to indicate that two clocks do not communicate. You cannot use clock-to-clock false path exceptions to do this because they do not work for all throughpaths; the false path exception does not conform to the rule that the exception must start and become satisfied within one segment.

Note:

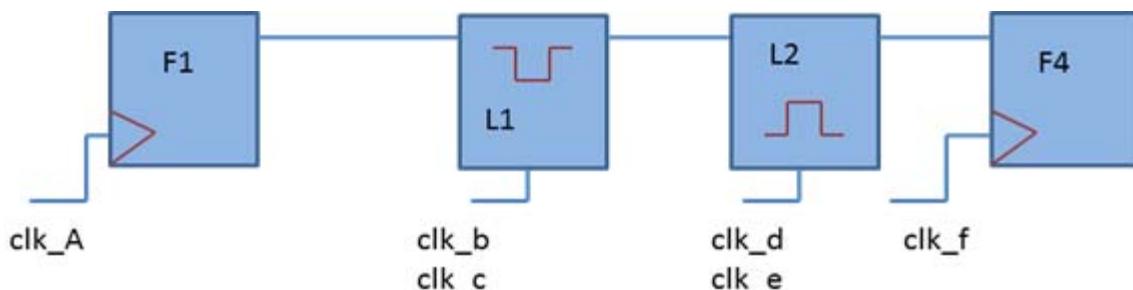
You can use clock-to-clock false paths for hold checks because hold checks are not applied on throughpaths.

The tool applies clock-to-clock exceptions only if the throughpath has a segment that satisfies the exception. For example, if the first intermediate latch is clocked by clkA, and the second intermediate latch is clocked by clkB, then `set_false_path -from clkA -to clkB` causes the throughpath to be false.

Clock groups are taken into account for throughpaths. Throughpaths are not constrained if the throughpath relies on two clocks that are exclusive. This is true even if the throughpath relies on the exclusive clocks only at intermediate latches.

When several clocks arrive at the clock pin of a latch, as shown by the circuit example in [Figure 15-28](#), potentially distinct throughpaths can be created. Exclusive clock groups are honored by preventing some of the potential throughpaths.

Figure 15-28 Circuit With Multiple Clocks at Latches



In [Figure 15-28](#), if no clock groups are set, the following paths are valid from F1/clk to F4/D:

- F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_d) -> F4/D
- F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_e) -> F4/D
- F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_d) -> F4/D
- F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_e) -> F4/D

Setting the following clock groups reduces the number of valid paths:

```
set_clock_groups -exclusive -group {clk_b} -group {clk_d}
set_clock_groups -exclusive -group {clk_c} -group {clk_e}
```

The following valid paths remain:

- F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_e) -> F4/D
- F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_d) -> F4/D

Using Clock-to-Clock False Path Exceptions for Setup Only

For clocks that are exclusive, use the `set_clock_groups` command.

If you want to avoid setup checks between two clocks, but you also want to keep the hold checks, use the `set_false_path` exception for a partial solution:

```
set_false_path -from [get_clocks clkA] -to [get_clocks clkB] -setup
```

Single-segment paths launched from clkA and captured by clkB are affected by the exception. Throughpaths are not affected unless there is a segment in the throughpath that satisfies the exception. For example, if the launch clock of the path is clkA, and the first intermediate latch is clocked by clkB, the exception is applied to the throughpath. However, for throughpaths launched by clkA and captured by clkB, but have an intermediate latch clocked by other clocks, the exception is not applied.

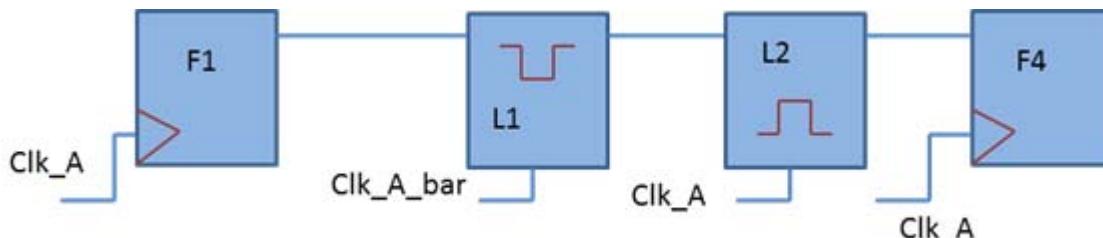
Specification of Exceptions on Throughpaths

Exception specifications that use multiple path specifiers should only use specifiers that are satisfied within one path segment. It is incorrect to specify exceptions with specifiers that are not satisfied in one segment.

For the circuit example in [Figure 15-29](#), the following exceptions are incorrectly specified and have no effect on throughpaths:

```
set_multicycle_path 2 -from F1/clk -through L1/Q
set_multicycle_path 2 -from F1/clk -to F4/D
```

Figure 15-29 Incorrectly Specified Exceptions on Throughpaths



Exceptions that are incorrectly specified have no effect on timing; to list these exceptions, use the `report_exceptions -ignored` command.

Reporting Paths Through Latches

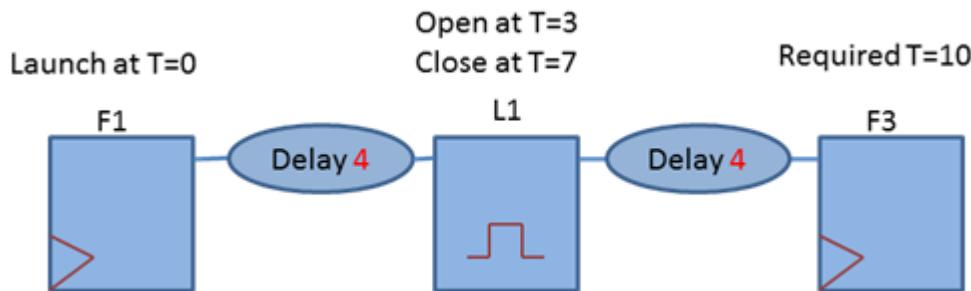
In circuits with latches, the transitive slack provides an estimate of the needed timing improvement for paths through a certain pin. To find the critical path that determines the transitive slack at a specified pin, use this command:

```
report_timing -through pin_name
```

To improve the slack at a specified pin, you can choose any location on the critical path to perform optimization. If you want to optimize power, the same report indicates the timing path that limits sizing operations at the specified pin.

The report for throughpaths includes a description of each intermediate transparency window. See the report in [Example 15-2](#), which corresponds to the circuit in [Figure 15-30](#).

Figure 15-30 Slack Example With Borrowing



Pin	Arrival	Required	Slack
F1/clk	0	2	+2
L1/D	4 (path from F1/clk)	6 (from downstream; local required time is 7)	+2
L1/Q	4 (path from F1/clk)	6	+2
F3/D	8 (path from F1/clk)	10	+2

Example 15-2 Timing Report of Paths Through Latches

```
pt_shell> report_timing
*****
Report : timing
    -path full
    -delay max
...
*****
Startpoint: F1/clk (rising edge-triggered flip-flop clocked by CLK)
Endpoint: F3/clk (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max
Point           Incr      Path
-----
clock CLK (rise edge)    0.00    0.00
clock network delay (ideal) 0.00    0.00
F1/clk (flop)          0.00    0.00 r
F1/Q (flop)            0.00    0.00 r
delay1/z (delay)       9.00    9.00 f
L1/D (latch)          0.00    9.00 f
-----
Transparency Window #1 (missed)
clock CLK (rise edge)      3.00
clock network delay (ideal) 0.02    3.02
Transparency max open edge 3.02
clock CLK (fall edge)      7.00
clock network delay (ideal) 0.03    7.03
library setup time        -0.01    7.02
inter-clock uncertainty   -0.02    7.00
Transparency max close edge 7.00
L1/D (latch)              9.00
Path recovery             -2.00    7.00
-----
L1/D (latch)          0.00    7.00 f
L1/Q (latch)            0.00    7.00 f
delay2/z (delay)       2.00    9.00 r
F3/D (flop)             0.00    9.00 r
data arrival time       9.00
clock CLK (rise edge)    10.00   10.00
clock network delay (ideal) 0.00    10.00
F3/clk (flop)           10.00   10.00 r
library setup time       0.00    10.00
data required time       10.00
-----
slack (MET)              1.00
-----
normalization delay     10.00
normalized slack         0.10
```

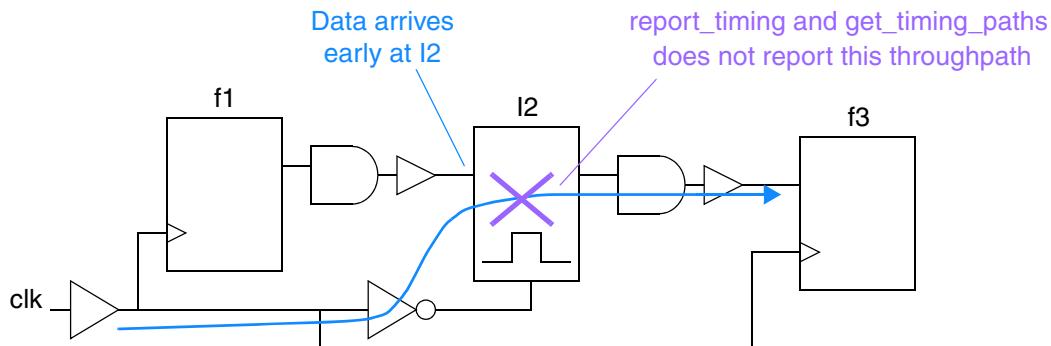
The last section of the report describes the normalized slack data. For more information about normalized slack, see [Normalized Slack Analysis](#).

Filtering Paths That Arrive Early at Intermediate Latches

By default, the `report_timing` command reports throughpaths that arrive before the open edge of intermediate latches along the path. To filter out these throughpaths, set the `timing_report_skip_early_paths_at_intermediate_latches` variable to `true`.

[Figure 15-31](#) shows an example of filtering a path that arrives early.

*Figure 15-31 Filtering Throughpaths With the
timing_report_skip_early_paths_at_intermediate_latches Variable*



Reporting Through Loop-Breaker Latches

The PrimeTime ADV tool considers downstream timing when calculating the required time at a loop-breaker latch. This occurs when you use the `report_timing` or `get_timing_paths` command without the `-to` option. For example,

```
report_timing -through $latch_loop_breaker
report_timing -from $startpt -through $latch_loop_breaker
report_timing -from $startpt
```

Tracing Forward Through Loop-Breaker Latches

The PrimeTime ADV tool traces forward through loop-breaker latches when you use the `report_timing` command with the `-trace_latch_forward` option. This capability is similar to the `-trace_latch_borrow` option, which traces backward.

The trace begins from a primary path specified by the `-from` or `-through` options. Additional path segments are included in the timing report, tracing forward from the endpoint of the primary path segment. The trace is in the direction of the worst required time. [Figure 15-32](#) shows an example of reporting through loop-breaker latches.

Figure 15-32 Report of a Path Through a Loop-Breaker Latch

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (propagated)	60.00	60.00
fd_in2/CP (FD1)	0.00	60.00 r
fd_in2/Q (FD1)	1.58	61.58 f
b2/Z (B1I)	39.00 H	100.58 f
...		
ld1/D (LD1)	0.00	103.88 f
data arrival time		103.88
clock clk' (rise edge)	30.00	30.00
clock network delay (propagated)	22.00	52.00
clock reconvergence pessimism	44.00	96.00
ld1/G (LD1)		96.00 r
transparency open edge		96.00
time required through endpoint	98.00	
data required time	98.00	
data required time	98.00	
data arrival time	-103.88	
slack (VIOLATED)		-5.88

Calculating the Worst and Total Negative Slack

The following pin attributes help you calculate the local costing in your design:

- `max_rise_local_slack`
- `max_fall_local_slack`

These attributes are defined on the D pins of latches and other timing endpoints. These attributes are not defined for combinational pins.

The worst negative slack (WNS) is the slack at the endpoint of the worst violating path. The path can be a single segment or throughpath. If there are no violating paths, the WNS is zero. The WNS is the minimum of all `max_rise_local_slack` and `max_fall_local_slack` attributes.

PrimeTime ADV calculates the total negative slack (TNS) with the following equation:

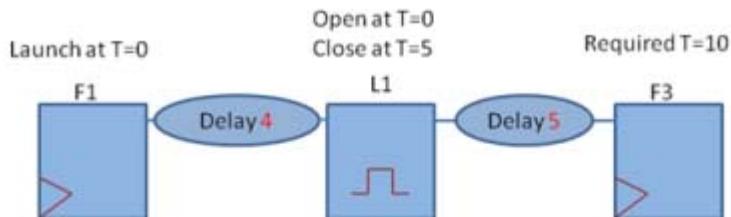
TNS = For all timing endpoints, summation of

$$\text{minimum}(0, \text{minimum}(\text{max_rise_local_slack}, \text{max_fall_local_slack}))$$

Normalized Slack Analysis

With advanced latch analysis, you can use the normalized slack to calculate the maximum frequency for a path. For example, [Figure 15-33](#) shows the normalized slack for three paths in a circuit.

Figure 15-33 Example of Normalized Slack



Path	Allowed Propagation delay	Slack	Normalized slack
F1/clk->L1/D	5	1	0.2
F1/clk->F3/D	10	1	0.1
L1/clk->F3/D	5	5	1

The PrimeTime ADV tool computes the allowed propagation delay for the path using ideal clock edges; the tool ignores setup time, uncertainty, and clock latency. The allowed propagation delay can be half-cycle, full-cycle, or multiple cycles; it can also be more complicated to compute when the launch and capture of a path exist in different clock domains.

To report and sort paths by normalized slack, use the `report_timing` command with the `-normalized_slack` option.

For information about running normalized slack analysis, see [Reporting Normalized Slack](#).

Finding Recovered Paths

A path is recovered when the arrival time at a latch is later than the closing edge of the transparency window minus the clock uncertainty and setup time. The `is_recovered` timing path attribute indicates that one of the latches in the timing path arrived after the closing edge, and there is a violation at the latch.

To find recovered paths, query the `is_recovered` attribute. This attribute is set to `true` for a path that was recovered at an intermediate latch.

16

Checking the Design and Analysis Setup

Before you begin a full analysis, it is a good idea to check your design characteristics and constraints. Paths that are incorrectly constrained might not appear in the violation reports, possibly causing you to overlook paths with violations.

To check your design, use the design reporting commands described in [Design Reports](#).

To check your constraints, use the following methods:

- [Basic Constraint Checking With the check_timing Command](#)
- [Detailed Constraint Checking and Debugging With PrimeTime GCA](#)

Basic Constraint Checking With the `check_timing` Command

To check for constraint problems such as undefined clocking, undefined input arrival times, and undefined output constraints, use the `check_timing` command. In addition, this command provides information about potential problems related to minimum clock separation (for master-slave clocking), ignored timing exceptions, combinational feedback loops, and latch fanout. You can correct unconstrained paths by adding constraints, such as `create_clock`, `set_input_delay`, and `set_output_delay`.

The following example shows a typical `check_timing` report:

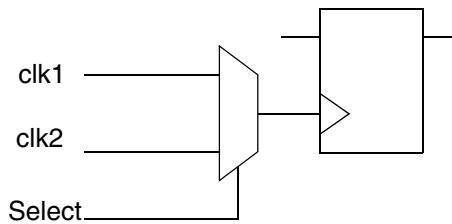
```
pt_shell> check_timing

Information: Checking 'no_clock'.
Warning: There are 4 register clock pins with no clock.
Information: Checking 'no_input_delay'.
Information: Checking 'unconstrained_endpoints'.
Information: Checking 'generic'.
Information: Checking 'latch_fanout'.
Warning: There are 2 level-sensitive latches which fanout to
themselves.
Information: Checking 'loops'.
Warning: There are 6 timing loops in the design.
Information: Checking 'generated_clocks'.
```

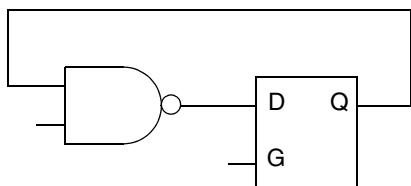
By default, the `check_timing` command performs several types of constraint checking and issues a summary report like the one shown in the preceding example. To get a detailed report, use the `-verbose` option. To add to or subtract from the default list of checks, use the `-include` or `-exclude` option of the `check_timing` command or set the `timing_check_defaults` variable to specify the list of checks for subsequent `check_timing` commands. For a list of all checks performed by the `check_timing` command, see the man page.

Warnings reported by the `check_timing` command do not necessarily indicate true design problems. To obtain more information, you can use a variety of report commands to get information about the characteristics of the design and the timing constraints that have been placed on the design.

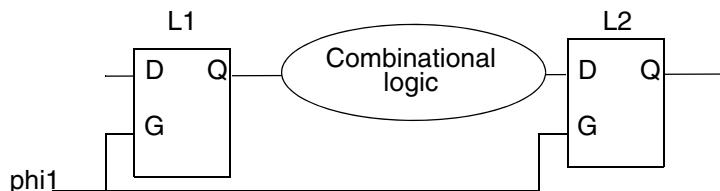
[Figure 16-1](#) shows register clock pins that are reached by multiple clock signals.

Figure 16-1 Multiple Clock Fanin

[Figure 16-2](#) shows an example of a self-looping latch.

Figure 16-2 Latch Self-Loop

[Figure 16-3](#) shows latch fanout to another latch of the same clock.

Figure 16-3 Latch Fanout to Another Latch of the Same Clock

Note:

If timing paths are unconstrained, the `check_timing` command only reports the unconstrained endpoints, not the unconstrained startpoints. Similarly, for paths constrained only by `set_max_delay`, `set_min_delay`, or both rather than `set_input_delay` and `set_output_delay`, the `check_timing` command only reports any unconstrained endpoints, not unconstrained startpoints.

The `check_timing` command allows interface logic models to be used at the chip-level. The `is_interface_logic_model` hierarchical instance attribute indicates whether an instance is an interface logic model. PrimeTime automatically sets this attribute to `true` when you create an interface logic model with the `create_ilm` command. The attribute allows the `check_timing` command to identify interface logic models and suppress false reporting of unconnected clock pins.

Detailed Constraint Checking and Debugging With PrimeTime GCA

To report incorrect constraints or other potential problems, use the detailed constraint checking and debugging capabilities of PrimeTime GCA. This add-on tool provides a broad set of predefined rules that cover problems in clocks, exceptions, cases, and design rules, such as blocked timing paths and missing clock definitions.

To run PrimeTime GCA from the PrimeTime shell,

1. Specify the PrimeTime GCA setup in the .synopsys_gca.setup file.
2. Specify the current design. For example,

```
pt_shell> current_design top
```

3. (Optional) Specify a PrimeTime GCA rule file that contains user-defined violation waivers and custom rules. For example,

```
pt_shell> set_app_var gca_setup_file ./top_rules.tcl
```

If you do not set this variable, the tool does not load a rule file.

4. Perform constraint analysis. For example,

```
pt_shell> check_constraints
```

```
GCA start-up loading .synopsys_gca.setup  
GCA loading design & constraints  
GCA loading ./top_rules.tcl  
GCA running analyze_design
```

5. To perform constraint analysis on another design, repeat steps 2 to 4. You can optionally specify a different rule file. For example,

```
pt_shell> current_design block  
pt_shell> set_app_var gca_setup_file ./block_rules.tcl  
pt_shell> check_constraints
```

```
GCA start-up loading .synopsys_gca.setup  
GCA loading design & constraints  
GCA loading ./block_rules.tcl  
GCA running analyze_design
```

Example 16-1 shows results of the `check_constraints` command.

Example 16-1 Constraint Checking Results Displayed in pt_shell

```
pt_shell> check_constraints

...
*****
Report : check_constraints
...
*****

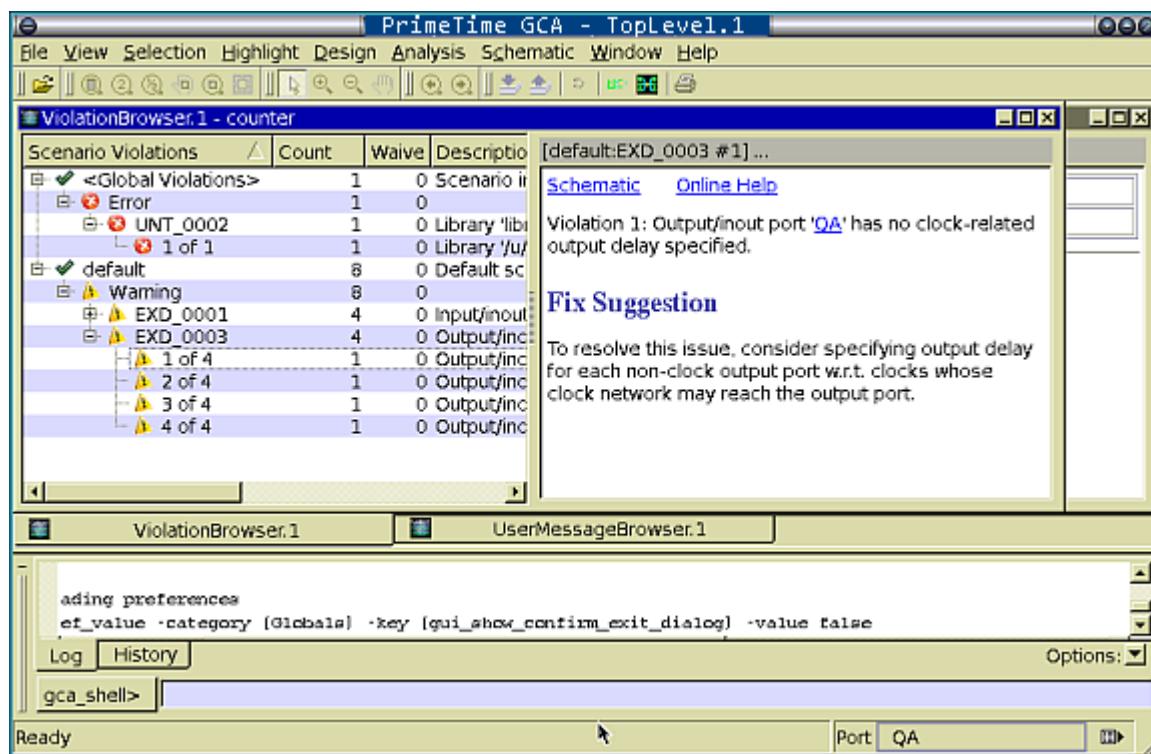


Scenario Violations Count Waived Description
-----
Design: counter
default          30   0    Default scenario violations
error            5    0
  CAP_0001        5    0    Output/inout port 'port' has zero or incomplete
                           capacitance values.
  1 of 5          0    0    Output/inout port 'QA' has zero or incomplete
                           capacitance values.
  2 of 5          0    0    Output/inout port 'QB' has zero or incomplete
                           capacitance values.

...
-----
Total Error Messages      6    0
Total Warning Messages    25   0
Total Info Messages       0    0
Maximum memory usage for this GCA flow: 25.27 MB
CPU usage for this GCA flow: 0 seconds
```

Alternatively, you can perform constraint analysis in the PrimeTime GUI by choosing Constraints > Constraint Checking. A separate PrimeTime GCA window displays the results, as shown in [Figure 16-4](#).

Figure 16-4 Constraint Checking Results Displayed in the GUI



To access more extensive constraint checking capabilities, run PrimeTime GCA as a standalone tool. For more information, see the *PrimeTime GCA User Guide*.

17

Reporting and Debugging Analysis Results

After you load and constrain the design and specify the conditions for analysis, you can perform a full static timing analysis and report any timing and design rule violations. You can also generate a wide range of reports that help you examine and debug violations.

To learn about the most commonly used reports, see

- [Global Timing Summary Report](#)
- [Path Timing Report](#)
- [Cover Design Report](#)
- [Quality of Results Report](#)
- [Constraint Reports](#)
- [Bottleneck Report](#)
- [Global Slack Report](#)
- [Analysis Coverage Report](#)
- [Design Reports](#)
- [Disabled Timing Arcs Report](#)
- [Clock Network Timing Report](#)

- Clock-Gating and Recovery/Removal Checks
- Timing Update Efficiency

Global Timing Summary Report

The `report_global_timing` command generates a top-level summary of the timing of the current design, including the following information:

- Worst negative slack (WNS) among all endpoints
- Total negative slack (TNS) of all endpoints
- Number of violating endpoints

The global timing report is organized into columns to show violations from input to register, from input to output, from register to register, and from register to output.

Use the `report_global_timing` command when you want only a top-level summary of the design timing. It provides this information more efficiently than using the `report_timing` command.

Here is a typical usage example:

```
pt_shell> report_global_timing -group [get_path_groups clk*]
...
Setup violations
Group: clk1
-----
      Total  reg->reg  reg->out  in->reg  in->out
-----
WNS      -2.14      0.00     -2.07     -2.14      0.00
TNS      -4.21      0.00     -2.07     -2.14      0.00
NUM        2          0          1          1          0
-----
Setup violations
Group: clk2
-----
      Total  reg->reg  reg->out  in->reg  in->out
-----
WNS      -1.87      -1.87      0.00      0.00      0.00
TNS      -1.87      -1.87      0.00      0.00      0.00
NUM        1          1          0          0          0
-----
```

If you do not use the `-group` option, the command generates a single report with all path groups combined.

To generate a top-level summary using path-based analysis, use the `report_global_timing` command with the `-pba_mode` option:

```
report_global_timing -pba_mode path | exhaustive
```

For example:

```
pt_shell> report_global_timing -pba_mode exhaustive

Report : global_timing
         -format { narrow }
         -pba_mode exhaustive

Setup violations
-----
      Total   reg->reg  in->reg  reg->out  in->out
-----
WNS     -1.15    0.00    -0.68    -1.15    0.00
TNS   -313.87    0.00   -13.11  -300.76    0.00
NUM      938        0       43      895        0
-----
```

No hold violations found.

If you run path-based analysis from a session saved after executing path-based `report_global_timing`, the results are generated much faster. To achieve this speedup, use the `report_global_timing` command with the same options that were used during the previous session.

Path Timing Report

You can use path timing reports to focus on particular timing violations and to determine the cause of a violation. By default, the `report_timing` command reports the path with the worst setup slack.

A path timing report provides detailed timing information about any number of requested paths. The level of detail that you want to see in the output can vary. For example, you can view this information:

- Gate levels in the logic path
- Incremental delay value of each gate level
- Sum of the total path delays
- Amount of slack in the path
- Source and destination clock name, latency, and uncertainty
- On-chip variation (OCV) with clock reconvergence pessimism removed

Using the report_timing Command

The `report_timing` command provides options to control reporting of the following:

- Number of paths
- Types of paths
- Amount of detail
- Startpoints, endpoints, and intermediate points along the path

The `-significant_digits` option of the `report_timing` command lets you specify the number of digits after the decimal point displayed for time values in the generated report. This option only controls the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

Example 1

The `report_timing` command reports the following:

```
pt_shell> report_timing

*****
Report : timing
Design : FP_SHR
*****


Operating Conditions:
Wire Loading Model Mode: top

Startpoint: a (input port)
Endpoint: c_d (output port)
Path Group: default
Path Type: max

Point           Incr      Path
-----
input external delay    10.00   10.00 r
a (in)            0.00    10.00 r
m1/Z (MUX21H)     1.00    11.00 r
u1/S (FA1)         1.00    12.00 r
c_d/Z (AN2)        1.00    13.00 r
c_d (out)          0.00    13.00 r
data arrival time          13.00

max_delay        15.00   15.00
output external delay -10.00    5.00
data required time          5.00
-----
data required time          5.00
data arrival time          -13.00
-----
slack (VIOLATED)       -8.00
```

Example 2

To show detailed information about the four worst paths, enter

```
pt_shell> report_timing -input_pins -max_paths 4
```

When `-max_paths` is set to any value larger than 1, the command only reports paths that have negative slack. To include positive-slack paths in multiple-paths reports, use the `-slack_lesser_than` option, as in the following example:

```
pt_shell> report_timing -slack_lesser_than 100 -max_paths 40
```

Example 3

You can use multiple `-through` options in a single command to specify paths that traverse multiple points. For example,

```
pt_shell> report_timing -from A1 -through B1 -through C1 \
           -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1.

```
pt_shell> report_timing -from A1 -through {B1 B2} \
           -through {C1 C2} -to D1
```

This means any path that starts at A1, passes through either B1 or B2, then passes through either C1 or C2, and ends at D1.

```
pt_shell> report_timing -from A1 -through {B1 C1} -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1. You cannot use more than one `-through` option in this syntax mode.

Example 4

To show the transition time and capacitance, use the following syntax:

```
pt_shell> report_timing -transition_time -capacitance
*****
Report : timing
    -path full
    -delay max
    -max_paths 1
    -transition_time
    -capacitance
Design : counter
...
*****
Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point           Cap      Trans     Incr      Path
-----
clock CLK (rise edge)      0.00    0.00
clock network delay (ideal) 0.00    0.00
ffa/CLK (DTC10)            0.00    0.00      0.00 r
ffa/Q (DTC10)              3.85    0.57    1.70      1.70 f
U7/Y (IV110)                6.59    1.32    0.84      2.55 r
U12/Y (NA310)               8.87    2.47    2.04      4.58 f
U17/Y (NA211)                4.87    1.01    1.35      5.94 f
U23/Y (IV120)               2.59    0.51    0.37      6.30 r
U15/Y (BF003)                 2.61    0.88    0.82      7.12 f
U16/Y (BF003)                 2.61    1.46    0.99      8.11 r
U10/Y (AN220)                  2.63    0.46    1.04      9.15 r
ffd/D (DTN10)                  0.46    0.00
data arrival time                   9.15
clock CLK (rise edge)             10.00   10.00
clock network delay (ideal)        0.00   10.00
ffd/CLK (DTN10)                  10.00   r
library setup time                -1.33   8.67
data required time                   8.67
data arrival time                   -9.15
-----
data required time                   8.67
data arrival time                   -9.15
-----
slack (VIOLATED)                  -0.48
```

Reporting Normalized Slack

The maximum frequency for a path depends on the propagation delay of the path and the number of clock cycles needed for the path. To determine the achievable frequency for an existing design, you need to find the timing paths that limit the frequency and focus on optimizing those paths.

PrimeTime helps you find the paths with the greatest effect on the clock frequency by calculating the *normalized slack*, a metric for timing paths. PrimeTime calculates the normalized slack for a path with the following equation:

$$[\text{Normalized slack}] = [\text{Path slack}] / [\text{Allowed propagation delay for path}]$$

For example, [Table 17-1](#) describes two paths with different slack values. Although Path B has worse slack than Path A, Path B has a smaller normalized slack and therefore a smaller effect on the clock frequency.

Table 17-1 A Path With Worse Slack Can Have a Smaller Normalized Slack

	Path A (Single-segment path)	Path B (Multicycle path)
Path slack	-10 ps	-20 ps
Allowed number of clock cycles	1	4
Clock period	500 ps	500 ps
Allowed propagation delay for path	500 ps	2000 ps
Normalized slack	-0.02	-0.01

Running Normalized Slack Analysis

To enable normalized slack analysis, set the `timing_enable_normalized_slack` variable to `true` before running timing analysis.

After you enable normalized slack analysis, the `report_timing` command reports normalized slack data. The tool can also sort paths by normalized slack, as well as find the paths with the worst normalized slack. To report the worst normalized slack, enter

```
pt_shell> report_timing -normalized_slack
```

In addition, the `normalized_slack` attribute on timing path objects represents the normalized slack computed by the `get_timing_paths` command. The `normalized_slack` attribute is available only if you enabled normalized slack before the last full timing update.

Setting Limits for Normalized Slack Analysis

Normalized slack analysis can potentially take a lot of runtime and memory. You can reduce the effect by limiting the allowed propagation delay along paths to a specified multiple of the clock period. To specify this multiple, set the `timing_max_normalization_cycles` variable:

```
pt_shell> set_app_var timing_max_normalization_cycles clock_cycles
```

The default is four clock cycles.

Using Normalized Slack to Adjust the Clock Period

When a path is launched and captured from clocks with varying periods, the normalized slack of the path indicates how much the clock period needs to change for the path to meet timing. If the launch and capture clocks are the same, the needed change in the clock period is

$$\Delta\text{Period} = -(normalized_slack) \times (period)$$

To calculate the needed change in the clock period so that all paths in the clock domain meet timing, use the worst normalized slack among the paths:

$$\Delta\text{Period} = -(worst_normalized_slack) \times (period)$$

For example, a design has the following characteristics:

- Initial clock period = 1000 ps
- Worst normalized slack = -0.2

To calculate the change in the period to meet timing, use this equation:

$$\Delta\text{Period} = -(-0.2) \times (1000 \text{ ps}) = 200 \text{ ps}$$

Therefore, the design in this example meets timing if you increase the clock period to 1200 ps.

The formulas in this section work for both positive and negative changes. If the design already meets timing, and all the normalized slack values are positive, the normalized slack can be used to compute the negative value of ΔPeriod , which you can use to speed up the clock.

Using the report_timing -exclude Option

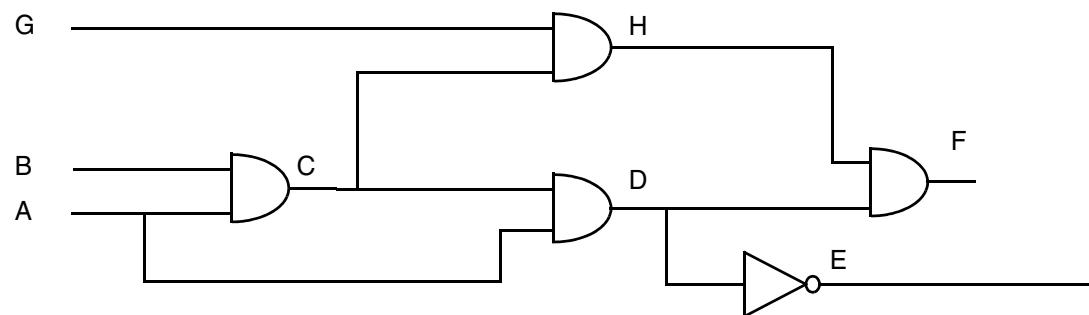
The `report_timing` command has the `-exclude`, `-rise_exclude`, and `-fall_exclude` options that allow you to skip paths that contain any excluded netlist objects. All of these options take a list of objects (pins, ports, nets, and cells). These options behave as follows:

- For pins and ports, they exclude all paths from, to, or through any excluded pin or port. This option does not apply to clock pins.
- For cells and nets, they exclude all paths from, to, or through all pins of any excluded cell or net.
- The `-exclude` option has higher precedence over the `-from`, `-to`, and `-through` options.
- The `-rise_exclude` and `-fall_exclude` options prune paths with rise and fall transition, respectively.
- When used with the `-trace_latch_borrow` option, the `-exclude` option does not apply to the borrowing path.
- When used with the `-path_type full_clock_expanded` or `full_clock` options, the `-exclude` option does not apply to the clock path.
- Multiple levels of the `-exclude` options are not supported. That is, you cannot exclude a particular path by specifying a series of the `-exclude` options.

Example of Using the -exclude Option

For example, assume that you had a design with the paths shown in [Figure 17-1](#), where C is both an endpoint and a startpoint.

Figure 17-1 Using the -exclude Option



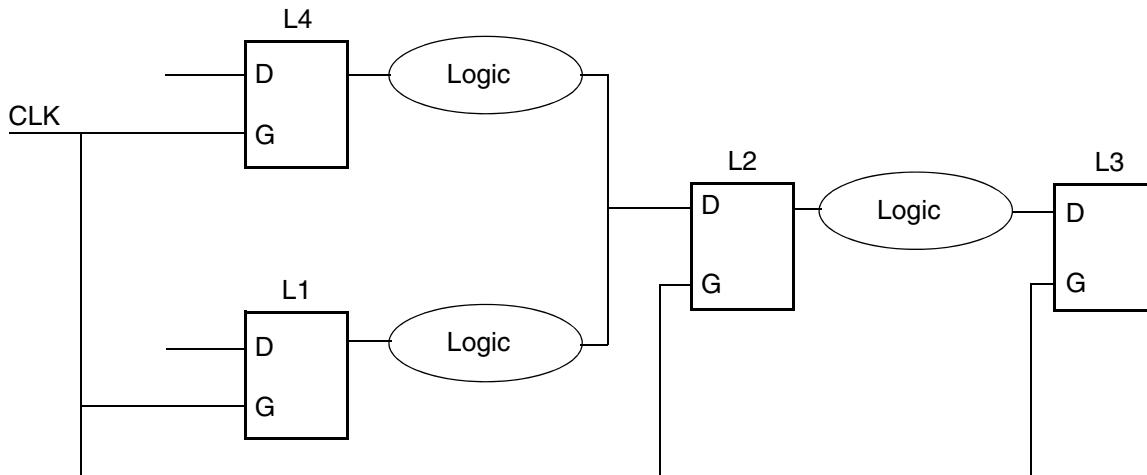
The following command reports three paths: A-->D-->E, A-->D-->F, and G-->H-->F.

```
pt_shell> report_timing -nworst 2 -max_paths 3 -exclude C
```

Example of Using the `-exclude` Option With the `-trace_latch_borrow` Option

The `-exclude` option only applies to data paths, not borrowing paths. For example, assume you have a circuit like the one shown in [Figure 17-2](#). In this circuit, LD2 borrows from both LD1 and LD4, but the path from LD1 is the worst path.

Figure 17-2 Using -exclude With -trace_latch_borrow



The following commands show the worst path starting from LD2 and still borrowing from LD1, because the `-exclude` option does not apply to borrowing paths:

```
pt_shell> report_timing -exclude LD1 -to LD3/D
pt_shell> report_timing -exclude LD1 -to LD3/D -trace_latch_borrow
```

Cover Design Report

To report or collect the worst path through each violating pin in the design, use the `report_timing` or `get_timing_paths` command with the `-cover_design` option. This capability overcomes the possibility of missing paths in cases where the `-nworst` option specifies a low value or where the `-start_end_pair` option misses reconvergent paths.

To run path-based analysis on the path set returned by the cover design, perform cover design reporting with the `-pba_mode path` option. This allows you to identify more path-based analysis violations earlier in the flow with quick turnaround time. For signoff, you should use exhaustive path-based analysis to ensure maximum pessimism removal.

You can use cover design reporting with signal integrity analysis, advanced on-chip variation (AOCV), parametric on-chip variation (POCV), and simultaneous multivoltage analysis (SMVA) flows. In distributed multi-scenario analysis (DMSA), you can perform cover design reporting at the slave processes, but you cannot perform merged reporting at the master process.

Quality of Results Report

The quality of results (QoR) report provides an overview of the quality of your design, including

- Length of the worst paths (shown as "Critical Path Length" in the report, which represents the arrival time of the signal at the path's endpoint)
- Total negative slack
- Minimum delay and hold
- Detailed design rule constraints summary

To report the quality of results, use the `report_qor` command.

Note:

The `report_qor` command does not provide a merged distributed multi-scenario analysis (DMSA) report. Running the command on the DMSA master results in an error message. If you use DMSA, use the `report_qor` Tcl procedure, which is described in SolvNet article 0008602, “[A report_qor Tcl Procedure for PrimeTime and PrimeTime SI, Including Distributed Multi-Scenario Analysis](#).”

Constraint Reports

The `report_constraint` command summarizes the constraint violations, including the amount by which a constraint is violated or met and the design object that is the worst violator. PrimeTime can report the maximum area of a design and certain timing constraints. It can also verify whether the netlist meets specific pin limits.

Timing Constraints

There are several types of timing constraints, such as

- Maximum path delay and setup
- Minimum path delay and hold
- Recovery time, the minimum amount of time required between an asynchronous control signal (such as the asynchronous clear input of a flip-flop) going inactive and a subsequent active clock edge
- Removal time, the minimum amount of time required between a clock edge that occurs while an asynchronous input is active and the subsequent removal of the asserted asynchronous control signal

- Clock-gating setup and hold
- Minimum pulse width high or low at one or several clock pins in the network
- Minimum period at a clock pin
- Maximum skew between two clock pins of a cell

Design Rule Constraints

PrimeTime performs checks to ensure that the netlist meets the design rules defined by the ASIC vendor and specified in the design library. You can also specify design rule parameters in PrimeTime, using commands such as `set_max_capacitance`.

The `report_constraint` command checks the following design rules if they are defined in the library or design:

- Total net capacitance (minimum and maximum limits)
- Pin transition time (minimum and maximum limits)
- Sum of fanout load attributes on net (minimum and maximum limits)

Usually only maximum capacitance and maximum transition are considered important. You can set the design rule constraint separately for rising and falling transitions, capacitance, and different clock domains and data paths for these clock domains.

Generating a Default Report

A default report displays brief information about the worst evaluation for each constraint in the current design and the overall cost. To report the constraints of a design, use the `report_constraint` command. For example,

```
pt_shell> report_constraint
```

```
*****
Report : constraint
Design : TOP
*****
```

Weighted Group (max_delay/setup)	Cost	Weight	Cost
C1	1.50	1.00	1.50
C2	0.00	1.00	0.00
max_delay/setup			1.50

Weighted Group (min_delay/hold)	Cost	Weight	Cost
C1	0.00	1.00	0.00
C2	0.00	1.00	0.00
min_delay/hold			0.00

Constraint	Cost
max_delay/setup	1.50 (VIOLATED)
min_delay/hold	0.00

Reporting Violations

To report all instances of constraint violations, use the `report_constraint -all_violators` command, as shown in the following example.

Example 17-1 Reporting Constraint Violations

```
pt_shell> report_constraint -all_violators
*****
Report : constraint
        -all_violators
Design : TOP
*****
max_delay/setup      ('C1' group)

Endpoint            Slack
-----
OUT1                -1.50 (VIOLATED)
OUT2                -1.50 (VIOLATED)
```

To show detailed information about the violations, use the `report_constraint -verbose` command, as shown in the following example.

Example 17-2 Reporting Details of Constraint Violations

```
pt_shell> report_constraint -all_violators -verbose

*****
Report : constraint
    -all_violators
    -verbose
Design : TOP
*****
```

Point	Incr	Path
clock C1 (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
ff3/CP (FF)	0.00	0.00 r
ff3/Q (FF)	1.00	1.00 r
OUT1 (out)	0.00	1.00 r
data arrival time		1.00
clock C1 (rise edge)	3.00	3.00
clock network delay (ideal)	0.00	3.00
output external delay	-3.50	-0.50
data required time		-0.50

data required time		-0.50
data arrival time		-1.00

slack (VIOLATED)		-1.50

Point	Incr	Path
clock C2 (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
ff4/CP (FF)	0.00	0.00 r
ff4/Q (FF)	1.00	1.00 r
OUT2 (out)	0.00	1.00 r
data arrival time		1.00
clock C1 (rise edge)	3.00	3.00
clock network delay (ideal)	0.00	3.00
output external delay	-3.50	-0.50
data required time		-0.50

data required time	-0.50
data arrival time	-1.00
slack (VIOLATED)	-1.50

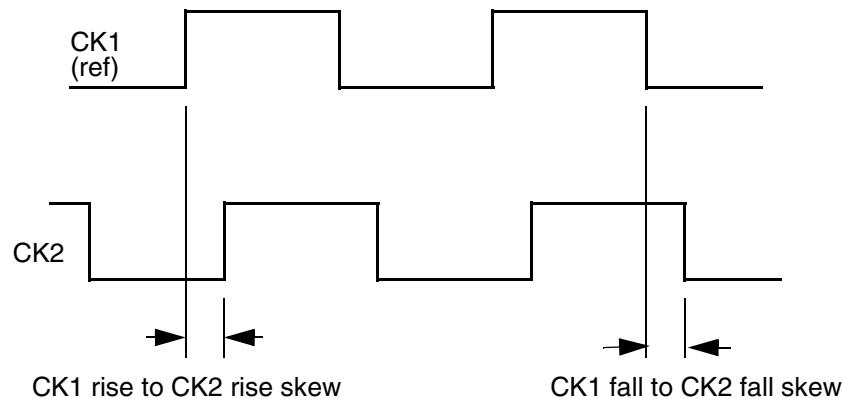
Maximum Skew Checks

You sometimes need skew timing checks on sequential devices with more than one clock signal. The skew constraint defines the maximum separation time allowed between two clock signals. You can describe skew constraints in the Synopsys library format (.lib). The following Library Compiler keywords describe skew constraints:

- `skew_rising`
- `skew_falling`

Using Library Compiler, you identify a skew constraint by defining the `timing_type` attribute as one of these two keywords in a timing group. The `related_pin` attribute specifies a reference clock pin. [Figure 17-3](#) describes a timing diagram for skew constraint.

Figure 17-3 Timing Diagram for Skew Constraint



When the timing type is `skew_rising`, the timing constraint interval is measured from the rising edge of the reference pin (specified in `related_pin`) to a transition edge of the constrained pin. The `intrinsic_rise` value is the maximum skew time between the reference pin rising to the constrained pin rising. The `intrinsic_fall` value is the maximum skew time between the reference pin rising to the constrained pin falling.

When the timing type is `skew_falling`, the timing constraint interval is measured from the falling edge of the reference pin (specified in `related_pin`) to a transition edge of the constrained pin. The `intrinsic_rise` value is the maximum skew time between the reference pin falling to the constrained pin rising. The `intrinsic_fall` value is the maximum skew time between the reference pin falling to the constrained pin falling. You can specify either the `intrinsic_rise` or `intrinsic_fall` value. You cannot specify both.

In PrimeTime, use the `report_constraint` command with the `-max_skew` option to report maximum skew checks. PrimeTime reports only maximum skew timing checks. The `-max_skew` option checks the maximum separation time allowed between two clock signals. The default displays all timing and design rule constraints.

PrimeTime calculates the actual skew for the specific skew check by taking the absolute value of the difference between delayed reference and constrained clock edges. The ideal reference clock edge is delayed by minimum clock latency to the reference pin; the ideal constrained clock edge is delayed by maximum clock latency to the constrained pin. Clock uncertainties are not considered for the skew timing checks. To report maximum skew for all violators, use this command:

Pin	Required	Actual	Slack
	Skew	Skew	
ff3/c (r->f)	0.15	5.47	-5.32 (VIOLATED)
ff2/c (f->f)	0.14	2.32	-2.18 (VIOLATED)
ff2/c (r->r)	0.12	1.18	-1.06 (VIOLATED)
ff4/c (f->f)	0.14	0.84	-0.70 (VIOLATED)
ff4/c (r->r)	0.12	0.42	-0.30 (VIOLATED)

To report detailed information, use this command:

```
pt_shell> report_constraint -max_skew -verbose
```

```
Constrained Pin: ff2/c
Reference Pin: ff2/cn
Check: max_skew
```

Point	Incr	Path
clock sclk (rise edge)	0.00	0.00
clock network delay (ideal)	1.40	1.40 r
ff2/cn	0.00	1.40
reference pin arrival time		1.40
clock mclk (rise edge)	0.00	0.00
clock network delay (ideal)	1.60	1.60 r
ff2/c	0.00	1.60 r
constrained pin arrival time		1.60
allowable skew	0.12	
actual skew	0.20	
slack (VIOLATED)	-0.08	

No-Change Timing Checks

Certain signals need no-change timing checks to make sure that they do not switch during the active interval of a periodic signal such as a clock. A no-change check is described in the library.

Performing a no-change check is equivalent to performing a setup check against the active edge transition and a hold check against the inactive edge transition. For a sequential element with an active-high clock, the no-change setup check is performed against the rising clock edge and the hold check is performed against the falling clock edge.

The `report_timing` and `report_constraint` commands report no-change checks as library no-change setup time and library no-change hold time.

In the following report, pay particular attention to the lines in bold.

```
Startpoint: EN1 (level-sensitive input port clocked by CLK)
Endpoint: UCKLENH (positive nochange timing check clocked
by CLK')
Path Group: CLK
Path Type: max
```

Point	Incr	Path
clock CLK (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
input external delay	1.00	1.00 r
EN1 (in)	0.00	1.00 r
U24/A (N1A)	0.00	1.00 r
U24/Z (N1A)	0.07	1.07 f
U25/A (N1B)	0.00	1.07 f
U25/Z (N1B)	0.08	1.15 r
U26/A (N1C)	0.00	1.15 r
U26/Z (N1C)	0.07	1.21 f
U27/A (N1D)	0.00	1.21 f
U27/Z (N1D)	0.04	1.25 r
UCKLENH/EN (LD1QC_HH)	0.00	1.25 r
data arrival time		1.25
clock CLK' (rise edge)	2.00	2.00
clock network delay (ideal)	0.00	2.00
UCKLENH/G (LD1QC_HH)	0.00	2.00 r
library nochange setup time	-0.32	1.68
data required time		1.68
data required time		1.68
data arrival time		-1.25
slack (MET)		0.43

Minimum Pulse Width Report

To report detailed information about the minimum pulse width for specified pins or ports, use the `report_min_pulse_width` command. By default, the report includes all objects with minimum pulse width constraints. To display only the violating minimum pulse width checks, use the `-all_violators` option, as shown in the following example:

```
pt_shell> set_timing_derate -late 2.0 -cell_check
pt_shell> report_min_pulse_width -all_violators \
          -path_type full_clock -derate
```

```
*****
Report : min pulse width
         -all_violators
         -path_type full_clock
         -derate
...
*****
Pin: reg3g/CP
Related clock: clk1
Check: sequential_clock_pulse_width
Point             Derate   Incr    Path
-----
clock clk1 (rise edge)           0.00    0.00
clock source latency            0.00    0.00
d (in)                          0.00    0.00 r
abufc/Z (BUFFHVTD1)            1.00    0.05
del3/Z (DELHVT4)               1.00    4.30   4.35 r
abufcd/Z (BUFFHVTD1)           1.00    0.07   4.42 r
reg3g/CP (DFCNHVTD1)           1.00    0.00   4.42 r
open edge clock latency        4.42

clock clk1 (fall edge)          2.97    2.97
clock source latency            0.00    2.97
d (in)                          0.00    2.97 f
abufc/Z (BUFFHVTD1)            1.00    0.06   3.03 f
del3/Z (DELHVT4)               1.00    3.90   6.93 f
abufcd/Z (BUFFHVTD1)           1.00    0.10   7.04 f
reg3g/CP (DFCNHVTD1)           1.00    0.00   7.04 f
close edge clock latency       7.04

required pulse width (high)    2.00    5.40
actual pulse width             2.61

slack (VIOLATED)                -2.79
```

Minimum Period Report

To report the details of minimum period calculation and constraint checking, use the `report_min_period` command.

By default, the report shows the required minimum period and the actual period:

```
pt_shell> report_min_period -significant_digits 5 -nosplit
*****
Report : min period
    -nosplit
    -path_type summary
...
*****
sequential_clock_min_period

      Required      Actual
Pin          min period   min period  Slack
-----
FF_C/CP (clk rise)  0.20000    0.19780   -0.00220  (VIOLATED)
FF_C/CP (clk fall)  0.20000    0.19800   -0.00200  (VIOLATED)
```

If there is a minimum period violation, you can debug the problem by reporting the full clock path:

```
pt_shell> report_min_period -path_type full_clock_expanded
*****
Report : min period
    -nosplit
    -path_type full_clock_expanded
...
*****
Pin: FF_C/CP
Related clock: clk
Check: sequential_clock_min_period

Point           Incr      Path
-----
clock clk (rise edge)      0.00000  0.00000
clock source latency       0.00000  0.00000
clk (in)                  0.00000 & 0.00000 r
CB1/Z (BUFFD1BWP)          0.12279 & 0.12279 r
CB2/Z (BUFFD1BWP)          0.08361 H 0.20640 r
CB3/Z (BUFFD1BWP)          0.13528 H 0.34168 r
FF_C/CP (DFD1BWP)          0.00001 & 0.34169 r
open edge clock latency    0.34169

clock clk (rise edge)      0.20000  0.20000
clock source latency       0.00000  0.20000
clk (in)                  0.00000 & 0.20000 r
CB1/Z (BUFFD1BWP)          0.12263 & 0.32263 r
CB2/Z (BUFFD1BWP)          0.08315 H 0.40578 r
CB3/Z (BUFFD1BWP)          0.13509 H 0.54088 r
FF_C/CP (DFD1BWP)          0.00001 & 0.54089 r
clock reconvergence pessimism 0.00060  0.54149
clock uncertainty          -0.00200 0.53949
close edge clock latency   0.53949

-----
required min period        0.20000
actual period              0.19780
-----
slack (VIOLATED)          -0.00220
```

Bottleneck Report

A bottleneck is a common point that contributes to multiple violations. Bottleneck analysis helps you identify the worst bottlenecks, determine the likelihood of being able to significantly improve a bottleneck, and make a change to the netlist (guide synthesis, try a gate sizing change, and so forth).

PrimeTime bottleneck analysis associates a bottleneck with leaf cells in a design. You can generate a bottleneck report to know which gates or nets to change to improve the overall

timing quality. Analyzing bottlenecks enables you to fix many paths with a single change (whether the change is logical or physical). In some cases, fixing many subcritical violating paths is preferable to fixing a few worst paths because a few violators might be resolved by place and route.

To report bottleneck information in PrimeTime, use one of the following methods:

- Graphically, by generating a bottleneck histogram (Reports > Histograms > Timing Bottlenecks in the PrimeTime GUI).
- In text form, by using the `report_bottleneck` command.

Note:

If you intend to use the `report_bottleneck` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update.

- By writing a Tcl program that retrieves, organizes, and reports bottleneck attributes. The Tcl procedure in [Example 17-4](#) is provided in `install_dir/auxx/pt/examples/tcl/bottleneck_utils.tcl`.

You can specify one of three cost types for the report: `path_count` (the default), `path_cost`, and `endpoint_cost`. Use the path type options as follows:

- To consider all violating paths equally, use `path_count`.
- To give greater weight to paths with larger violations, use `path_cost`.
- To consider only the slack of violating endpoints in the fanout of each cell (for faster analysis), use `endpoint_cost`.

The `report_bottleneck` command lists the leaf cells with the highest bottleneck cost.

To verify the result of bottleneck analysis, use the `report_timing` or `get_timing_paths` command.

Example 17-3 reports the 20 worst bottleneck cells in the current design based on the number of paths through the cells with slack less than 0.0.

Example 17-3 Bottleneck Report

```
pt_shell> report_bottleneck
```

```
*****
Report : bottleneck
    -max_cells 20
    -nworst_paths 100
...
*****
```

Bottleneck Cost = Number of violating paths through cell

Cell	Reference	Bottleneck Cost
ipxr/ir1/i2/i0/u3	DFF1A	39656.00
ipxr/ir1/U14	INV2	39654.00
ipxr/ir1/U16	INV8	39654.00
ipxr/ir1/i2/i0/u4	DFF1A	31806.00
ipxr/ir1/U15	BUF8B	31804.00
ipxr/U1712	MUX2I	23999.00
ipxr/U558	INV4	23999.00
ipxr/U1370	NAN4	22485.00
ipxr/x01	INV2	22485.00
dmac/BufEn	BUF3	22485.00
dmac/U574	INV	22485.00
ipxr/U1335	NAN6CH	21844.00
ipxr/U564	MUX2I	20074.00
ipxr/U1694	MUX2A	19936.00
ipxr/U1559	BUF2C	14785.00
ipxr/U1484	MUX2I	14252.00
ipxr/U1486	MUX2I	12468.00
SccMOD/U929	OR3	12135.00
ipxr/U1493	MUX2I	11248.00
dmac/BiuSMOD/U879	NAN2	10949.00

The following Tcl procedure validates the bottleneck cost for a cell to compare with `report_bottleneck` or the GUI bottleneck data. You can use the `print_report` command to list the paths.

Example 17-4 Tcl Procedure to Compare Bottleneck Values

```
proc verify_bottleneck_cell {cell slack_limit nworst
    print_report} {
    set through_pins [get_pins -of_object [get_cells $cell] -filter
"direction!=in"]
    set path_count 0
    set path_cost 0.0
    set fanout_endpoint_cost 0.0
    foreach_in_collection through_pins $through_pins {
        set paths [get_timing_paths -through $through_pins \
            -slack_lesser_than $slack_limit -nworst $nworst]
        foreach_in_collection path $paths {
            incr path_count
            set this_path_cost [expr 0.0 - [get_attribute $path slack]]
            set path_cost [expr $this_path_cost + $path_cost]
        }
        if {$print_report} {
            report_timing -through $through_pins \
                -slack_lesser_than $slack_limit -nworst $nworst
        }
        set paths [get_timing_paths -through $through_pins \
            -slack_lesser_than $slack_limit -max_paths 10000]
        foreach_in_collection path $paths {
            set this_path_cost [expr 0.0 - [get_attribute $path slack]]
            set fanout_endpoint_cost [expr $fanout_endpoint_cost + \
                $this_path_cost]
        }
    }
    echo -----
    echo "path_count = $path_count"
    echo "path_cost = $path_cost"
    echo "fanout_endpoint_cost = $fanout_endpoint_cost"
}
```

Global Slack Report

You can use the `report_global_slack` command to display the slack for a specified pin or port. All pins are reported by default, except pins of hierarchical cells and ports of the design.

Note:

Before using this command, time the design with the `timing_save_pin_arrival_and_slack` variable set to `true`. If the variable is `false` (the default), `report_global_slack` might have a longer runtime.

You can choose any combination from maximum or minimum and rise or fall to report a particular slack value. The `-max` and `-min` options are mutually exclusive as are the `-rise` and `-fall` options. Use the `object_list` option to list pins or ports. If no object list is provided, then the default is all pins.

To get a list of path endpoints that have setup timing violations, use this command:

```
pt_shell> get_attribute [current_design] violating_endpoints_max
{ "TOP/I_BLENDER/op2_reg[31]/D", ... }
```

The path endpoints are listed in order of increasing slack.

Similarly, to get a list of path endpoints that have hold timing violations, use this command:

```
pt_shell> get_attribute [current_design] violating_endpoints_min
{ "sd_DQ[0]", ... }
```

Analysis Coverage Report

You can report timing checks in the current design or current instance by using the `report_analysis_coverage` command. Generating information about timing checks is most critical for new designs.

Perform these checks right after you resolve errors found while using the `check_timing` command. Perform additional checks whenever significant changes are made to the design or the timing assertions. Analysis coverage checks are critical for sign-off. Follow this basic flow:

1. Run `link_design` and resolve link errors.
2. Run `check_timing` and resolve timing check errors.
3. Run `report_analysis_coverage` and resolve untested issues.
4. Perform the rest of the analysis.

The `report_analysis_coverage` command summarizes these checks:

- Setup
- Hold
- No-change
- Minimum period
- Recovery
- Removal
- Minimum pulse width

- Clock separation (master-slave)
- Clock-gating setup
- Clock-gating hold
- Output setup
- Output hold
- Maximum skew

The default report is a summary of checks organized by type. For each type of check, such as setup, the report shows the number and percentage of checks that meet constraints, violate constraints, and are untested. If there are no checks of a certain type, the report does not show that check type.

Use the report generated by the `report_analysis_coverage` command ensure that the analysis was complete. Static timing is considered exhaustive—all paths are checked. However, if the assertions are incomplete or if paths are disabled (using false paths, disabled arcs, case analysis, and so forth), some timing checks are not tested. You can use this report with the `check_timing` command to make sure the design and assertions are valid.

In some cases, you might want to use case analysis to analyze the design in different configurations. You can use the `report_analysis_coverage` command to determine which checks are untested in each configuration. If a check is untested in all configurations, you might want to do more analysis.

Use the `-status_details` option to show more information about the individual timing checks. The report shows all checks with the corresponding status. Untested checks have information about why they are untested, if the reason can be determined. You can use the `-exclude_untested` option to sort the list of reasons.

To display the summary report, use the `report_analysis_coverage` command. For example,

```
pt_shell> report_analysis_coverage
*****
Report : analysis_coverage
Design : counter
...
*****

```

Type of Check	Total	Met	Violated	Untested
setup	5	0 (0 %)	3 (60 %)	2 (40 %)
hold	5	3 (60 %)	0 (0 %)	2 (40 %)
All Checks	10	3 (30 %)	3 (30 %)	4 (40 %)

To display the detailed report of untested setup checks, use the `report_analysis_coverage` command with its options. For example,

```
pt_shell> report_analysis_coverage -status_details {untested} \
           -check_type {setup}
```

```
*****
Report : analysis_coverage
         -status_details {untested}
         -sort_by slack
         -check_type {setup }
Design : counter
...
*****
Type of Check      Total          Met          Violated        Untested
-----
setup              5              0 ( 0%)       3 ( 60%)       2 ( 40%)
-----
All Checks         5              0 ( 0%)       3 ( 60%)       2 ( 40%)
Constrained Pin      Related Pin    Check Type      Slack      Reason
-----  

ffd/CR            CP             setup        untested    no_clock
ffd/D             CP             setup        untested    no_clock
```

Design Reports

Before running a full timing analysis, check your design with the commands in [Table 17-2](#). After running timing analysis, these commands can also help you to debug violations in your design.

Table 17-2 Design Reporting Commands

Command	Description
<code>report_design</code>	Lists the attributes of the design, including the chosen operating conditions, wire load information, design rules, and derating factors.
<code>report_port</code>	Lists the ports and shows port information such as the direction, pin capacitance, wire capacitance, input delay, output delay, related clock, design rules, and wire load information.
<code>report_net</code>	Lists the nets and shows net information such as the fanin, fanout, capacitance, wire resistance, number of pins, net attributes, and connections.

Table 17-2 Design Reporting Commands (Continued)

Command	Description
report_cell	Lists the cells used and cell information such as the library name, input names, output names, net connections, cell area, and cell attributes.
report_hierarchy	Generates a hierarchical list of submodules and leaf-level cells in the current design.
report_reference	Lists hierarchical references, showing for each submodule the reference name, unit area, number of occurrences, total area, and cell attributes.
report_lib	Reports a specified library showing the time units of the library, capacitance units, wire load information, defined operating conditions, logic trip-point thresholds, and names of library cells.
report_clock	Reports the clocks defined for the design, showing for each clock the name, period, rise and fall times, and timing characteristics such as latency and uncertainty.
report_wire_load	Shows the wire load models set on the current design or on specified cells.
report_path_group	Reports the path groups. PrimeTime organizes timing paths into groups based on the conditions at the path endpoints.
report_bus	Reports information about buses (pins or ports) in the current instance or current design.
report_transitive_fanin	Reports the logic that fans in to specified pins, ports, or nets.
report_transitive_fanout	Reports the logic that fans out from specified pins, ports, or nets.
report_units	Shows the units of measurement for current, capacitance, resistance, time, and voltage used in the current design.

Disabled Timing Arcs Report

The `report_disable_timing` command reports the disabled timing arcs. You can disable timing arcs in several ways: by using the `set_disable_timing` command, propagating constants, case analysis, loop breaking, conditional arcs (arcs that have a `when` statement defined in the library), and default arcs (when the `timing_disable_cond_default_arcs` variable is set to `true`).

To produce a report about disabled timing arcs, use the `report_disable_timing` command. For example,

```
pt_shell> report_disable_timing

*****
Report : disable_timing
Design : TOP
...
*****

Flags :      c  case-analysis
             C  Conditional arc
             d  default conditional arc
             f  false net-arc
             l  loop breaking
             L  db inherited loop breaking
             m  mode
             p  propagated constant
             u  user-defined
Cell or Port   From   To    Sense          Flag     Reason
-----
DISP1[7]           p      DISP1[7] = 0
U1/U1/reg[0]   E      D    hold_clk_rise   c      D = 0
U1/U1/reg[0]   E      D    setup_clk_rise  c      D = 0
U1/U1/reg[0]   E      E    clock_pulse_width_high  c      D = 0
```

Clock Network Timing Report

The timing characteristics of the clock network are important in any high-performance design. To obtain information about the clock networks in a design, use the `report_clock_timing` command. This command reports the clock latency, transition time, and skew characteristics at specified clock pins of sequential elements in the network.

In the `report_clock_timing` command, you specify the type of report you want (latency, transition time, single-clock skew, interclock skew, or summary), the scope of the design to analyze, and any desired filtering or ordering options for the report. PrimeTime gathers the requested information and reports it in the specified order.

Latency and Transition Time Reporting

The information reported by the `report_clock_timing` command is based on the clock latency and transition times calculated by PrimeTime. This information is maintained for all clock pins of sequential devices (flip-flops and latches).

The clock latency at a particular pin depends on the following analysis conditions:

- Constraint type: setup or hold
- Timing path role: launch or capture
- Transition type: rise or fall

To restrict the scope of the `report_clock_timing` command, you can optionally specify the pins to analyze and the conditions at those pins. For example,

```
pt_shell> report_clock_timing -type latency -to U1/CP \
           -hold -capture -rise
```

In this example, PrimeTime reports the latency at pin U1/CP for a hold check, for data capture at the flip-flop, for a rising edge at the clock pin. If you specify neither `-rise` nor `-fall`, PrimeTime considers the device type, in conjunction with its launch or capture role, to determine the appropriate transition to report.

Using the `-to` option, you can selectively restrict the scope of the design checked for the report. For example,

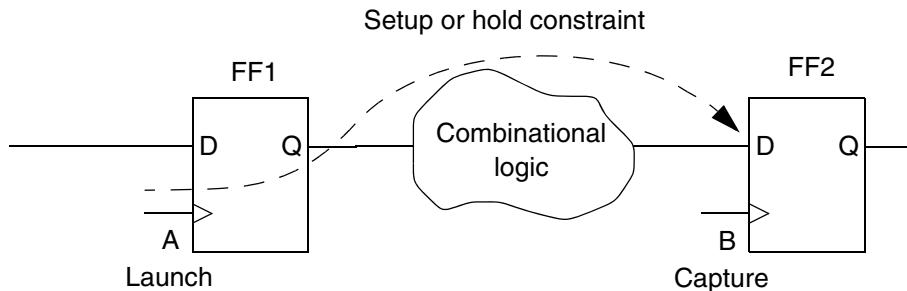
```
pt_shell> report_clock_timing -type latency \
           -to {U1/CP U7/CP} -hold -capture -rise
```

When a pin named in the “to” list is not a clock pin of a sequential element, PrimeTime replaces that pin with the set of clock pins in the transitive fanout of the named pin. (Here, “clock pin” means the clock pin of a flip-flop or the gate pin of a latch.)

Skew Reporting

To get a single-clock skew report, use `report_clock_timing -type skew`. This reports skews between pins clocked by the same clock. To report skews between pins clocked by different clocks as well as by the same clock, use `report_clock_timing -type interclock_skew`, as described in [Interclock Skew Reporting](#).

The skew between the clock pins of two different sequential devices is the difference between the latency values of the two pins, as shown in [Figure 17-4](#). PrimeTime calculates the latency at points A and B, and then subtracts latency at A from the latency at B to obtain the clock skew.

Figure 17-4 Clock Skew Calculation

$$\text{Skew} = (\text{latency at B}) - (\text{latency at A}) [- (\text{clock reconvergence pessimism})]$$

To determine the latency at the two pins for skew calculation, PrimeTime considers the following conditions at each pin:

- Type of sequential device involved: rising or falling edge-sensitive; or high or low level-sensitive
- Type of constraint: setup or hold (which determines the path type, transition type, and library)
- Role of the sequential device in the constraint relationship: launch or capture

If CRPR is enabled, PrimeTime takes it into account for the skew calculation. For information, see [Clock Reconvergence Pessimism Removal](#).

You can optionally restrict the scope of the report by specifying “from” and “to” pins. For example, to report the clock skew for a setup path between a negative-level-sensitive launch latch and a rising-edge-triggered capture flip-flop, use this command:

```
pt_shell> report_clock_timing -from latch/G -to ff/CP \
    -type skew -setup
```

PrimeTime reports the skew between a pair of sequential devices only if they can communicate by one or more data paths in the specified “from” and “to” direction. The existence of such a data path is sufficient for reporting. PrimeTime does not check to see whether the path has been declared false.

To find the worst skew between any pair of sequential devices, you can specify `-from` without `-to` or `-to` without `-from`. For the unspecified pins, PrimeTime uses the set of all sequential device clock pins that communicate with the specified pins in the specified direction. To restrict the clocks considered in the report, use the `-clock clock_list` option. To include clock uncertainty in the skew calculation, use the `-include_uncertainty_in_skew` option.

Like the `report_timing` command, the `report_clock_timing` command calculates skew based on the opening edge at the “to” device, even for a level-sensitive latch that allows time borrowing.

Interclock Skew Reporting

To report skew between pins clocked by different clocks as well as by the same clock, use the `report_clock_timing -type interclock_skew` command. An interclock skew report can help you get information, such as the following:

- Skew between pin A (clocked by CLK1) and pin B (clocked by CLK2)
- Worst local skew between all sequential devices clocked by CLK1 and all sequential devices clocked by CLK2
- Ten worst skews to all devices that communicate with pin A, irrespective of their domain

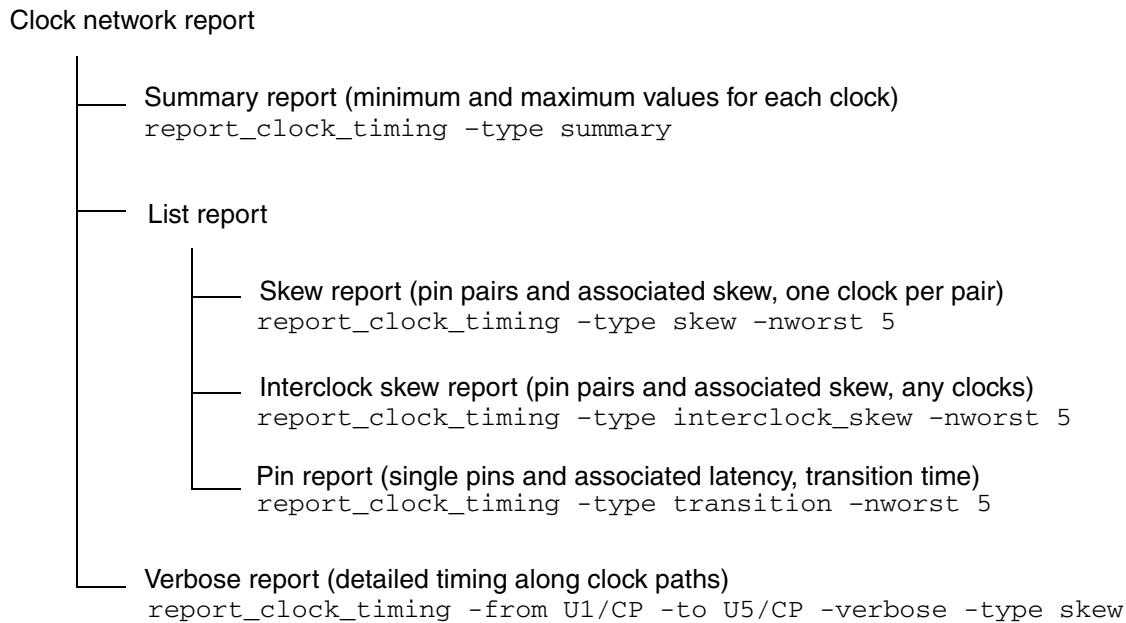
You can restrict the scope of the report by using the `-from_clock from_clock_list` option or the `-to_clock to_clock_list` option, or both options. Because of the potentially large number of clock pins that PrimeTime must analyze, it is a good idea to be as specific as possible when you specify an interclock skew report.

To display the names of the “from” clock and the “to” clock in the interclock skew report, use the `-show_clocks` option of the `report_clock_timing` command.

Clock Timing Reporting Options

The `report_clock_timing` command offers several different types of reports, as summarized in [Figure 17-5](#). The figure shows the report types, starting with the most general type at the top (summary report) and ending with the most specific type at the bottom (verbose report). The figure also shows an example of each type of command.

Figure 17-5 Clock Network Timing Report Types



Summary Report

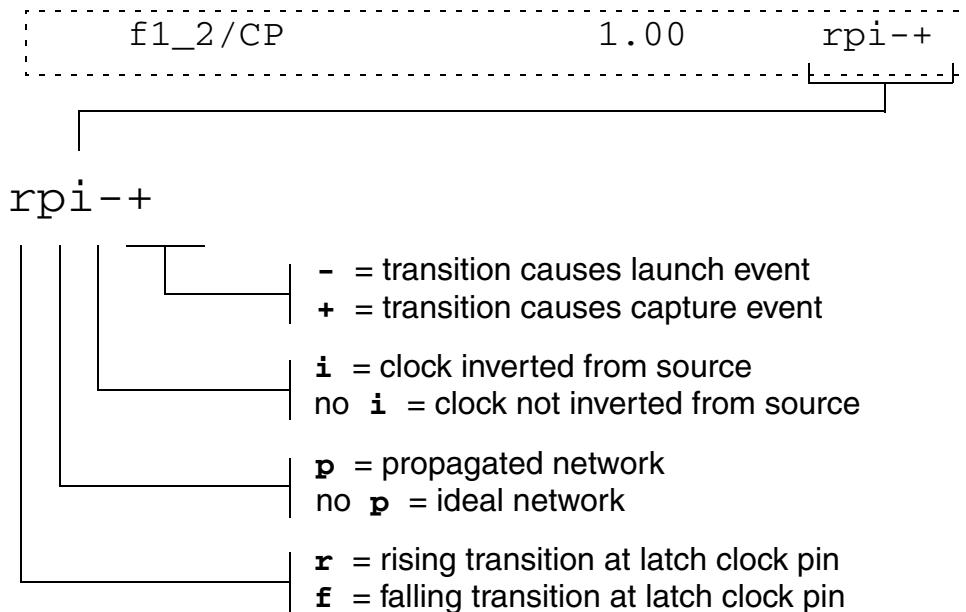
A summary report shows a list of the minimum and maximum latency, transition time, and clock skew values found in the requested scope of the clock network. For example,

```
pt_shell> report_clock_timing -type summary \
           -clock [get_clocks CLK1]
...
Clock: CLK1
-----
Maximum setup launch latency:
  f2_2/CP          6.11    rp-+
Minimum setup capture latency:
  f1_2/CP          1.00    rpi-+
Minimum hold launch latency:
  f1_2/CP          1.00    rpi-+
Maximum hold capture latency:
  f2_2/CP          6.11    rp-+
Maximum active transition:
  13_2/G           0.13    rpi-
Minimum active transition:
  12_3/G           0.00    rp-
Maximum setup skew:
  f2_2/CP          rp-+
  f2_1/CP          4.00    rp-+
Maximum hold skew:
  f3_3/CP          rpi-+
  f2_2/CP          3.01    rp-+
-----
```

The report shows the following information for each minimum and maximum latency, transition time, and skew value:

- Pins at which the minimum or maximum value occurred
- Minimum or maximum time value
- Conditions under which the minimum or maximum time value occurred

The string of characters in the rightmost column shows the conditions under which the minimum or maximum time value occurred, following the conventions shown in [Figure 17-6](#).

Figure 17-6 Latency, Transition Time, and Skew Condition Codes

To restrict the scope of the report, you can use the **-from** and **-to** options of the command. For example,

```
pt_shell> report_clock_timing -type summary -clock CLK1 \
    -from {A B C} -to {D E}
```

This command restricts the scope of the report to CLK1 latency and transition times at pins A through E, and restricts the scope of the skew report to CLK1 skew between the pin groups {A B C} and {D E}.

List Report

A list report provides latency, transition time, and skew information in greater detail than a summary report. You can have PrimeTime gather, filter, sort, and display a collection of clock pins according to a specified attribute of interest.

There are two types of lists you can generate: skew reports and pin reports. A skew report lists pin pairs and shows the skew value for each pair. A pin report lists individual pins and shows the transition time and latency values for each pin. The items are listed in order of skew, transition time, or latency, as specified by the options in the `report_clock_timing` command.

The following example shows a skew report:

```
pt_shell> report_clock_timing -clock CLK1 -type skew -setup \
    -nworst 3
```

The report shows the three largest skew values listed in order of decreasing skew, together with the latency at each pin and the clock reconvergence pessimism used in the skew calculation:

Clock: CLK1

Clock Pin	Latency	CRP	Skew
f2_2/CP	6.11		rp-+
f2_1/CP	2.01	-0.10	rp-+
l2_2/G	4.11		rp-
f1_2/CP	1.00	-0.10	rpi-+
f2_2/CP	6.11		rp-+
l3_3/G	3.01	-0.10	rp-

The rightmost column shows the conditions under which the reported values occurred at each corresponding pin, using the codes shown in [Figure 17-6](#).

An example of a command to generate an interclock skew report is as follows:

```
pt_shell> report_clock_timing -type interclock_skew \
    -nworst 12 -setup -include_uncertainty_in_skew
```

The report shows the 12 largest skew values between clock pins, whether clocked by the same clock or by different clocks. The report starts by showing the number of startpoint and endpoint pins and clocks under consideration. (Very large numbers at this point indicate a possibly long runtime to generate the rest of the report.) An example of an interclock skew report is:

```
*****
Report : clock timing
    -type interclock_skew
...
*****
Number of startpoint pins:      1023
Number of endpoint pins:       2496
Number of startpoint clocks:     4
Number of endpoint clocks:      6

Clock Pin          Latency   Uncert   Skew
-----
f2_2/CP           6.11      0.11     rp-+
f2_1/CP           2.01      0.11     fp-+
...

```

To show the names of the two clocks for each entry in the interclock skew report, use the `-show_clocks` option of the command.

An example of a command to generate a pin report is as follows:

```
pt_shell> report_clock_timing -clock CLK1 -type transition \
-nworst 5
```

The report shows the five largest transition times listed in decreasing order, together with the source, network, and total latency of the corresponding pins:

Clock	Pin	Source	Network	Total	Trans	
--- Latency ---						
13_2/G	0.10	4.00	4.10	0.13	rpi-	
f3_1/CP	0.11	3.00	3.11	0.12	rp-+	
12_1/G	0.10	4.00	4.10	0.10	rpi-	
f3_3/CP	0.10	3.00	3.10	0.08	rpi-+	
13_3/G	0.11	3.00	3.11	0.06	rp-	

The rightmost column shows the conditions under which the reported values occurred, using the codes shown in [Figure 17-6](#).

To get a list of the largest latency values rather than transition times, use `-type latency` instead of `-type transition`.

Verbose Path-Based Report

The most detailed type of clock network timing report is the verbose, path-based report. This type of report shows the calculation of skew, latency, and transition times along the clock path. For example,

```
pt_shell> report_clock_timing \
-from f3_3/CP -to f2_2/CP -verbose \
-hold -type skew -include_uncertainty_in_skew
```

This command produces a report showing the transition time, incremental delay, and total latency along the clock paths to the specified pins; and the clock skew between the two pins:

Clock: CLK1

```
Startpoint: f3_3 (rising edge-triggered flip-flop clocked
by CLK1')
Endpoint: f2_2 (rising edge-triggered flip-flop clocked
by CLK1)
```

Point	Trans	Incr	Path
<hr/>			
clock source latency		0.00	0.00
clk3 (in)	0.00	0.00	0.00 f
bf3_3_1/Z (B1I)	0.01	1.00 H	1.00 f
bf3_3_2/Z (B1I)	0.00	1.00 H	2.00 f
if3_3_1/Z (IVA)	0.04	1.00 H	3.00 r
f3_3/CP (FD1)	0.04	0.00	3.00 r
startpoint clock latency			3.00
<hr/>			
clock source latency		0.11	0.11
clk2 (in)	0.00	0.00	0.11 r
az_1/Z (B1I)	0.09	1.00 H	1.11 r
az_2/Z (B1I)	0.13	1.00 H	2.11 r
bf2_2_1/Z (B1I)	0.02	1.00 H	3.11 r
if2_2_1/Z (IVA)	0.44	1.00 H	4.11 f
bf2_2_2/Z (B1I)	0.01	1.00 H	5.11 f
if2_2_2/Z (IVA)	0.13	1.00 H	6.11 r
f2_2/CP (FD1)	0.13	0.00	6.11 r
endpoint clock latency			6.11
<hr/>			
endpoint clock latency			6.11
startpoint clock latency			-3.00
clock reconvergence pessimism			-0.10
inter-clock uncertainty			0.21
<hr/>			
skew			3.22

A command using the `-type latency` or `-type transition` option rather than the `-type skew` option produces a similar report, except that only one clock path is reported rather than two.

Limitations of Clock Network Reporting

The following limitations apply to the `report_clock_timing` command:

- When you ask for the skew between two clock pins, it calculates and reports the skew even if the path between the two clocks has been declared false.
 - If the “from” and “to” lists in the command contain a large number of pins, execution of the command can take a long time due to the large number of paths that must be checked, especially for interclock skew reports.
-

Using the `get_clock_network_objects` Command

The `get_clock_network_objects` command allows you to thoroughly view the clock network in your design. This command operates as a debugging tool similar to the such commands as the `get_cells` or `get_nets` command.

When you run the `get_clock_network_objects` command, PrimeTime returns a collection of clock network objects (including latches, flip-flops, and black box IPs driven by the clock network). Use the `-type` option to instruct PrimeTime to return clock objects you specified. These clock object can belong or relate to one or several clocks domains (specified using the `clock_list` option) or all clock domains. The `object_type` can be one of the following: `cell`, `register`, `net`, `pin`, `clock_gating_output`.

If you want PrimeTime to include clock-gating networks in the collection of clock networks, use the `-include_clock_gating_network` option.

The following example returns a collection of clock network pins of all clock domains, not including pins of the clock-gating networks.

```
pt_shell> get_clock_network_objects -type pin
```

Clock-Gating and Recovery/Removal Checks

By default, PrimeTime automatically determines clock gating and performs clock-gating setup and hold checks. To disable reporting clock-gating setup and hold checks, set the `timing_disable_clock_gating_checks` variable to `true`.

By default, PrimeTime performs recovery and removal checks. To disable reporting recovery and removal checks, set the `timing_disable_recovery_removal_checks` variable to `true`.

Timing Update Efficiency

PrimeTime has a built-in algorithm to efficiently update the timing of a design after its initial timing to accommodate a change in conditions. The algorithm reuses a portion of the computation done for the initial timing. For example, if you load and analyze a design using the `update_timing` or `report_timing` command, and then change the capacitance on a port with the `set_load` command, the subsequent `report_timing` command reuses results from the previous timing update, and only analyzes the timing changes resulting from the capacitance change. As a result, the second timing update takes much less time than the first one.

When any timing changes occur, timing is automatically updated by commands that need the information, such as the `report_timing` and `report_constraint` commands. You do not need to do a manual update.

Note:

The `update_timing -full` command causes a complete timing update and overrides the fast timing updates previously described. Avoid invoking `update_timing -full` unless it is necessary.

To update timing for the design manually, use the `update_timing` command. This command causes PrimeTime to recompute all the timing information.

The preceding algorithm trades off computational effort for memory usage. The `timing_update_effort` variable controls this tradeoff. You can set this variable to `low`, `medium` (the default), or `high`.

When one or more of the following commands causes a change to the timing of the design, PrimeTime updates only the timing for the changed data, resulting in a faster timing update.

- `set_load`
- `remove_capacitance`
- `set_resistance`
- `remove_resistance`
- `set_port_fanout_number`
- `remove_port_fanout_number`
- `set_input_transition`
- `set_driving_cell`
- `remove_driving_cell`
- `set_drive`

Status Messages During a Timing Update

PrimeTime can display messages during the timing update phase of an analysis, providing you with the current status of the update. For example, PrimeTime can issue messages when it is propagating constants, calculating delays, and calculating slack for paths. These messages can be very useful in debugging large designs and monitoring the progress of the analysis.

The `timing_update_status_level` variable controls the detail and number of progress messages that the timing update process issues. The allowed values are none, low, medium, or high. The default is none, indicating no intermediate status messages issued.

You can report the progress of the update timing explicitly by using the `update_timing` command and setting the `timing_update_status_level` variable to low, medium, or high, or for an implicit update use the `report_timing` command. The number of messages varies based on the variable setting.

If you want a detailed status of a timing update, set the `timing_update_status_level` to high before you begin your analysis. To show the messages during update timing, use the following syntax:

```
pt_shell> set timing_update_status_level high  
high
```

```
pt_shell> update_timing  
Information: Updating design - Started (UIITE-214)  
Information: Updating design - Propagating Constants (UIITE-214)  
Information: Updating design - Calculating delays (UIITE-214)  
Information: Updating design - Calculating delays 10%... (UIITE-214)  
Information: Updating design - Calculating delays 20%... (UIITE-214)  
Information: Updating design - Calculating delays 30%... (UIITE-214)  
Information: Updating design - Calculating delays 40%... (UIITE-214)  
Information: Updating design - Calculating delays 50%... (UIITE-214)  
Information: Updating design - Calculating delays 60%... (UIITE-214)  
Information: Updating design - Calculating delays 70%... (UIITE-214)  
Information: Updating design - Calculating delays 80%... (UIITE-214)  
Information: Updating design - Calculating delays 90%... (UIITE-214)  
Information: Updating design - Calculating delays 100%... (UIITE-214)  
Information: Updating design - Calculating slacks (UIITE-214)  
Information: Updating design - Calculating slacks (max type)  
    for group 'CLK3' (UIITE-214)  
Information: Updating design - Calculating slacks (min type)  
    for group 'CLK3' (UIITE-214)  
Information: Updating design - Calculating slacks (max type)  
    for group 'CLK4' (UIITE-214)  
Information: Updating design - Calculating slacks (min type)  
    for group 'CLK4' (UIITE-214)  
Information: Updating design - Completed (UIITE-214)
```

Some messages issued during the `read_parasitics`, `report_annotated_parasitics` -check, `read_sdf`, and an implicit or explicit `update_timing` commands have a default limit each time the command is invoked. If the number of messages exceeds this limit, a note stating that no further messages will be issued is added to the log file. In addition, a summary of the messages affected and the number suppressed is printed at the end of the PrimeTime session. If there are a large number of messages, the size of the log file is reduced. The `sh_message_limit` variable controls the default message limit and the `sh_limited_messages` variable controls the messages affected by this feature. Only messages issued during the commands mentioned are affected by the `sh_limited_messages` variable.

18

Graphical User Interface

The PrimeTime graphical user interface (GUI) allows you to visualize design data and analyze results by using analysis tools such as histograms, schematics, abstract clock graphs, waveform plots, and data tables. To learn about using the PrimeTime GUI, see

- [Opening and Closing the GUI](#)
- [GUI Windows](#)
- [Timing Analysis Flow With the GUI](#)
- [Viewing Timing Analysis Histograms](#)
- [Analyzing Collections of Timing Paths](#)
- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)
- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Clock Paths in a Schematic](#)
- [Viewing and Modifying Schematics](#)
- [Examining Timing Path Details in a Data Table](#)
- [Inspecting Timing Path Elements](#)
- [Viewing Object Attributes](#)

- Recalculating Timing Paths
- Analyzing Signal Integrity
- Analyzing Timing Paths from Multiple Scenarios
- Menu Command Reference

Opening and Closing the GUI

The PrimeTime GUI is a window- and menu-driven interface that operates in your X windows environment on UNIX or Linux. You can open the GUI when you start the PrimeTime tool or during a PrimeTime session.

When you open the PrimeTime GUI, it reads the GUI setup and preferences files and opens a new PrimeTime main window.

- The setup files perform basic setup tasks, such as initializing variables and declaring design libraries. The preference files set schematic and abstract clock graph view properties and global application preferences.
- The main window contains the menus, toolbars, and view windows you use to perform timing analysis and other analysis tasks.

You can open or close the GUI at any time during a session. For example, you can close the GUI when you need to perform time-consuming tasks or batch processes in `pt_shell`, and then you can reopen the GUI to perform visual analysis tasks.

Related Tasks

- [Opening the GUI](#)
 - [Closing the GUI](#)
-

Opening the GUI

Before you open the GUI, make sure that your DISPLAY environment variable is set to your UNIX display name. You can optionally set the DISPLAY variable at the same time that you start the session.

To start a PrimeTime session and open the GUI, enter the following command:

```
% pt_shell -gui
```

To set the DISPLAY environment variable when you start the PrimeTime session, include the `-display host_name` option, where `host_name` is the name of your UNIX display terminal. For example,

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

To open the GUI during a PrimeTime session, enter the `gui_start` command at the `pt_shell` prompt:

```
pt_shell> gui_start
```

See Also

- [Opening and Closing the GUI](#)
-

Closing the GUI

When you close the GUI, your designs remain loaded in memory and the command-line prompt remains active in the shell.

To close the GUI without exiting pt_shell,

- Choose File > Close GUI from the menu.

Alternatively, you can enter the `gui_stop` command on the console or pt_shell command line.

To close the GUI and exit the PrimeTime session entirely, choose File > Exit.

See Also

- [Opening and Closing the GUI](#)
-

GUI Windows

The PrimeTime GUI displays information in windows that provide a flexible working environment for performing different tasks. These windows operate independently in your X windows environment, but they share the same designs in memory, the same current timing information, and the global selection data in the tool.

The PrimeTime main window appears automatically when you open the GUI. It provides many of the view windows that you use to perform various types of analysis. Initially, the main window displays the hierarchy browser view window and the Console panel. You can open other views as needed, depending on the types of analysis that you need to perform.

In addition to the main window, the GUI provides the following application windows:

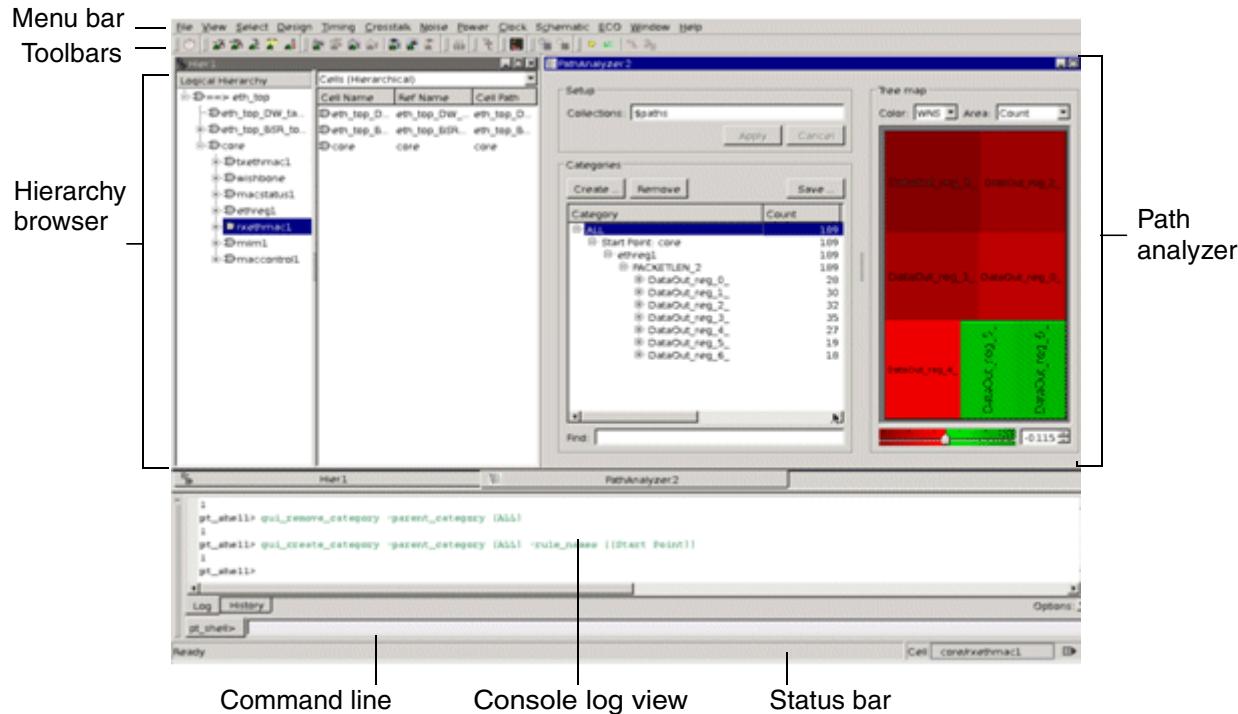
- The schematic window appears when you create an instance or clock schematic and provides the tools you need to visually examine the paths and design objects in the schematic.
- The abstract clock graph window appears when you create an abstract clock graph view and provides the tools you need to visually examine the clock paths in the graph.

Each application window contains a title bar, menus, toolbars and panels, a status bar, and the view windows that display specific design information. View windows and panels appear in the workspace area between the toolbars and the status bar. You can move, resize,

minimize, or maximize GUI windows by using the window management tools on your UNIX desktop.

Figure 18-1 shows an example of the main window containing the hierarchy browser, path analyzer, and console.

Figure 18-1 PrimeTime Main Window



The title bar, menu bar, and status bar are always visible. You can display or hide individual toolbars.

The menu bar 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.

Note:

If a window is not wide enough to display all the menu names on the menu bar, the window displays all the menu names that fit, from left to right, followed by an overflow button (). To access the other menus, click the overflow button.

The status bar displays information about command processing and the number and type of selected objects. 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. You can display the list of selected objects in the Selection List dialog box by clicking the  button at the right end of the status bar.

The workspace area between the toolbars and the status bar displays view windows and panels. View windows provide graphic or textual views of design information. Panels provide interactive tools for setting options or performing often used tasks. Some view windows and panels contain tabs with multiple views or pages. The *active view* is the view that has the mouse focus.

Toolbars are always attached to a window edge. You can attach individual panels to window edges or allow them to float inside or outside the workspace area. Most panels are associated with a particular type of view and operate on the active view. An exception is the console, which contains a command line and its own views.

You can adjust the sizes of view windows and panels for viewing purposes. You can also minimize individual view windows or maximize a view window to fill the workspace area. In addition, you can arrange the open view windows by tiling or cascading them within the workspace area.

You can open multiple instances of a GUI window and use them to compare views, or different design information within a view, side by side. The title bar displays the window instance name, which includes the unique number of the window. The windows are numbered sequentially throughout the session. You can configure the toolbars, view windows, and panels independently for each application window.

For more information about GUI windows, see

- [View Windows](#)
- [Toolbars and Panels](#)
- [Hierarchy Browser](#)
- [Console](#)
- [Object Selection](#)
- [Object Chooser](#)

Related Tasks

- [Viewing and Customizing Histograms](#)
- [Configuring Data Table Columns](#)
- [Filtering Tables and Lists](#)
- [Setting GUI Preferences](#)

See Also

- [Menu Command Reference](#)

View Windows

View windows are child windows that display design information inside a GUI window. When you open a new view window, the GUI displays a tab for the window at the bottom of the workspace area.

[Table 18-1](#) lists the types of view windows that you can open within an application window.

Table 18-1 Types of View Windows

Window type	Description
Hierarchy browser	Displays the design hierarchy and lists information about the objects (cells, ports, pins, or nets) in a block or hierarchical cell. (See Hierarchy Browser .)
Histograms	Display the distribution of design or analysis data in bar graph form, such as path slack or timing bottlenecks. The GUI provides histograms for timing analysis, signal integrity analysis in the PrimeTime SI tool, and power analysis in the PrimeTime PX tool. (See Analyzing Collections of Timing Paths , Analyzing Signal Integrity , and the <i>PrimeTime PX User Guide</i> .)
Path analyzer	Displays the design status for a collection of timing paths and categorizes them by attribute. (See Analyzing Collections of Timing Paths .)
Abstract clock graphs	Display the relationships between different clocks, such as primary and generated, clock convergence points, classes of clocked elements, and logic levels. (See Examining Clock Paths in an Abstract Clock Graph .)
Instance schematics	Display schematic representations of selected design instances (hierarchical cells), timing paths, or design objects. (See Examining Timing Paths and Design Logic in a Schematic .)
Clock analyzer	Provides a central place for running clock analysis tasks. (See Analyzing Clock Domains and Clock-to-Clock Relationships .)
Clock schematics	Display schematic representations of selected clock paths. (See Examining Clock Paths in a Schematic .)
Path data tables	Display timing path details in a data table view. (See Examining Timing Path Details in a Data Table .)

Table 18-1 Types of View Windows (Continued)

Window type	Description
Path inspector	Display various types of information about a timing path, including a schematic, a path element table, a delay profile, clocking information, and timing waveforms. (See Inspecting Timing Path Elements .)
Path and path pin comparison tables	Display normal paths or path pins with their recalculated path or path pin counterparts for side-by-side comparisons. (See Recalculating Timing Paths .)
Attribute data tables	Display object attribute values in a data table view for cells, ports, pins, nets, or clocks. (See Examining Object Attributes in a Data Table .)
Power analyzer	Provides a central place for running power analysis tasks. (See the <i>PrimeTime PX User Guide</i> .)
Power maps	Display power data, such as total power density or leakage power density, in a treemap that represents cells in the design hierarchy. (See the <i>PrimeTime PX User Guide</i> .)
Reports	Displays reports that you generate from a data table or by choosing a report command on the Design menu. (See Viewing Reports .)

If you click anywhere within a view window, the GUI highlights its title bar to indicate that it is the active view and can receive keyboard and mouse input. Where windows overlap, the active view window appears in front of all other windows. You can activate a view window by clicking its tab or by choosing its name on the Window menu. You can also cycle through the open view windows by choosing Window > Next or Window > Previous.

View windows that contain multiple views provide a tab for each view. When you open a view window that has multiple views, it displays a default view. To change to a different view, you click its tab.

See Also

- [Configuring View Windows](#)
- [View Settings Panel](#)

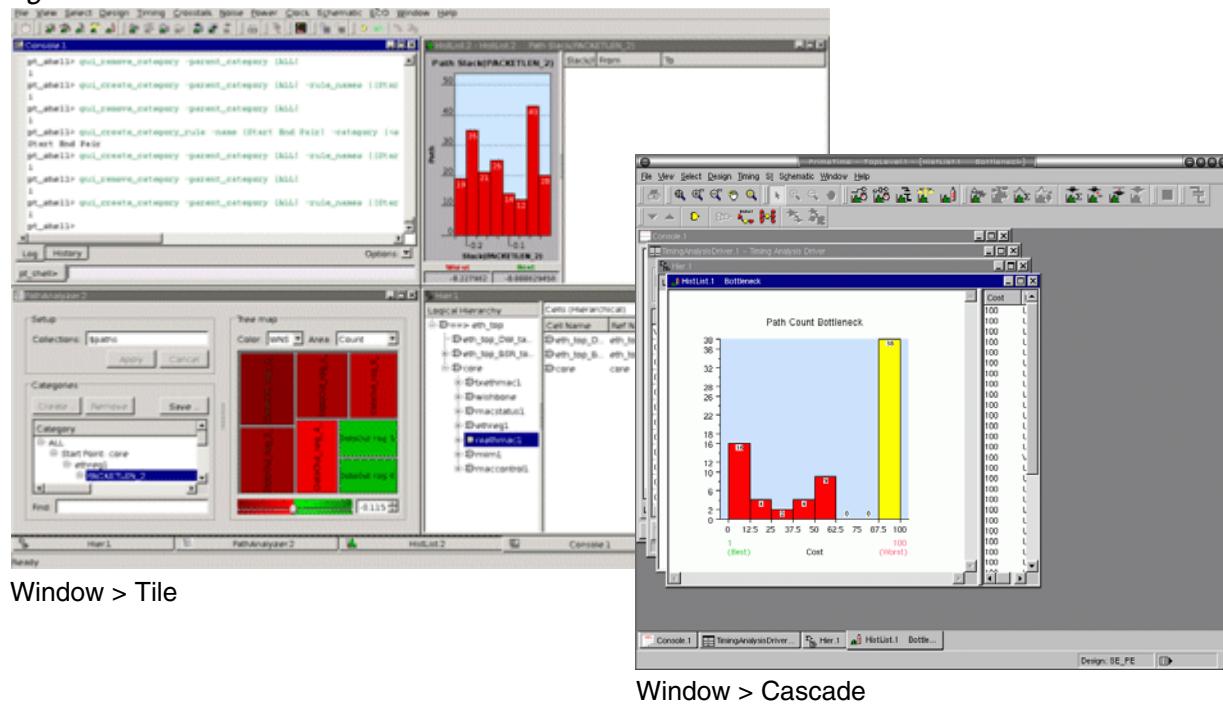
Configuring View Windows

You can move, resize, or organize view windows at any time.

- To move a view window, drag its title bar to the desired position.
- To resize or reshape a view window, move the pointer over an edge or corner until the pointer shape changes, and then drag the edge or corner until the window is the desired size or shape.
- To lay out all the view windows evenly like tiles, choose Window > Tile.
- To fan out the view windows like a deck of cards, choose Window > Cascade.

[Figure 18-2](#) shows examples of tiled and cascaded windows.

Figure 18-2 Tiled and Cascaded View Windows



Window > Tile

Window > Cascade

You can maximize, minimize, or restore any view window.

- To maximize a view window and make it fill the workspace area, right-click its title bar or its tab and choose Maximize.
- To minimize a view window and hide it from view, right-click its title bar or its tab and choose Minimize.
- To restore a minimized view window, double-click its tab or right-click the tab and choose Restore.

- To restore a maximized view window, right-click its tab and choose Restore.
- To close a view window, right-click its title bar or tab and choose Close.

You can close the active view window by choosing Window > Close View, or close all open views by choosing Window > Close All Views.

See Also

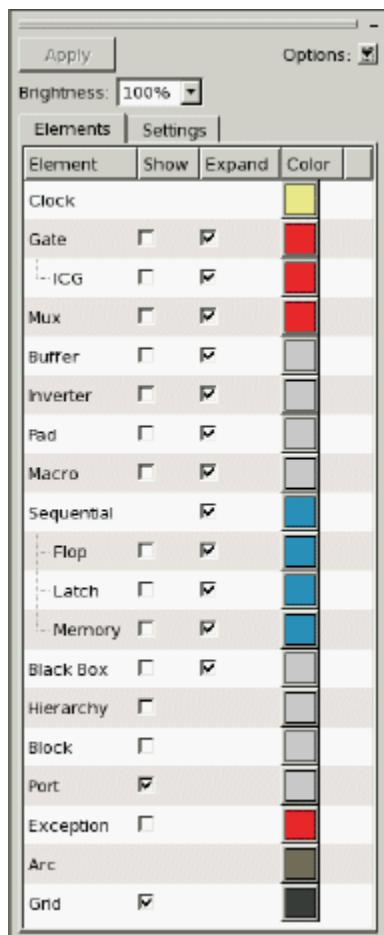
- [View Windows](#)

View Settings Panel

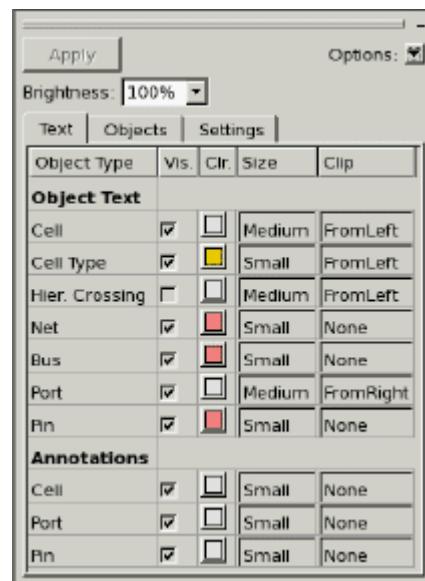
You can view and modify display settings for the active view by changing options on the View Settings panel. [Figure 18-3](#) shows the default appearance of the View Settings panel for abstract clock graph views and for schematic views.

Figure 18-3 View Settings Panel

Abstract clock graph view settings



Schematic view settings



To display or hide the View Settings panel,

- Choose View > Toolbars > View Settings.

The settings that are available depend on whether you are working in an abstract clock graph or a schematic view. By default, when you change an option, the option and the Apply button turn blue. You must click Apply before the change takes effect.

To operate the View Settings panel, you choose commands on the Options menu. The following choices are available:

- Preferences > Save to preferences

Saves the view settings across sessions.

- Preferences > Set from preferences

Applies previously saved preference settings to an existing view.

- Write settings script

Saves a Tcl script file that you can source to restore the settings. This is helpful if you plan to modify settings to see what the results are or if you plan to send the settings to another user.

- Auto apply

The changes are applied automatically and take effect immediately. Otherwise, the settings are implemented after clicking Apply.

- Cancel changes

Discards any changes that are made before clicking Apply.

- Defaults

Resets the view settings to the options available before you customized them. If this is the first time you are running the tool, the default settings are displayed.

See Also

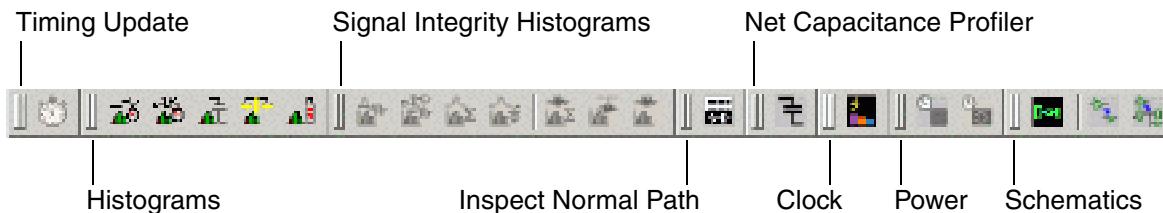
- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Clock Paths in a Schematic](#)

Toolbars and Panels

Toolbars provide quick access for the most often-used commands and interactive operations in view windows. Panels are enhanced toolbars that contain tools for setting options or working with design data.

Figure 18-4 shows the toolbars in the main window.

Figure 18-4 Main Window Toolbars



Buttons and other options that cannot be used in the active view or are not available for the current selection are dimmed (displayed in light gray). For example, if you select a path, the toolbar buttons for commands that operate on paths become available.

Related Tasks

- [Displaying or Hiding Toolbars and Panels](#)
- [Configuring Toolbars and Panels](#)

Displaying or Hiding Toolbars and Panels

Toolbars appear by default in a row at the top of the workspace area below the menu bar.

To display or hide a toolbar or panel,

- Choose View > Toolbars > *toolbar_name*, where *toolbar_name* is the name of the toolbar or panel you want to display or hide.

A check mark next to the name of a toolbar or panel on the Toolbars menu indicates that the toolbar or panel is visible.

See Also

- [Toolbars and Panels](#)
- [Configuring Toolbars and Panels](#)

Configuring Toolbars and Panels

Toolbars are always attached to a window edge. You can move panels to different locations inside or outside the GUI window. You can also resize panels and dock or undock a panel by attaching it to an edge of the window or separating it from the window edge so that it floats inside or outside the window.

- To move a toolbar or panel, drag its move bars (the raised bars near its top or left edge) to the desired location.

- To resize or reshape a panel, move the pointer over an edge or corner until the pointer shape changes, and then drag the edge or corner until the window is the desired size or shape.
 - To dock a panel, right-click its move bars and choose Dock > *window_edge*, where *window_edge* is an edge of the GUI window where the panel can be docked.
- To separate a panel from the window edge, right-click its move bars and choose Float.

If a window edge is not long enough to display all of the toolbars attached to it, the GUI displays the full toolbars that fit and shortened versions of the other toolbars. A shortened toolbar consists of a default toolbar button and an overflow button (>). To access the other toolbar buttons on a shortened toolbar, click the overflow button.

See Also

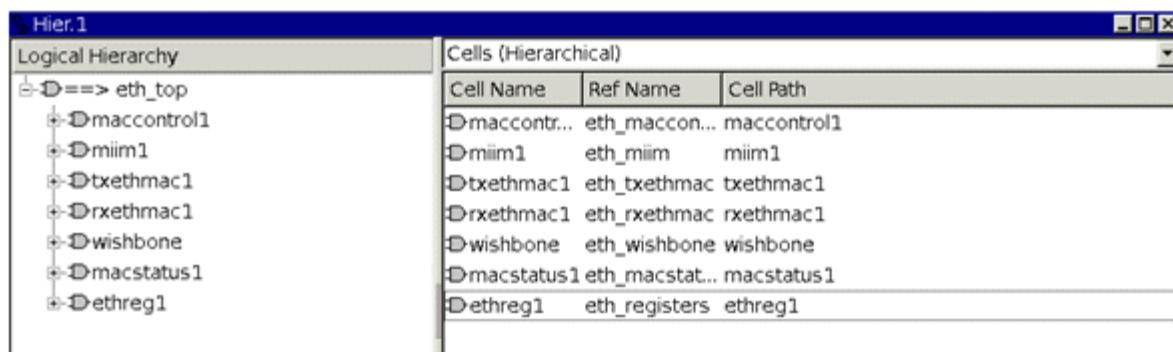
- [Toolbars and Panels](#)
- [Displaying or Hiding Toolbars and Panels](#)

Hierarchy Browser

The hierarchy browser lets you traverse the design hierarchy and select parts of the design for subsequent analysis. If a hierarchy browser view window is not already open, you can open one by choosing Design > New Hierarchy Browser View.

[Figure 18-5](#) shows an example of the hierarchy browser.

Figure 18-5 Hierarchy Browser View Window



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

When you select a cell in the left pane, the right pane shows the contents of that cell. You can select the types of objects to be listed: hierarchical cells, all cells, pins/ports, pins of child cells, or nets. Hierarchical cells are listed by default.

You can use the Up Arrow and Down Arrow keys to scroll up or down in the object list. If some of the text in a column is missing because the column is too narrow, you can hold the pointer over the column to display the text in an InfoTip.

You can sort the object list by the alphanumerical order of the contents in any column and adjust the widths of individual columns.

- To sort a column, click the column heading.
Click again if you want to reverse the order.
- To resize a column, drag the right edge of the column to the left or right.

You can also filter the object list based on a character string or regular expression that you define. For details, see [Filtering Tables and Lists](#).

When you select an object in the hierarchy browser, the object is also selected in other views, such as schematic views.

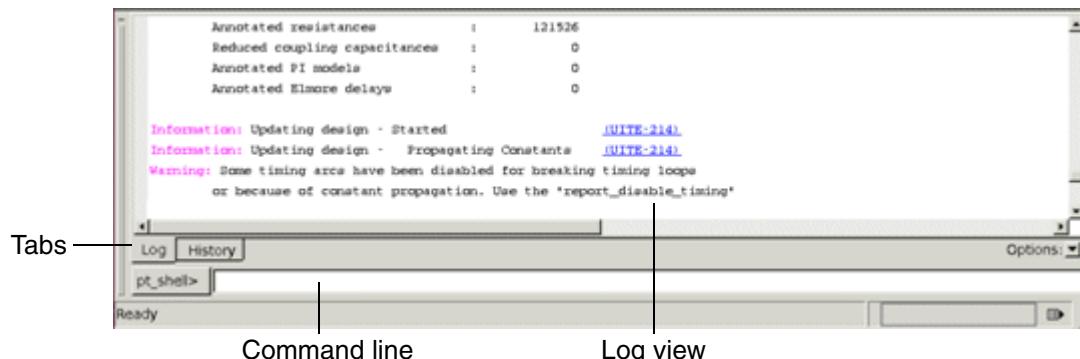
See Also

- [View Windows](#)
- [Filtering Tables and Lists](#)

Console

You can use the console to execute pt_shell commands, view the text-format PrimeTime response, and view the command history. An example of the console is shown in [Figure 18-6](#). The command line is the long box at the bottom, to the right of the pt_shell button. You can enter commands on the command line, just as you would at the pt_shell prompt in a terminal window.

Figure 18-6 Console



Operating the console mirrors what is entered and displayed in the pt_shell terminal window from which you opened the GUI. You can enter commands and view the response in either

the pt_shell terminal window or in the console. The commands entered and the PrimeTime response are displayed in both places. If you do not need the console, you can close it and just use the terminal window.

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

- Click the Log tab to display the pt_shell text-format response from the tool.

This is the same as what you see in a pt_shell terminal window. You can choose commands on the Options menu (on the right side of the console above the command line) to select and copy text, search for commands or messages, and save text in a file.

- Click the History tab to view a history of recently executed commands.

You can select commands to execute again or copy and paste the commands to create a script.

Note:

The GUI provides both a Console panel and a console view window. However, they have the same features.

See Also

- [Toolbars and Panels](#)

Object Selection

You can select objects for subsequent actions by clicking them in a view window or by choosing commands on the Select menu. To select multiple objects interactively, hold down the Ctrl key while you select. Otherwise, each new selection cancels the previous selection.

Objects that you select are displayed in white in graphic views, such as a schematic view or an abstract clock graph view, and in reverse video in list views, such as the hierarchy browser or a timing path table.

Objects that you select are selected in all views, not just the view in which you make the selection. For example, if a cell appears in two different schematic windows and is also listed in the hierarchy browser, selecting that cell anywhere (for example, in one schematic) causes all three occurrences to be highlighted.

For more information about selecting objects, see

- [Selecting Timing Paths](#)
- [Selecting Fanin or Fanout Logic](#)

See Also

- [Selecting or Highlighting Objects By Name](#)
- [Viewing the Current Selection](#)
- [Select Menu Commands](#)

Selecting Timing Paths

You can select specific timing paths or select one or more timing paths with the worst slack in the design. For example, to display a path profile, you first need to select the path. There are different ways to make this selection. Usually the simplest method is to go to the timing path table and click the path of interest. Another method is to use the Select Paths dialog box.

To select timing paths,

1. Choose Select > Paths From/Through/To.
2. Set options in the Select Paths dialog box.

You can specify the paths by using the From, Through, and To boxes or by setting options to define the criteria, such as the amount of slack.

3. Click OK.

See Also

- [Object Selection](#)
- [Selecting Fanin or Fanout Logic](#)

Selecting Fanin or Fanout Logic

You can select fanin or fanout logic for one or more selected objects. You can specify the start logic, the stop logic, and the number of logic levels to add.

To select objects in the fanin or fanout of specified objects,

1. Choose Select > Fanin/Fanout to open the Select Fanin/Fanout dialog box.
2. Set options as needed.
3. Click OK.

See Also

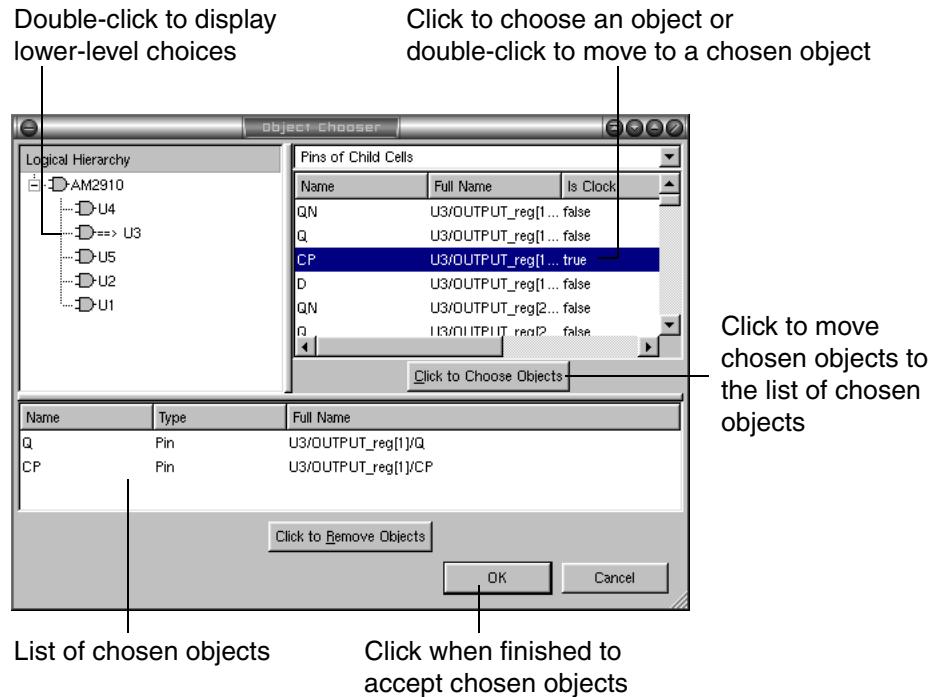
- [Object Selection](#)
- [Selecting Timing Paths](#)

Object Chooser

You can search for and select the objects you need to specify in a dialog box.

If you want to browse the design hierarchy to find the objects you need to specify, click the  button. This opens the Object Chooser dialog box shown in [Figure 18-7](#).

Figure 18-7 Object Chooser Dialog Box



See Also

- [GUI Windows](#)

Viewing Reports

Report views display report information. By default, when you generate a report from a data table or by choosing a report command on the Design menu, the GUI displays the report in a new report view.

You can use the Up Arrow and Down Arrow keys to scroll through the report. In reports that list object names, you can select an object by clicking its name (blue text) in the report view.

The buttons at the top of the report view let you clear the view (remove the report text), save the report in a text file, and display a report saved in a text file.

To remove a report from a report view,

- Click the  button.

To save a report displayed in a report view,

1. Click the  button to open the Save Report to File dialog box.
2. Select a file or enter a file name in the “File name” box.
If the file already exists, the tool overwrites it.

3. Click Save.

To open a report saved in a file,

1. Make sure the report view is empty.
2. Click the  button to open the Open Report to File dialog box.
3. Select the file or enter its name in the “File name” box.
4. Click Open.

To find text in a report,

1. Type a search string in the text box at the top of the report view.
2. Click the  button.

The tool highlights the first occurrence of the text that it finds in the report. You can repeat this step to find additional occurrences of the same text.

See Also

- [Viewing Timing Reports](#)
- [Viewing Object Reports](#)
- [Design Menu Commands](#)

Viewing and Customizing Histograms

You can use histograms to examine the distribution of data values, such as slack or capacitance, in bar-graph form within a range that you can set. The GUI provides histograms that you can use for timing analysis and for signal integrity analysis. You can also generate a custom histogram for any attribute-specified data that you can extract from the design.

When you create a histogram, you can set options to control the distribution of values and configure the histogram plot. You can set the number of histogram bins, the range of values

plotted in the histogram, the vertical scale, and the text labels displayed in the histogram. The default horizontal range is from the minimum value to the maximum value.

To customize the value range and appearance of a histogram,

1. Set options to control the distribution of data values in the histogram.

- To increase or decrease the maximum number of bins, select the “Number of bins” option and change the value in the “Number of bins” box. The default is 8.
- To limit the range of values in each bin, select the “Value range per bin” option and enter a value in the “Value range per bin” box.
- To control the range of values used to generate the histogram, enter a minimum value in the box to the left of the <= symbol, a maximum value in the box to the right of the => symbol, or both.
- To set the exact minimum value limit, select the “Lower bound strict” option.
- To set the exact maximum value limit, select the “Upper bound strict” option.

By default, the “Lower bound strict” and “Upper bound strict” options are deselected, and the histogram shows the smallest possible range that encompasses all the values between the minimum and maximum values you specify.

2. Set options to control the appearance of the histogram plot.

- To set the maximum value for the y-axis, enter a value in the “Y maximum” list. By default, the y-axis is automatically scaled to match the height of the tallest bin.
- To customize the histogram name, change the name in the “Histogram title” box. The default name depends on the type of data you select in the Column list.
- To customize the label for the x-axis, change the name in the “X-axis title” box. The default label depends on the type of data you select in the Column list.
- To customize the label for the y-axis, change the name in the “Y-axis title” box. The default label depends on the type of data you select in the Column list.

Each time you create a histogram, the GUI opens a new histogram view window. You can move, resize, minimize, or maximize a histogram window within the PrimeTime main window.

Histogram view windows are split into two panes. By default, the histogram bar graph appears in the left pane and the object table appears in the right pane. You can

- Hold the pointer over a bin in the bar graph to display information about its contents in an InfoTip

- Click a bin to display its contents (object names and slack or capacitance values) in the object list
- Select objects in the object list for further examination with other analysis tools, such as schematics or path data tables

You can adjust the relative sizes of the histogram plot and the object list by dragging the split bar between them. You can also set an application preference to display the object list below the bar graph when you open a new histogram.

To configure the histogram view window,

1. Choose View > Preferences to open the Application Preferences dialog box.
2. Select a histogram list position option.
The choices are Right and Bottom. The default is Right.
3. Click OK.

See Also

- [Viewing Timing Analysis Histograms](#)
- [Analyzing Signal Integrity](#)
- [Setting GUI Preferences](#)

Configuring Data Table Columns

You can configure a path data table or attribute data table by hiding or displaying individual columns and reordering the visible columns. If you create a table data histogram, the data that you want to view in the histogram must be visible in the table. If you save the data by exporting it to a CSV file, the tool saves only the data in the visible columns.

To hide, display, or reorganize columns in a data table,

1. Click the Columns button on the button bar below the data table.
The Show and Order Columns dialog box appears. Column names appear in the “Visible columns” list in the same order that they appear in the table from left to right.
2. Hide or display columns as needed:
 - To hide a column, select the column title in the “Visible columns” list and click the Left Arrow button.
 - To display a column, select the column title in the “Hidden columns” list and click the Right Arrow button.

3. Reorder the columns as needed.
 - To move a column up in the “Visible columns” list, select the column title and click the Up Arrow button.
 - To move a column down in the “Visible columns” list, select the column title and click the Down Arrow button.
4. Click OK.

See Also

- [Examining Object Attributes in a Data Table](#)
- [Examining Timing Path Details in a Data Table](#)
- [Filtering Tables and Lists](#)

Filtering Tables and Lists

Some view windows display design information in a table or list view. You can filter a table or list to display only the information in which you are interested.

To filter a table or list, you select the title of a column and specify a filter pattern of one or more characters. The filter displays only the rows for those items in the selected column that match the filter pattern (and hides all the other rows).

When you define the filter, you can

- Control whether the filter pattern can include a regular expression
- Control whether the filter pattern is case-sensitive
- Specify an inverse filter (hide the list items matched by the filter pattern and display all other items in the list)

A filter pattern uses wildcard characters as placeholders to represent alphanumeric characters. You can use the question mark (?) wildcard character to represent a single character. Use the asterisk (*) wildcard character to represent any number of consecutive characters (including zero). For example, U200* finds all occurrences of cell names starting with U200 followed by any number of characters.

To filter the object list or table in the active window, view, or dialog box,

1. Right-click in the table or list view and choose Filter.
The Filter List dialog box appears.
2. Select a column heading in the Column list.

3. Enter a filter pattern in the Pattern box.
4. (Optional) If you want to prohibit wildcard characters in the filter pattern, deselect the “Enable regular expressions” option.
5. (Optional) Select or deselect the “Match case” option as needed.
6. Select a Filter option.
 - Select “On” to hide the names of objects that do not match the filter pattern.
 - Select “On inverse” to hide the names of objects that match the filter pattern.
7. Click OK.

To remove the filter from the object list or table in the active window, view, or dialog box,

1. Right-click in the table or list view and choose Filter.

The Filter List dialog box appears.

2. Select Off.
3. Click OK.

See Also

- [Configuring Data Table Columns](#)

Setting GUI Preferences

When you open the GUI, it loads GUI preferences from your preferences file. The default system preferences are set for optimal tool operation and work well for most designs. However, if necessary, you can change GUI preferences during the session. You can set preferences that control how text appears in GUI windows, command log and global default controls, and the maximum number of cells in a schematic.

To set GUI preferences,

1. Choose View > Preferences.
- The Application Preferences dialog box appears.
2. (Optional) If you need to restore the system default preferences, click the Restore System Defaults button.
 3. Select a category in the Categories tree.
- The page for that category appears.
4. Set options as needed.

5. (Optional) Repeat steps 3 and 4 if you need to set options in another category.
6. Click OK or Apply.

When you change preference settings, the GUI automatically saves the new settings in the preferences file named `.synopsys_pt_prefs.tcl` in your home directory. The next time you open the GUI, it loads the preferences from this file.

See Also

- [GUI Windows](#)

Timing Analysis Flow With the GUI

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

An analysis session might consist of the following tasks:

1. Read, constrain, and analyze the design.
2. Open the GUI.
3. Perform high-level timing analysis of the design.
 - Generate a slack histogram to get a high-level overview of the timing quality of the design and to identify specific items that contribute to the worst slack values, such as paths, path endpoints, cells, and nets.
 - Generate a bottleneck histogram to find points in the design that contribute to the largest number of multiple timing violations. The block at the timing bottleneck might benefit from context characterization followed by optimization in Design Compiler.
 - Examine collections of timing paths in the path analyzer and categorize them by available attributes to find the paths of interest.
 - Examine clock paths in an abstract clock graph to view the clock structures and understand clock constraints.
4. Perform in-depth (path-level) analysis by examining specific paths in detail.

A path-level analysis can include the following activities:

 - Examine a path slack or bottleneck histogram generated with `-from`, `-through`, or `-to` options to restrict the scope of the paths analyzed.

- View timing path details in a path data table, from which you can select paths and view them in a schematic or path inspector windows or generate reports.
- Examine individual path elements in the path inspector, which includes graphical path delay profiles, path element reports, and timing waveform diagrams.
- Examine clock domains in the clock analyzer, which allows you to analyze the clock-to-clock relationships in your design and drive the clock analysis.

In addition, you can view individual timing paths in an instance schematic and individual clock paths in a clock schematic at any time during the analysis flow. Instance schematics allow you to trace paths through the design and view information about the design objects (cells, pins, ports, and nets) along the path. Clock schematics allow you to traverse the clock hierarchy and view information about the objects along a path.

See Also

- [Viewing Timing Analysis Histograms](#)
- [Examining Timing Path Details in a Data Table](#)
- [Inspecting Timing Path Elements](#)
- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Viewing Timing Analysis Histograms

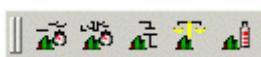
Histograms are focal points of visual timing analysis that allow you to view the overall timing performance of your design.

The GUI provides histograms that you can use to examine the following types of timing data:

- Timing slack for all endpoints
- Timing slack for specified paths
- Total net capacitance for all nets
- Design rule check values on all pins (maximum or minimum capacitance, fanout, or transition time)
- Bottleneck cost for all cells

You can generate a timing analysis histogram by clicking a button on the Histograms toolbar, shown in [Figure 18-8](#), or choosing *Timing > Histogram > Histogram Type*.

Figure 18-8 Histograms Toolbar



Note:

In the PrimeTime SI tool, you can display histograms of delta delay, bump voltages, and noise analysis results. For more information, see [Analyzing Signal Integrity](#).

For more information about viewing timing analysis histograms, see

- [Examining Endpoint Slack](#)
- [Examining Path Slack](#)
- [Examining Net Capacitance](#)
- [Examining Design Rule Slack](#)
- [Examining Timing Bottlenecks](#)

See Also

- [Analyzing Collections of Timing Paths](#)
- [Analyzing Signal Integrity](#)

Examining Endpoint Slack

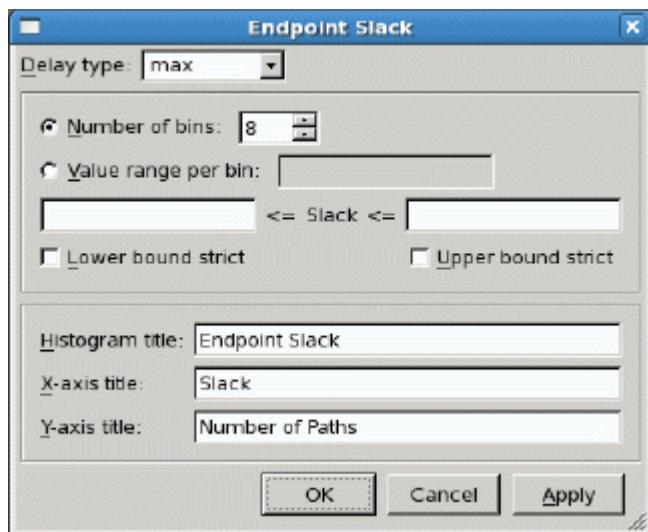
Use an endpoint slack histogram to view the distribution of worst slack values for all timing endpoints in the design. The slack distribution provides an overall picture of how close the design is to meeting requirements.

To generate an endpoint slack histogram,

1. Click the  button on the Histograms toolbar or choose Timing > Histogram > Endpoint Slack.

The Endpoint Slack dialog box appears as shown in [Figure 18-9](#).

Figure 18-9 Endpoint Slack Dialog Box



2. (Optional) Select a delay type option.

The default histogram is based on maximum delay times, which allows you to view the distribution of setup times. You can select a delay type based on minimum delay times to generate a histogram that displays the distribution of hold times. Alternatively, you can select a delay type based on maximum or minimum delay times on rising or falling clock edges.

3. (Optional) Set options to control the distribution of slack values and configure the histogram plot.
4. Click OK.

The tool uniformly divides the specified range, from minimum to maximum, into the specified number of bins, and then plots the number of slack values falling into each bin. [Figure 18-10](#) shows an example of an endpoint slack histogram.

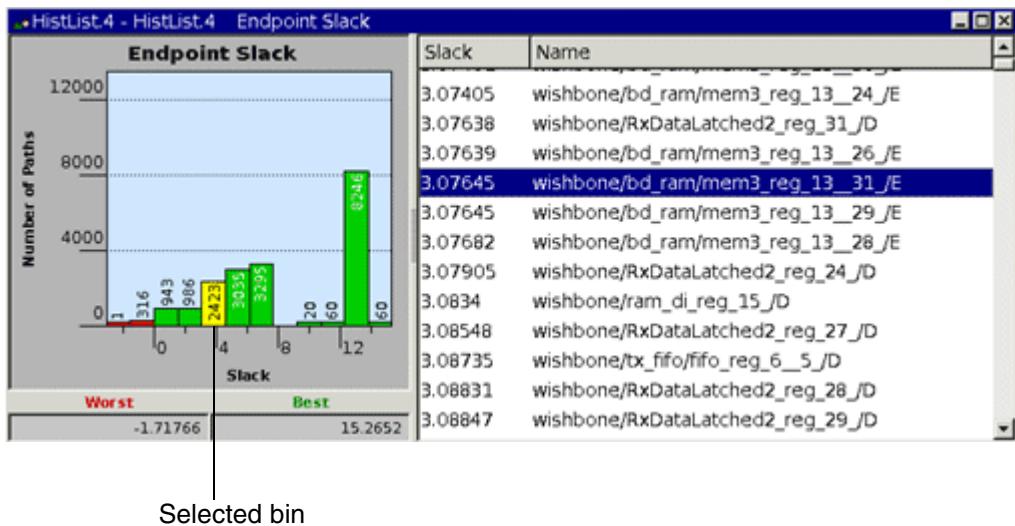
The numbers at the top of each bin indicate the total endpoints in the bin. Green bins (on the positive side of 0) contain endpoints in the design that met their constraints. Red bins (on the negative side of 0) contain endpoints that failed their constraints.

If you hold the pointer over a bin, an InfoTip displays the number of endpoints and the range of slack values in the bin. An InfoTip might read, for example,

Range: 0.08 to 0.496
Contents: 35

When you click a bin to select it, the bin turns yellow and the slack value and endpoint name for each pin appear in the list to the right of the histogram.

Figure 18-10 Endpoint Slack Histogram



You can select an endpoint in the list to view information about the worst path to the pin. Right-click and choose a command to select the path, generate a timing report for the path, or view the path in an instance schematic, a path inspector window, or a timing path table.

See Also

- [Examining Path Slack](#)
- [Viewing and Customizing Histograms](#)

Examining Path Slack

Use a path slack histogram to view the distribution of slack values for paths that you specify or for the paths with the worst in the design. The plot shows the slack values for all paths that meet the criteria you specify, and it can include the slack values of multiple paths that share a common endpoint.

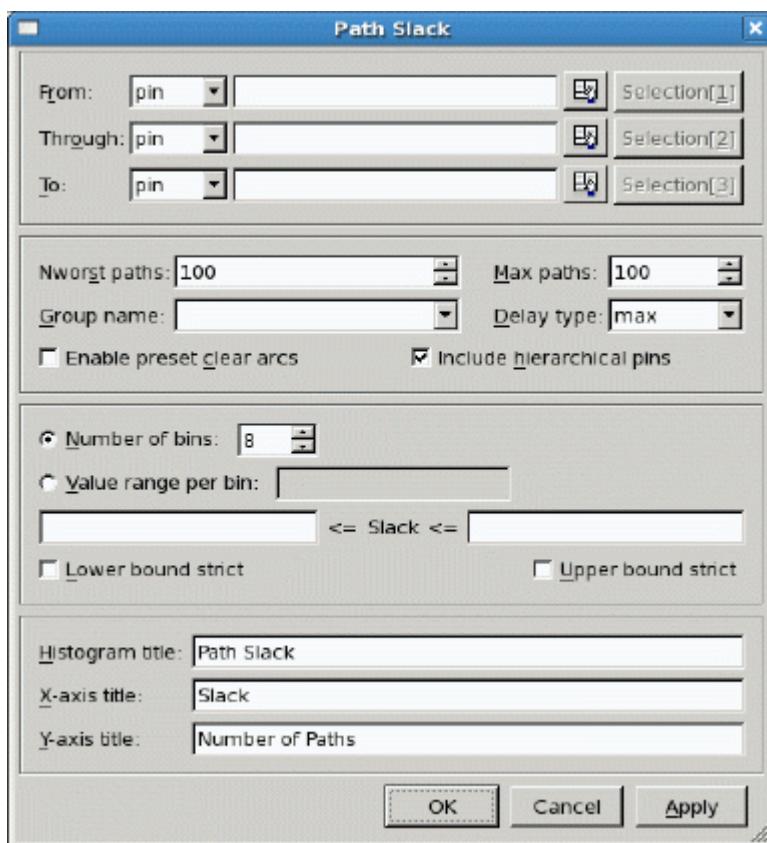
You can define the scope of the analysis by specifying a combination of one or more startpoints, throughpoints, or endpoints in the design. All paths that begin, pass through, or end on the respective points are analyzed and plotted in the histogram. Paths not within the specified scope are ignored for the analysis.

To generate a path slack histogram,

1. Click the button on the Histograms toolbar or choose Timing > Histogram > Path Slack.

The Path Slack dialog box appears as shown in [Figure 18-11](#).

Figure 18-11 Path Slack Dialog Box



2. (Optional) Specify the startpoints, throughpoints, or endpoints to generate a histogram for paths from, through, or to specific inputs, registers, or outputs.

You can specify the objects by selecting them or entering their names. For example, if you enter a single pin name in the From box (and leave the Through and To boxes blank), the tool finds all the paths that originate at the specified pin and plots the resulting slack values in the histogram.

To specify the currently selected pins, ports, nets, or clocks in the From, Through, or To box, select an object type option and click the Selection button.

For more information about using these options, see the man page for the `report_timing` command.

3. (Optional) Set options to select the delay type and control the distribution of slack values in the histogram.

The default histogram is based on maximum delay times, which allows you to view the distribution of setup times. You can select a delay type based on minimum delay times to generate a histogram that displays the distribution of hold times. Alternatively, you can

select a delay type based on maximum or minimum delay times on rising or falling clock edges.

You can restrict the number of data points included in the histogram plot by specifying the number of worst paths per endpoint (Nworst paths box), the total maximum number of path slacks to be reported (Max paths box), or both. You can also limit the histogram to the paths in a particular path group (Group name box).

For more information about these options, see the `report_timing` man page.

4. (Optional) Set options to control the distribution of slack values and configure the histogram plot.
5. Click OK.

The tool uniformly divides the specified range, from minimum to maximum, into the specified number of bins, and then plots the number of slack values falling into each bin. [Figure 18-12](#) shows an example of a path slack histogram.

The histogram plot shows the distribution of slack values for the paths included in the scope of the analysis. The numbers at the top of each bin indicate the total paths in the bin. Green bins (on the positive side of 0) contain paths in the design that met their constraints. Red bins (on the negative side of 0) contain paths that failed their constraints.

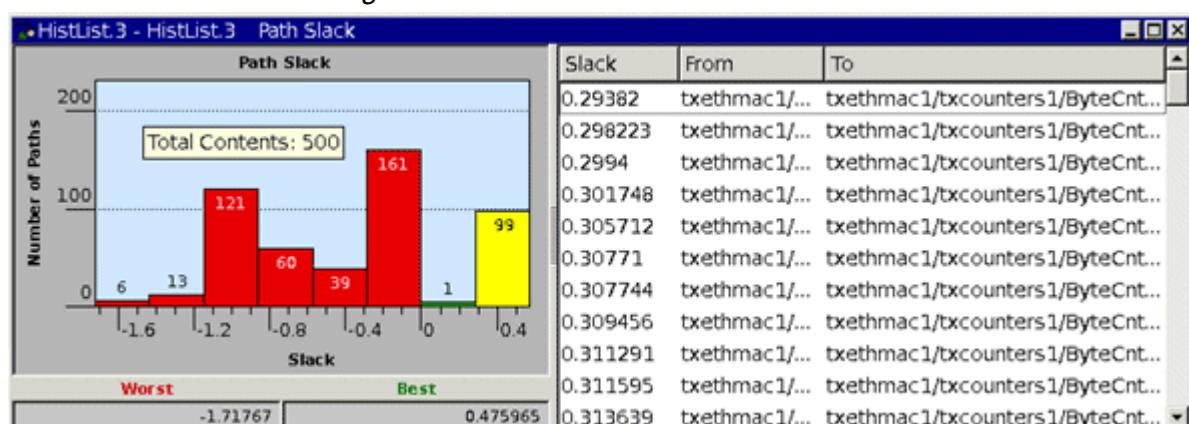
If you hold the pointer over a bin, an InfoTip displays the number of paths and the range of slack values in the bin. An InfoTip might read, for example,

Range: 0.08 to 0.496

Contents: 35

When you click a bin to select it, the bin turns yellow and the slack value, startpoint name, and endpoint name for each path appear in the list to the right of the histogram plot.

Figure 18-12 Path Slack Histogram



You can select paths in the list to view information about them. Right-click and choose a command to generate a timing report for the paths or view the paths in an instance schematic, a path inspector window, or a path data table.

See Also

- [Examining Endpoint Slack](#)
- [Examining Timing Bottlenecks](#)
- [Viewing and Customizing Histograms](#)

Examining Net Capacitance

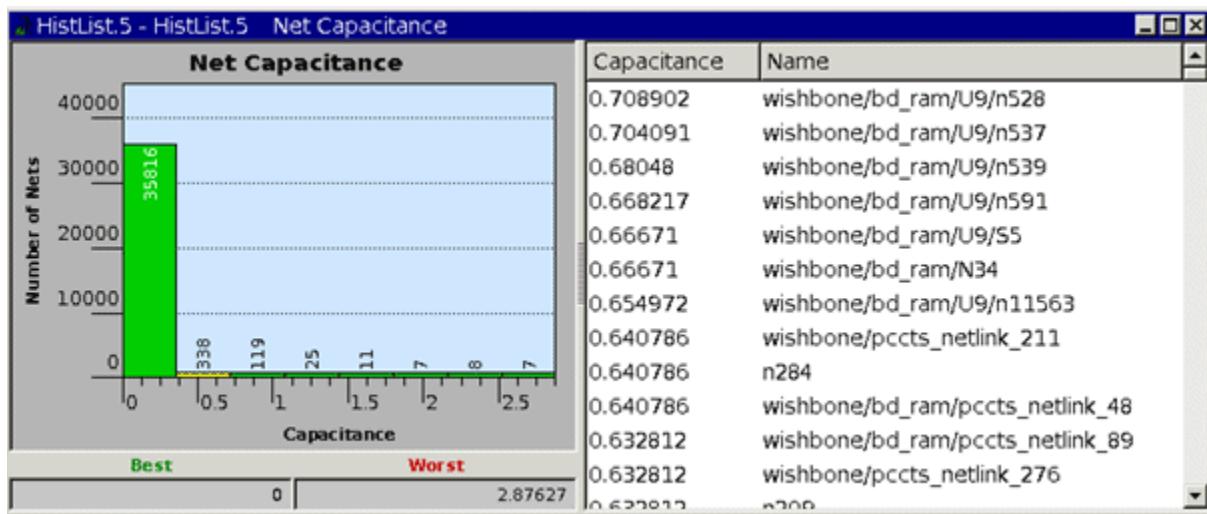
Use a net capacitance histogram to view the distribution of capacitance values for all the nets in the design.

To generate the histogram,

1. Click the  button on the Histograms toolbar or choose Timing > Histogram > Net Capacitance.
The Net Capacitance dialog box appears.
2. (Optional) Set options to control the distribution of capacitance values and configure the histogram plot.
3. Click OK.

[Figure 18-13](#) shows an example of a net capacitance histogram. You can click any histogram bar to select its bin and view a list of the nets in that bin, together with their corresponding capacitance values. The capacitance units (such as pF or nF) are determined by the logic library.

Figure 18-13 Net Capacitance Histogram



You can display a profile showing the relative amounts of pin and wire capacitance for a net, by selecting the net in the list and choosing Timing > Net Capacitance Profile. The net capacitance profiler appears showing the information about the selected net.

See Also

- [Viewing and Customizing Histograms](#)

Examining Design Rule Slack

Use the design rule histogram to view slack values from a design rule check. For example, a design rule histogram plot for the maximum capacitance rule shows the amount of capacitance slack of each net (the maximum capacitance limit of the design rule minus the actual capacitance of the net).

The tool checks nets, ports, and pins for violations of design rules specified in the logic library or by design rule commands, such as the `set_max_capacitance` command. For more information about design rule checking, see [Design Rule Constraints](#).

To generate a histogram plot of design rule slack values,

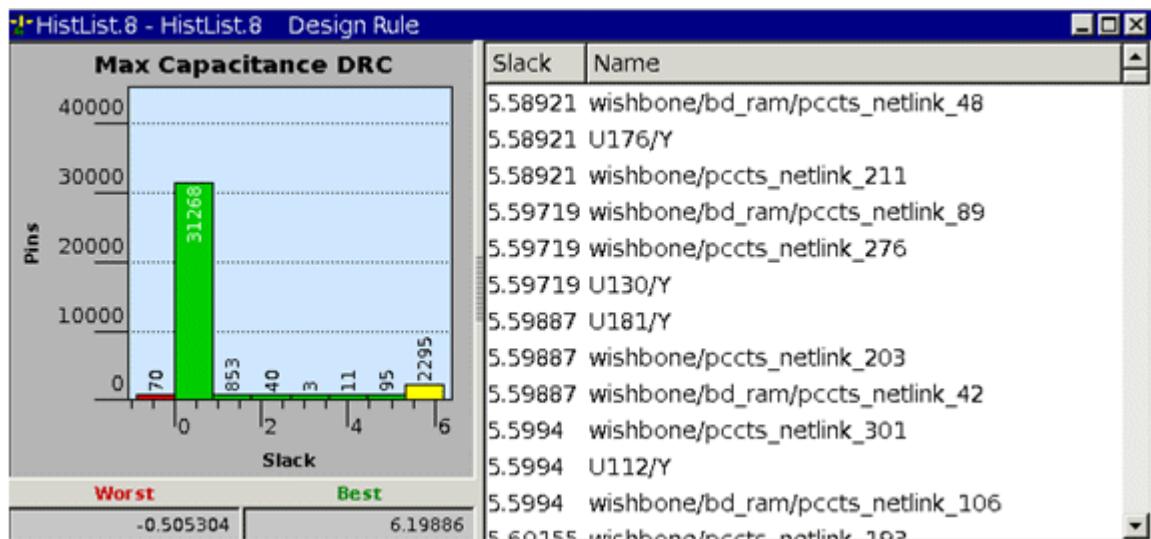
1. Choose Timing > Histogram > Design Rule Histogram, or click the equivalent icon in the histogram toolbar.
The DRC dialog box appears.
2. Select the type of design rule.

The choices are minimum or maximum capacitance, minimum or maximum transition time, and minimum or maximum fanout. The default is minimum capacitance.

3. (Optional) Set options to control the distribution of slack values and configure the histogram plot.
4. Click OK.

[Figure 18-14](#) shows an example of the design rule histogram. Bins that meet the design rule are shown in green, while those that violate the design rule are shown in red. The selected bin is highlighted in yellow.

Figure 18-14 Design Rule Histogram (Maximum Capacitance)



See Also

- [Viewing and Customizing Histograms](#)

Examining Timing Bottlenecks

Use a timing bottleneck histogram to find the cells in the design that are causing the greatest number of timing violations. A timing bottleneck is a point in the design that contributes to multiple timing violations. For more information about bottleneck analysis, see [Bottleneck Report](#).

To generate the histogram,

1. Click the button on the Histograms toolbar or choose Timing > Histogram > Timing Bottleneck.

The Timing Bottleneck dialog box appears.

2. Select the bottleneck cost type.

The choices are path count, path cost, or fanout endpoint cost. The default is path cost.

3. (Optional) Specify the startpoints, throughpoints, or endpoints to generate a histogram for paths from, through, or to specific inputs, registers, or outputs.

You can specify the objects by selecting them or entering their names. For example, if you enter a single pin name in the From box (and leave the Through and To boxes blank), The tool finds all the paths that originate at the specified pin and plots the resulting slack values in the histogram.

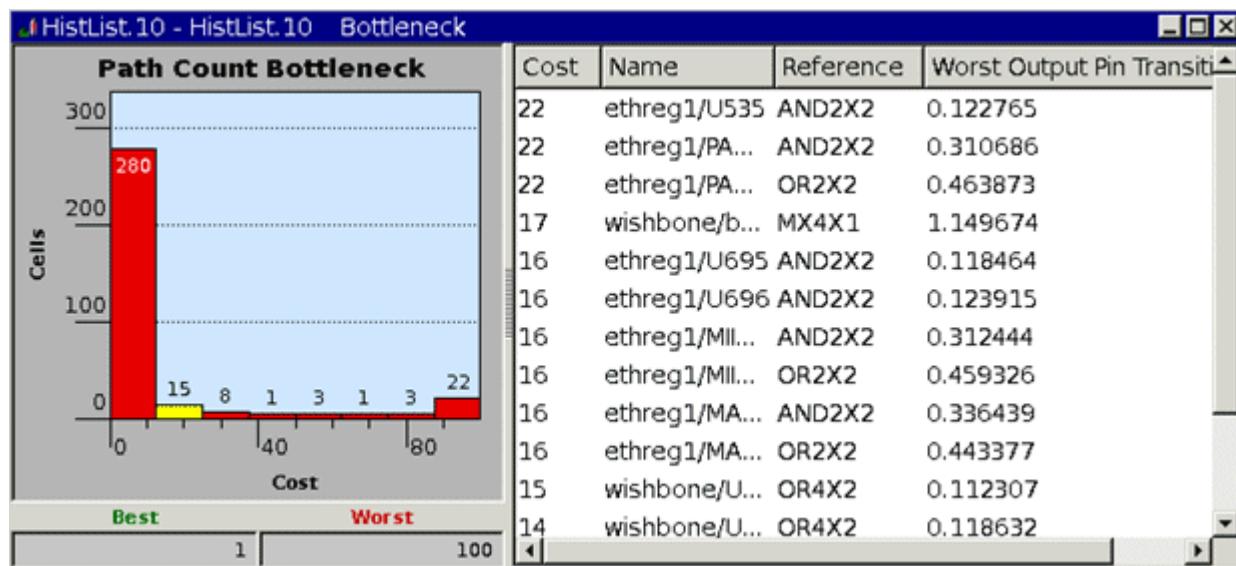
To specify the currently selected pins, ports, nets, or clocks in the From, Through, or To box, select an object type option and click the Selection button.

4. (Optional) Set options to control the distribution of count or cost values and configure the histogram plot.

5. Click OK.

[Figure 18-15](#) shows an example of the bottleneck histogram.

Figure 18-15 Bottleneck Histogram



The histogram plot shows the distribution of slack values for the paths included in the scope of the analysis. The numbers at the top of each bin indicate the total paths in the bin. Green bins (on the positive side of 0) contain paths in the design that met their constraints. Red bins (on the negative side of 0) contain paths that failed their constraints.

If you hold the pointer over a bin, an InfoTip displays the number of paths and the range of slack values in the bin. An InfoTip might read, for example,

Range: 11.8 to 23.6

Contents: 15

When you click a bin to select it, the bin turns yellow and the count or cost value, instance name, reference name, and worst output pin transition time for each cell appear in the list to the right of the histogram plot.

You can select cells in the list to view information about them. Right-click and choose a command to view a schematic of the worst path through each cell, a path data table of information about the worst path through the selected cells or the worst path through each selected cell, or a pin data table of information about the pins of each selected cell. If you select only one cell, you can view information about the worst path through the cell in a path inspector window.

See Also

- [Examining Path Slack](#)
- [Viewing and Customizing Histograms](#)

Analyzing Collections of Timing Paths

You can analyze collections of timing paths to determine where timing failures occur in your design. The path analyzer categorizes the timing paths by using rules based on available attributes, and then displays the categories in a color-coded treemap view. You can select predefined category rules or define custom category rules. You can also add subcategories.

The path analyzer is a high-level timing analysis tool that allows you to perform custom trend analysis. For example, you can

- Categorize paths by path group to see if timing violations are specific to a particular path group
- Create a custom category rule for paths with slack violations of more than one clock cycle to determine whether you need to specify multicycle path constraints on the paths
- Categorize the paths by their start clock and end clock values to identify cross-domain paths, which are the paths where these values are not the same

You can also tag appropriate blocks with block marks, such as physical partitions, and then categorize the paths by using the block mark attribute. This allows you to see if timing path failures result from a certain block in the design.

To open the path analyzer,

- Choose Timing > New Path Analyzer.

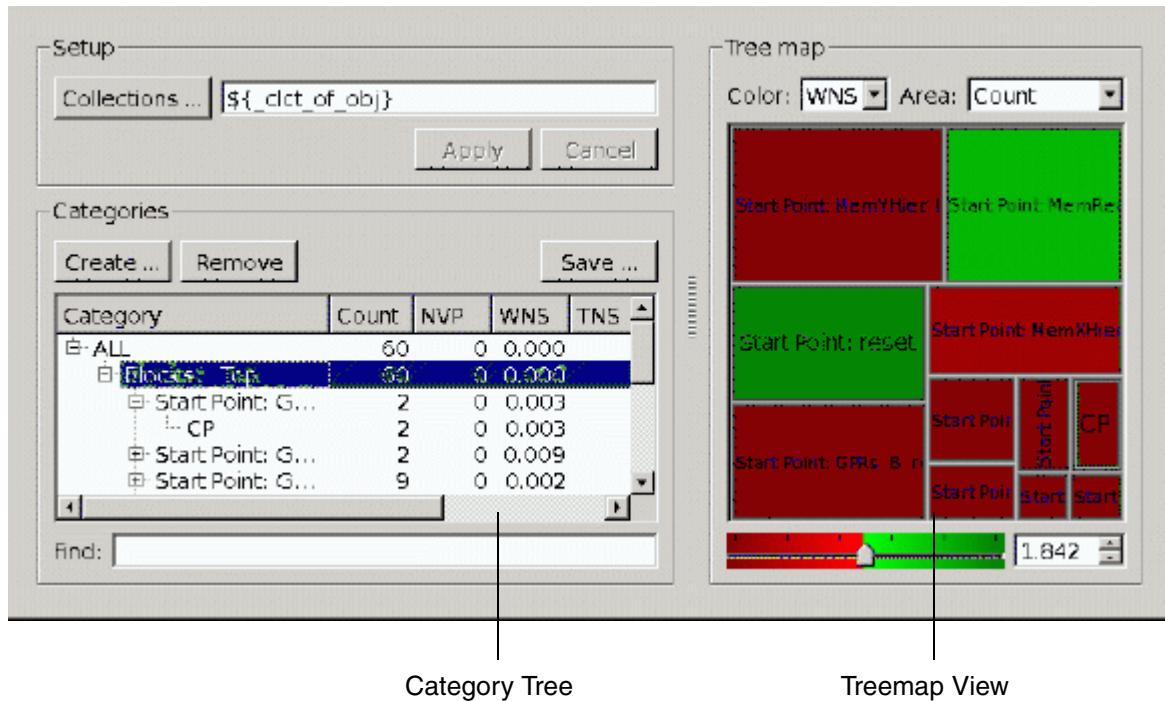
You must load and categorize the timing paths before you can view them in the treemap. For details, see [Loading Path Collections](#) and [Categorizing the Timing Paths](#). You can load one or more collections of timing paths from the same design.

Note:

The way you load timing paths in the Path Analyzer window depends on whether you are in the normal flow or the interactive multi-scenario analysis flow. For more information, see [Analyzing Timing Paths from Multiple Scenarios](#).

[Figure 18-16](#) shows an example of the path analyzer after the path collections have been loaded and categorized.

Figure 18-16 Path Analyzer in Normal Flow



The path analyzer window contains a category tree view on the left side and a treemap view on the right side. The category tree displays the path categories in an expandable tree. The treemap view displays path categories as a set of nested rectangles.

After categorizing the paths, you can save the path categories in a Tcl script file. You can categorize the paths by selecting and assigning individual category rules or by loading the categories that you previously saved in a file.

After loading and categorizing the paths, select All at the top of the category tree. The path analyzer displays the categories in the treemap view. The All category contains all the paths that you have loaded into the path analyzer.

Each row in the category tree represents a category or subcategory. For each category, the columns display the following information:

- Category displays the category name
- Count displays the number of paths in the category
- NVP displays the number of violating paths
- WNS displays the worst negative slack value
- TNS displays the total negative slack value
- TPS displays the total positive slack value

You can expand or collapse a category or subcategory by clicking its expansion button. A plus sign indicates a collapsed category. A minus sign indicates an expanded category.

You can expand or collapse multiple categories simultaneously.

- To expand all the categories and display their subcategories, right-click and choose Expand All.
- To collapse all the categories and hide their subcategories, right-click and choose Collapse All.

You can also traverse the category tree and expand or collapse individual categories or subcategories by using the arrow keys on the keyboard.

In the treemap view, each category has its own rectangle, which can be tiled with smaller rectangles that represent subcategories. The area of a rectangle represents one dimension of the data, and the color represents a different dimension of the data.

To view all the categories in the treemap, select All in the category tree. To view the subcategories for a single category, select the category.

- The area of a rectangle represents one dimension of the data, such as the total number of paths or the number of violating paths.
- The color of a rectangle represents the worst negative slack in the category.

Red means the slack value is lower than the threshold value set at the bottom of the treemap view, and green means the slack value is higher than the threshold value. These colors also change from light to dark to indicate how far the slack value is from the threshold value.

You can select a category containing paths that you want to analyze further with other analysis tools.

- To select the paths in the selected category, right-click and choose Select Category Paths.
- To view a histogram of the slack distribution in the selected category, right-click and choose Create Histogram.

From the histogram, you have access to the schematics, path data tables, timing reports, and path inspector windows to further analyze the paths.

- To display the paths from the selected category in a path data table, right-click and choose Create Data Table.

From the data table, you have access to the schematics, histograms, and path inspector windows to further analyze the paths.

- To load the paths from the selected category into a new path analyzer window, right-click and choose New Path Analyzer.

For more information about using the path analyzer, see

- [Loading Path Collections](#)
- [Categorizing the Timing Paths](#)
- [Removing Categories](#)
- [Saving Path Categories in a File](#)
- [Loading Path Categories From a File](#)
- [Marking Blocks and Applying Block Category Rules](#)
- [Creating Custom Category Rules](#)

See Also

- [Interactive Multi-Scenario Analysis Flow](#)

Loading Path Collections

Before you can view timing path data in the path analyzer, you must load and categorize the paths. You should specify only timing path collections. The path analyzer ignores collections that do not contain timing paths and objects, other than timing paths, within a hybrid collection, are also ignored.

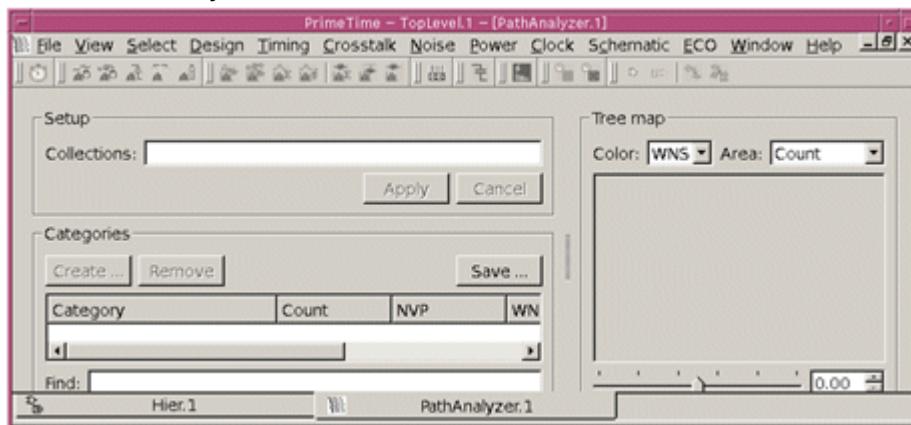
Note:

The way you load timing paths in the Path Analyzer window depends on whether you are in the normal flow or the interactive multi-scenario analysis flow. For more information, see [Analyzing Timing Paths from Multiple Scenarios](#).

To load timing path collections in the normal flow,

1. Choose Timing > New Path Analyzer to open the Path Analyzer window as shown in [Figure 18-17](#).

Figure 18-17 Path Analyzer Window



2. Specify the path collections in the Collections box by using one of the following methods:

- Click the Collections button to open the Collections Dialog dialog box, select the options for the collections that you want to load, and click OK.
- Enter the names of one or more collections in the Collections box.

Separate the collection names with blank spaces. Alternatively, you can click the Collections button to open the Collections Dialog dialog box, select the options for the collections that you want to load, and click OK.

- Enter a Tcl command that provides a timing path collection.

The command must be enclosed in square brackets ([]), which indicates that the tool must evaluate the command to obtain the collection. For example, type \$pathClct1 [get_timing_paths]. You can specify multiple collections in this way.

3. Click Apply.

See Also

- [Categorizing the Timing Paths](#)
- [Analyzing Collections of Timing Paths](#)

Categorizing the Timing Paths

The path analyzer uses rules based on timing attribute values to categorize the paths in a collection. You can select a predefined category rule or select a custom category rule that you defined. In addition, you can select a category and divide it into subcategories.

To categorize the timing paths,

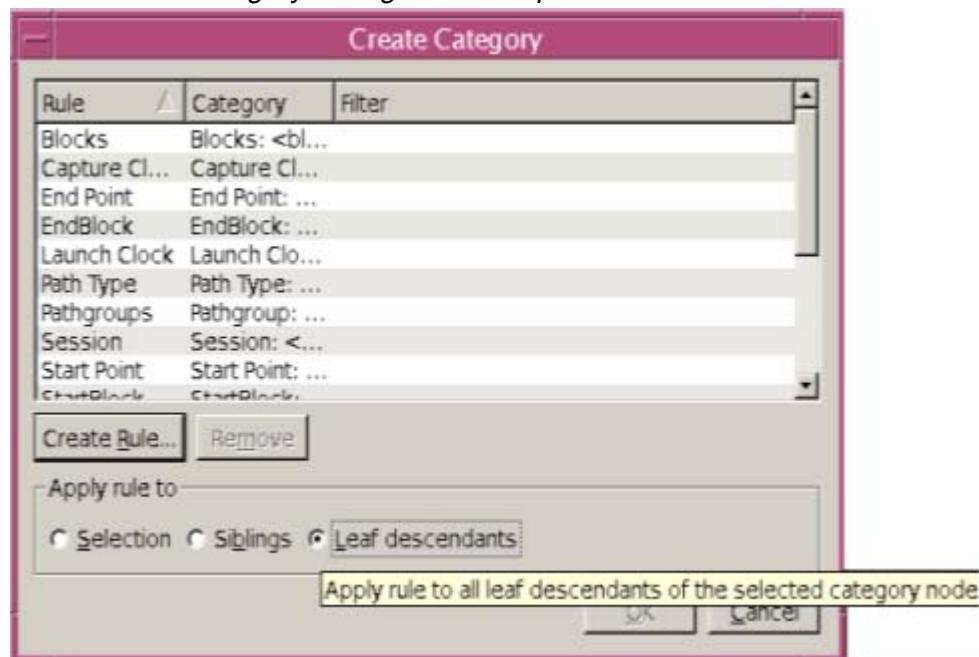
1. Select a category in the Categories list.

To categorize all the paths, select All. To further categorize a category into subcategories, select the category.

2. Click the Create button.

The Create Category dialog box appears listing all of the existing category rules. An example is shown in [Figure 18-18](#).

Figure 18-18 Create Category Dialog Box Example



3. Select a rule.

You can select a predefined rule or a custom rule that you have defined.

4. (Optional) Apply the rule to the selection, its siblings, or its leaf descendants by selecting the appropriate option.

The rule is applied to the selection by default.

5. Click OK.

For information about defining custom category rules, see [Creating Custom Category Rules](#).

The paths are categorized by the rules that you select. For each category, the number of violating paths (NVP), worst negative slack (WNS), total negative slack (TNS), and total positive slack (TPS) are displayed in separate columns in the path analyzer window.

The treemap displays hierarchical data in a tree structure as a set of nested rectangles. Each branch of the tree is displayed in a rectangle, which is then tiled with smaller rectangles representing subbranches. A rectangle representing a leaf node has an area proportional to a specified dimension on the data. The leaf nodes are colored to show a separate dimension of the data.

You can add or remove categories at any time. You can also save the categories in a file and load the categories from a file.

See Also

- [Removing Categories](#)
- [Saving Path Categories in a File](#)
- [Loading Path Categories From a File](#)
- [Marking Blocks and Applying Block Category Rules](#)
- [Creating Custom Category Rules](#)
- [Analyzing Collections of Timing Paths](#)

Removing Categories

You can remove all of the categories or just the subcategories of the selected category.

To remove a category or subcategory,

1. Select the category from which you want to remove subcategories.
To remove all the categories, select All.
2. Click Remove.

See Also

- [Categorizing the Timing Paths](#)
- [Analyzing Collections of Timing Paths](#)

Saving Path Categories in a File

You can save the timing path categories in a Tcl script file that you can use later to categorize other collections of timing paths or the timing paths in another instance of the path analyzer.

To save the path categories in a script file,

1. Click the Save button in the path analyzer.
The Save Categorization Script dialog box appears.
2. Select a file or enter the file name in the “File name” box.
3. Click Save.

See Also

- [Loading Path Categories From a File](#)
 - [Categorizing the Timing Paths](#)
 - [Analyzing Collections of Timing Paths](#)
-

Loading Path Categories From a File

You can categorize timing paths in the path analyzer by loading path categories that you previously saved in a Tcl script file.

To load path categories from a script file,

1. Click the Load button in the path analyzer.
The Load Categorization Script dialog box appears.
2. Select a file or enter the file name in the “File name” box.
3. Click Open.

The path analyzer removes the current categories, if any, and categorizes the paths by running the specified Tcl script.

Alternatively, you can load the path categories by sourcing the file.

See Also

- [Saving Path Categories in a File](#)
- [Categorizing the Timing Paths](#)
- [Analyzing Collections of Timing Paths](#)

Marking Blocks and Applying Block Category Rules

You can use block marks when categorizing timing paths in the path analyzer or during block abstraction from an abstract clock graph view.

A block mark can appear in the GUI as a text label for a timing path category generated from a dynamic category rule. A common application is to mark interesting IP blocks within a design. Keep these block marks short and meaningful. For example, you could use a team tag to mark blocks that belong to a specific team.

- To set a block mark on one or more hierarchical or leaf-level cell instances, use the `gui_set_cell_block_marks` command.
- To get a list of the block mark string values on one or more hierarchical or leaf-level cell instances, use the `gui_get_cell_block_marks` command.

For example, to set the block mark for a hierarchical cell instance identified by a cell name and then get the block mark of that cell instance, enter the following commands:

```
pt_shell> gui_set_cell_block_marks I_TOP/I_ALU ALU
pt_shell> gui_get_cell_block_marks I_TOP/I_ALU
ALU
```

- To list the cell names and block mark values for all cells of the design that are marked, use the `gui_list_cell_block_marks` command.

For example, to set block marks on hierarchical cell instances and then list all of the marked cell names and their block marks, enter the following commands:

```
pt_shell> gui_remove_cell_block_marks -all
pt_shell> gui_set_cell_block_marks I_TOP/I_ALU ALU
pt_shell> gui_set_cell_block_marks I_TOP/I_REG_FILE REG_FILE
pt_shell> gui_list_cell_block_marks
I_TOP/I_ALU ALU
I_TOP/I_REG_FILE REG_FILE
```

- To remove the block mark string value for one or more hierarchical or leaf-level cell instances, use the `gui_remove_cell_block_marks` command.
- To remove all of the block marks on any marked cell, use the `gui_remove_cell_block_marks -all` command.

When working with the path analyzer, you can use block-related path attributes as dynamic category rules. [Table 18-2](#) shows the available category rules for blocks.

Table 18-2 Category Rules for Blocks

Rule name	Attribute	Definition
Blocks	blocks	An ordered block level path; includes start, through, and end blocks
ThroughBlocks	through_blocks	Includes through blocks, but does not include the start or end blocks
StartBlock	start_blocks	Includes the start block
EndBlock	end_block	Includes the end block
StartEndBlocks	start_end_blocks	A pair of blocks consisting of the start and end blocks
StartEndBlocksSorted	start_end_blocks_sorted	A pair of blocks consisting of the start and end blocks sorted alphabetically

See Also

- [Analyzing Collections of Timing Paths](#)
- [Categorizing the Timing Paths](#)
- [Creating Custom Category Rules](#)

Creating Custom Category Rules

To categorize timing paths in the path analyzer, you can select a predefined category rule or create a custom category rule. To create a custom category rule, you define a rule based on a timing attribute value. You define the rule by using a regular expression of the form

attribute_name operator value

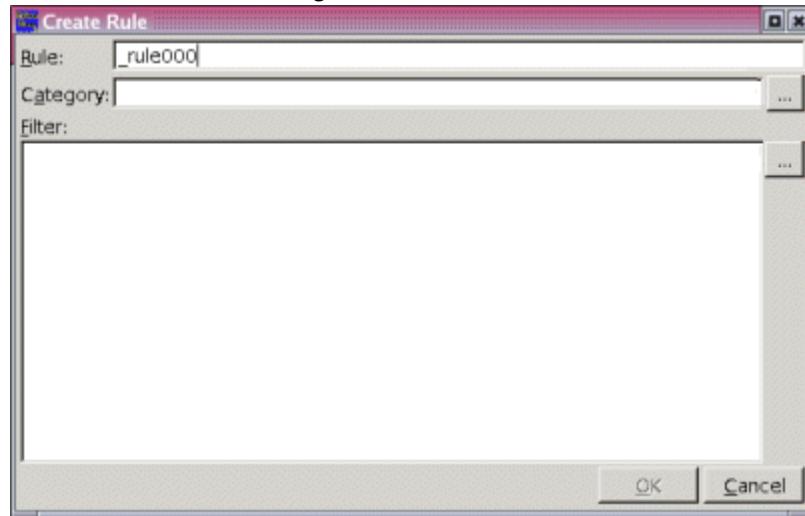
You can create custom category rules based on available timing path attributes. The priority of the custom categories is stored in the Category file. If a path does not satisfy any of the available grouping criteria, it is placed in the Uncategorized group for each level that has at least one subcategory.

To create a custom category rule,

1. Click the Create Rule button in the Create Category dialog box.

The Create Rule dialog box appears as shown in [Figure 18-19](#).

Figure 18-19 Create Rule Dialog Box



2. Enter a rule name in the Rule box.

This name is for reference only.

3. Enter a category name in the Category box.

Alternatively, you can click the button and select an attribute in the Category Attribute Chooser dialog box.

4. Enter a filter expression in the Filter box.

Alternatively, you can click the button and define a filter expression in the Filter Attribute Chooser dialog box.

5. Click OK.

For more information about selecting category attributes or defining a filter expression, see

- [Selecting Category Attributes for a Custom Category Rule](#)
- [Defining a Filter Expression for a Custom Category Rule](#)

The new category rule appears in the Create Category dialog box. You can use this rule to categorize the timing paths. When you apply the rule, the information in the Category box appears in the Categories List in the path analyzer window.

You can also use the Category box to specify a dynamic category. When creating a dynamic category, specify the attribute in the Category box and leave the filter box blank. For example, in the Category box you could type `EPCP: <endpoint_clock_pin.full_name>`, using the angle brackets to indicate a dynamic category.

You can concatenate multiple attributes by using a blank space or another separator character to create more complex dynamic categories. For example, you could type `EPC_SP: <endpoint_clock_pin.full_name> <startpoint.full_name>` to create a dynamic category based on the endpoint clock name and the startpoint name.

Note:

Using a slash (/) to separate the attribute values causes hierarchical categorization.

See Also

- [Categorizing the Timing Paths](#)
- [Analyzing Collections of Timing Paths](#)

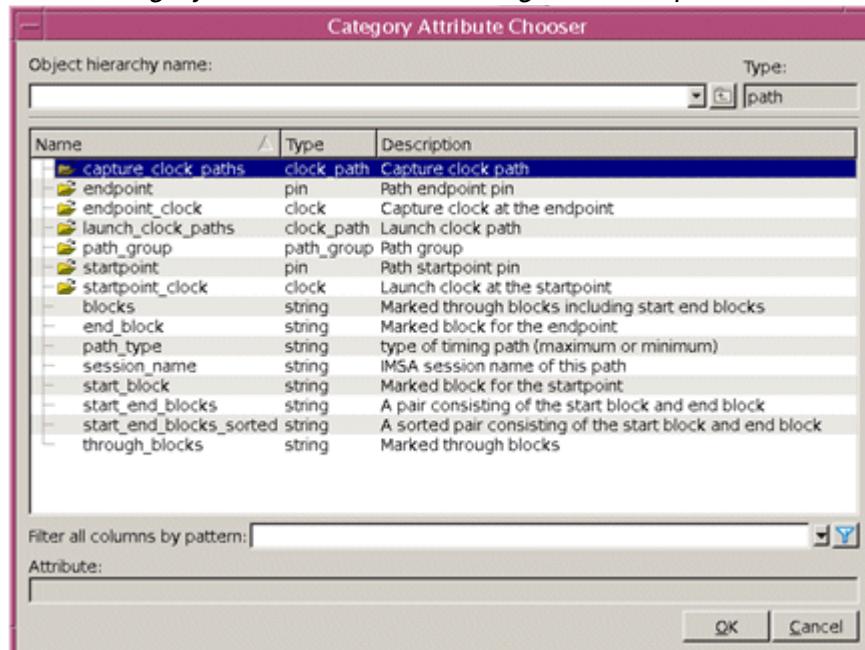
Selecting Category Attributes for a Custom Category Rule

To select an attribute in the Category Attribute Chooser dialog box,

1. Click the  button beside the Category box in the Create Rule dialog box.

The Category Attribute Chooser dialog box appears with a list of the available category attributes. An example is shown in [Figure 18-20](#).

Figure 18-20 Category Attribute Chooser Dialog Box Example



Some attributes are grouped hierarchically. A folder icon (📁) beside a name indicates a hierarchical attribute group.

2. (Optional) Move down an attribute group hierarchy, if necessary, to find an attribute.
 - To display the names of the attributes in a group, double-click the group name.
The path to the attribute group appears in the “Object hierarchy name” list and the attribute type appears in the Type box.
 - To move back up the hierarchy, select a path in the “Object hierarchy name” list.
To move up one level, click the  button.
3. (Optional) Filter the list of attributes to view only the names of categories in which you are interested.
 - a. Enter a name or name pattern in the “Filter all columns by pattern” box.
As you type the name, you can press the Tab key to display a name completion list of the available attribute names that match the pattern. If the name of the attribute that you want is in the list, select the name.
 - b. Click the  button.
To clear the filter and display the names of all the available attributes, enter a wildcard character (?) or (*) and click the  button.
4. Select an attribute in the list of attribute names.
The attribute name appears in the Attribute box.
5. Click OK.

See Also

- [Creating Custom Category Rules](#)
- [Defining a Filter Expression for a Custom Category Rule](#)

Defining a Filter Expression for a Custom Category Rule

You can define a filter expression for a custom category rule by using the Filter Attribute Chooser dialog box. You can select an attribute and operator to construct a rule. Both logic and arithmetic operators are supported. For example, define the expression `slack < 0` to identify violating paths of negative slack.

You can concatenate multiple filters just as you would concatenate logical expressions. You can construct full expressions by using `&&` and `||` to concatenate multiple filters. Select `&&`

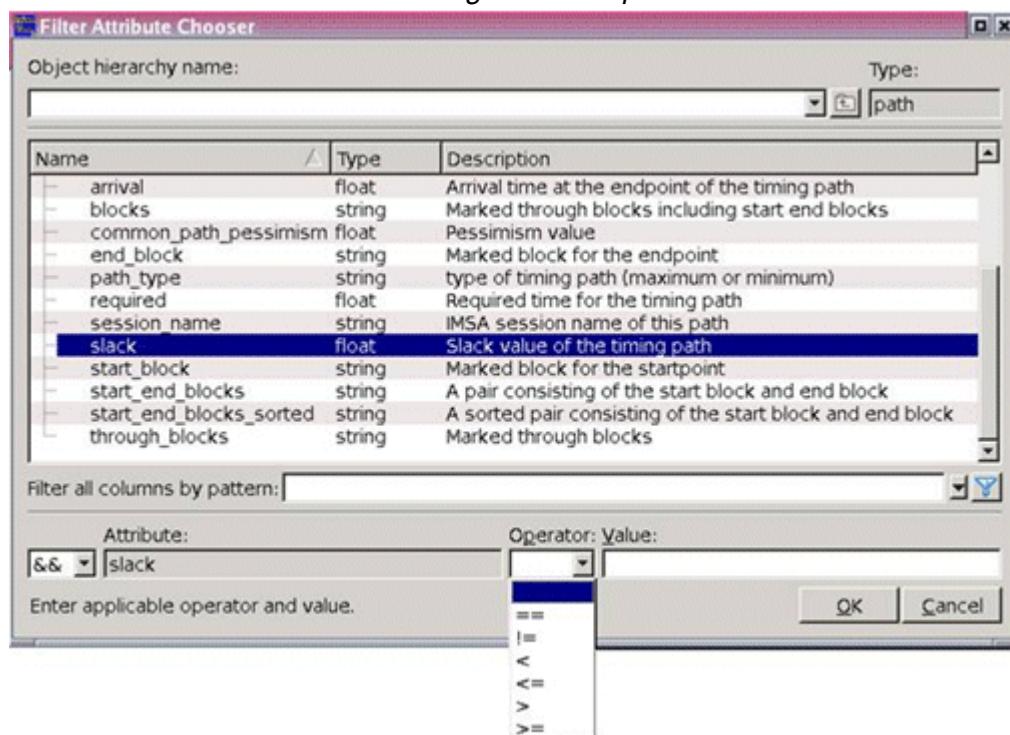
to create an AND relationship between filter expressions, or select **||** to create an OR relationship. Use parenthesis **()** to define the evaluation order for the expressions.

To define a filter expression in the Filter Attribute Chooser dialog box,

1. Click the **[...]** button beside the Filter box in the Create Rule dialog box.

The Filter Attribute Chooser dialog box appears with a list of the available filter attributes. An example is shown in [Figure 18-21](#).

Figure 18-21 Filter Attribute Chooser Dialog Box Example



Some attributes are grouped hierarchically. A folder icon () beside a name indicates a hierarchical attribute group.

2. (Optional) Move down an attribute group hierarchy, if necessary, to find an attribute.
 - To display the names of the attributes in a group, double-click the group name. The path to the attribute group appears in the “Object hierarchy name” list and the attribute type appears in the Type box.
 - To move back up the hierarchy, select a path in the “Object hierarchy name” list.

To move up one level, click the button.

3. (Optional) Filter the list of attributes to view only the names of attributes in which you are interested.

- a. Enter a name or name pattern in the “Filter all columns by pattern” box.

As you type the name, you can press the Tab key to display a name completion list of the available attribute names that match the pattern. If the name of the attribute that you want is in the list, select the name.

- b. Click the  button.

To clear the filter and display the names of all the available attributes, enter a wildcard character (?) or (*) and click the  button.

4. Select an attribute in the list of attribute names.

The attribute name appears in the Attribute box.

5. Select an operator in the Operator list.

The available operators depend on the type of attribute you select in step 4.

6. Enter an attribute value in the Value box.

7. Click OK.

See Also

- [Creating Custom Category Rules](#)
- [Selecting Category Attributes for a Custom Category Rule](#)

Examining Clock Paths in an Abstract Clock Graph

The abstract clock graph view allows you to visualize the relationships between the driving cells and output loads in a clock network or the relationships between clocks. You can view the relationships between primary and generated clocks, the convergence of multiple clocks, and the fanout of clocks to multiple sequential elements.

An abstract clock graph is similar to a clock schematic but with a higher level of abstraction. You can focus on key characteristics of the clock network while ignoring irrelevant objects, which can help you to selectively debug problems.

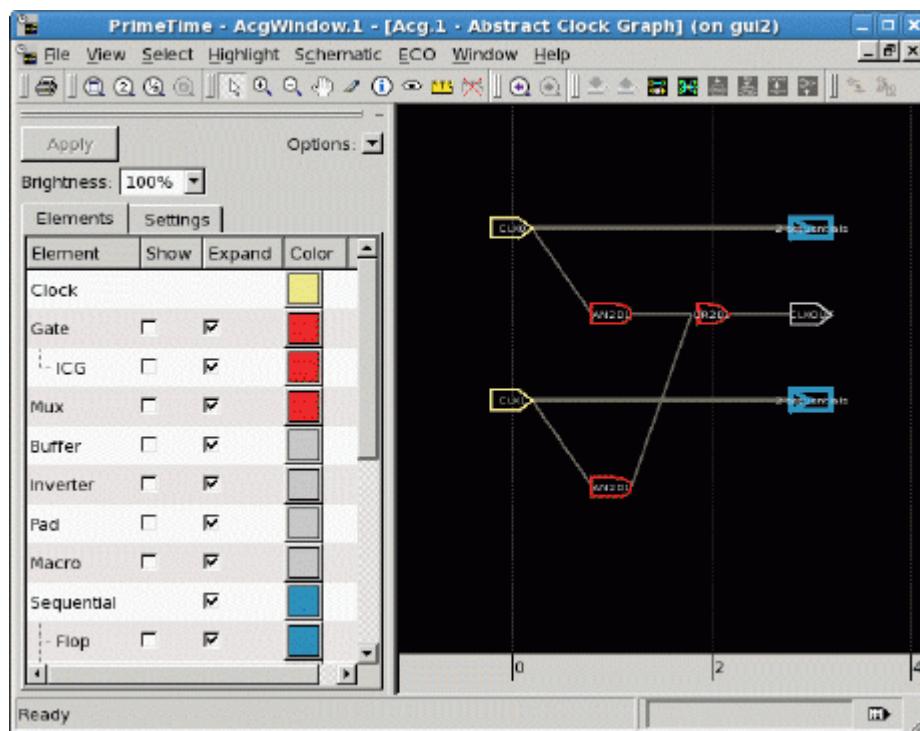
An abstract clock graph displays clock network objects by using symbols and arcs. Symbols represent visible objects, such as clocks, clock gates, buffers, inverters, and sequential cells. Arcs represent the clock paths between visible objects and are displayed as flylines.

- Logic gates such as buffers and inverters are represented by their schematic symbols and colored according to their gate types.

- Sequential cells are displayed as metacells and are initially collapsed into a single symbol for each clock path.
- Metacells that contain multiple cells have thicker lines, and the graph displays the number of cells. The lines emanating from a metacell do not necessarily represent the same net.

[Figure 18-22](#) shows an example of an abstract clock graph window.

Figure 18-22 Abstract Clock Graph Window



In this example, the clock graph contains two primary clocks, two clock gates, and one port. The blue shapes are metacells that represent the sets of sequential elements clocked by each of the clocks. Each line in the graph is an arc that represents the fanout of the clock. By default, buffers, gating logic, and pins belonging to the arcs are not shown.

Primary clocks, arcs, and metacells are always visible. Ports and grid lines are visible by default. Other objects are initially hidden in the arcs. For clarity and a more concise visualization, pins are not shown with their cells and reconvergent clock paths appear as diagonal lines.

You can view information about an object in an InfoTip by holding the pointer over the object.

- For a symbol, the InfoTip displays the object name and object type.
- For an arc, the InfoTip displays the path startpoint and endpoint.

- For a collapsed metacell, the InfoTip displays the number of sequential cells that it contains.

You can magnify and traverse the view by using the interactive zoom and pan tools and the zoom and pan commands. The x-axis is fixed at the bottom of the view, regardless of the vertical scroll position, and is scaled to reflect the positions of the gates. You can pan or scroll horizontally along the x-axis. The granularity of the x-axis tick marks and labels varies depending on the magnification level.

Note:

Text does not appear in a clock graph when it is below a certain size in pixels.

You can also use the arrow keys to scroll vertically or horizontally through the view.

You can select, highlight, and query objects in an abstract clock graph view by using interactive mouse tools. Objects that you select or highlight are automatically selected or highlighted in other views.

Note:

If you select or highlight an arc, the pins on the hidden cells in the arc are also selected or highlighted in other views, such as schematic or layout views.

You can customize the appearance of an abstract clock graph by setting options on the View Settings panel to display or hide grid lines, change object colors, and change the background color. You can also set display options that display or hide elements such as flylines, constraints, and text and that enable or disable InfoTips (“Tool tips”). To display these options on the View Settings panel, click the Settings tab.

For more information about abstract clock graph views, see

- [Opening Abstract Clock Graph Views](#)
- [Displaying and Hiding Clock Path Elements](#)
- [Reversing and Reapplying Changes](#)
- [Collapsing Selected Side Branches](#)
- [Collapsing Duplicate Clock Gates](#)
- [Viewing Clock Latency Over Time](#)
- [Displaying the Clock Trees Hierarchically](#)
- [Measuring Distances in an Abstract Clock Graph View](#)

See Also

- [Selecting or Highlighting Objects By Name](#)
- [Printing Images of Schematics, Abstract Clock Graphs, and Timing Waveforms](#)

Opening Abstract Clock Graph Views

You can create a new abstract clock graph view that shows all the clocks in the design or for one or more selected clocks.

To create an abstract clock graph view showing all clocks,

- Choose Clock > Clock Graph for All Clocks in the main window.

To create an abstract clock graph view showing selected clocks,

1. Select the clocks in the clock analyzer.
2. Click the Clock Graph button.

The GUI opens a new abstract clock graph window and displays the clocks in an abstract clock graph view. The View Settings panel also appears, which allows you to set options that control how the view displays and expands various types of objects.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
-

Displaying and Hiding Clock Path Elements

By selectively displaying or hiding clock path elements such as buffers, gating logic, and hierarchical pins, you can focus on the elements of the clock network that you need to examine while ignoring irrelevant elements. You can

- Display or hide all the objects of a particular object type.
- Incrementally display or hide objects by expanding or collapsing individual arcs or metacells.

When you expand an arc, the graph automatically expands sequential metacells as needed to display the visible clock paths.

You can set options on the View Settings panel to display or hide individual object types and to control which types of objects are displayed when you expand an arc. The Show options display objects globally throughout the clock network. The Expand options control which types of objects appear when you expand an arc.

To display or hide objects in the active abstract clock graph view,

1. Click the Elements tab if it is not already selected.
2. Set options in the Show column as needed.
 - Select the options for the objects that you want to display.
 - Deselect the options for the objects that you want to hide.

The Show options display objects globally throughout the clock network. For example, to display all the buffers, select the Buffer option in the Show column.

3. Click Apply.

Note:

By default, you must click Apply on the View Settings panel to apply your changes to the active layout view. For general information about View Settings panel operations, see [View Settings Panel](#).

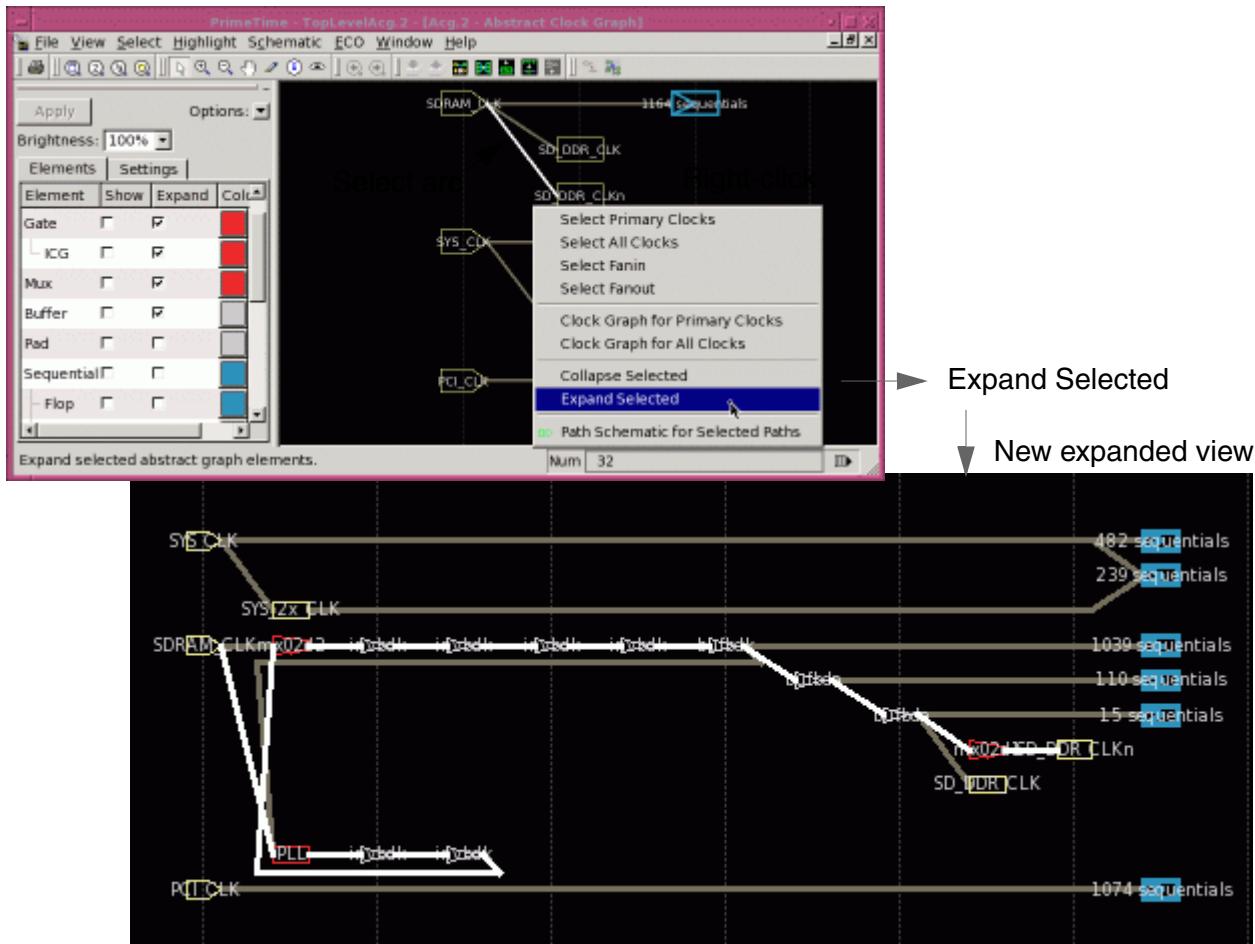
You can expand individual arcs to display their contents or collapse selected objects into arcs. You can also expand metacells to display the sequential cells or collapse selected sequential cells into metacells.

To expand an arc or metacell, you can

- Double-click the arc or metacell.
- Select the arc or metacell, and then right-click and choose Expand Selected.

[Figure 18-23](#) shows an example in which the arc from a primary clock to a generated clock is selected and expanded. The expanded view shows the PLL block, multiplexers, buffers, and inverters of the arc.

Figure 18-23 Expanding a Selected Arc



To collapse one or more logic gates into arc or sequential cells into metacells,

1. Select the logic gates or the sequential cells.
2. Right-click and choose Collapse Selected.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Collapsing Selected Side Branches](#)
- [Collapsing Duplicate Clock Gates](#)
- [Viewing Clock Latency Over Time](#)
- [Reversing and Reapplying Changes](#)

Reversing and Reapplying Changes

You can reverse or reapply operations that you perform in an abstract clock graph view, such as displaying the symbols for an object type, expanding an arc, or changing a view setting. You can reverse the most recent operation or sequentially reverse a series of operations. If you reverse one or more operations, you can reapply the most recently reversed operation or a series of operations.

To reverse an operation in an abstract clock graph view,

- Right-click and choose Back.

When you perform an operation that changes an abstract clock graph, the GUI retains a copy of the previous abstract clock graph, and redisplays it if you reverse the operation. Similarly, when you reverse an operation, the GUI retains a copy of the changed abstract clock graph, and redisplays it if you reapply the operation.

To reapply an operation in an abstract clock graph view,

- Right-click and choose Forward.

Note that you cannot reverse or reapply interactive view changes such as zooming or panning, highlighting, or applying or removing rulers.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
 - [Displaying and Hiding Clock Path Elements](#)
-

Collapsing Selected Side Branches

You can select and collapse individual side branches in an abstract clock graph. The GUI collapses a side branch into a single row, which provides more room for the expanded branches. By selectively collapsing side branches, you can focus on the branches of the clock tree that are of interest while keeping the overall context in view as you debug skew and insertion delay issues.

To collapse one or more side branches,

1. Select the branch roots that you need to collapse.
2. Right-click and choose Collapse Branches.

To expand collapsed side branches,

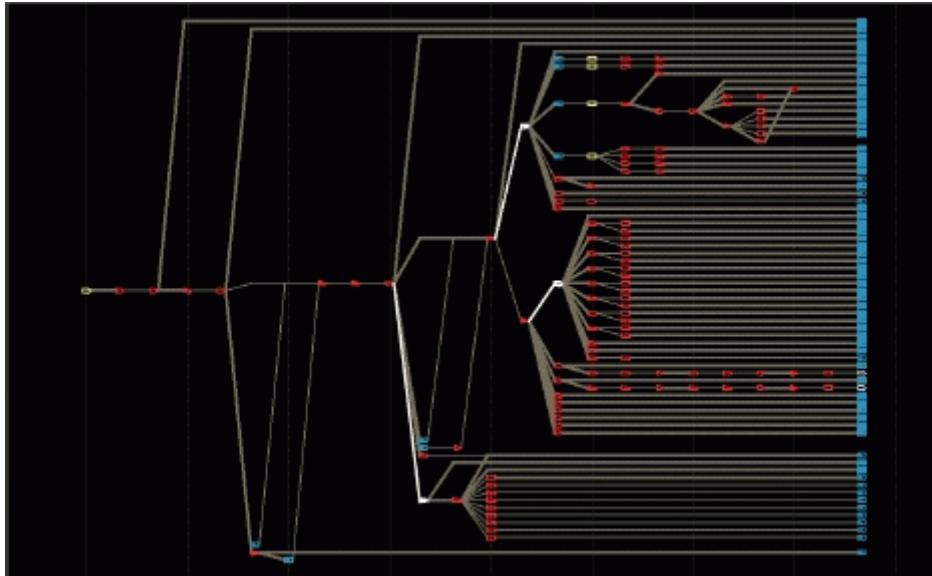
1. Select the branch roots you need to expand.

2. Right-click and choose Expand Branches.

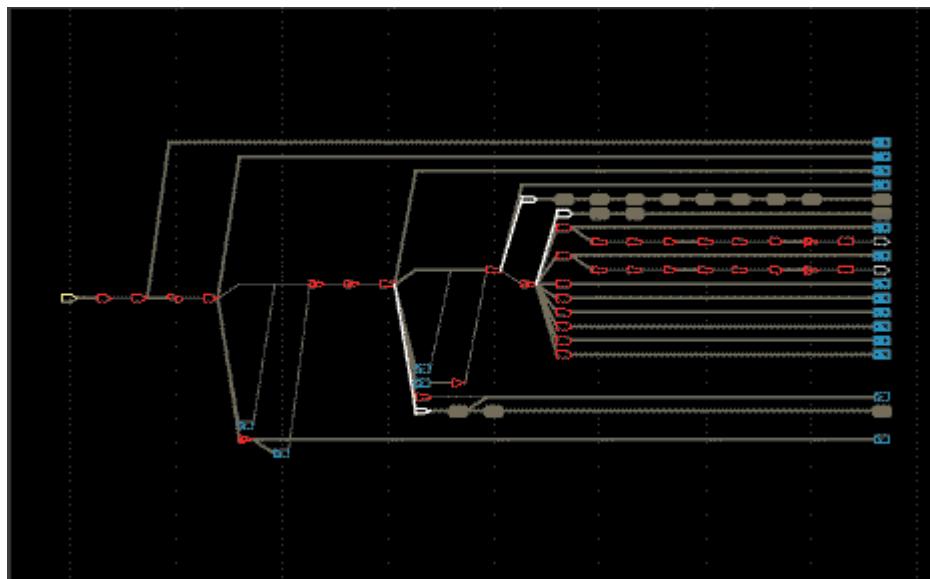
[Figure 18-24](#) shows examples of an abstract clock graph before and after selected side branches have been collapsed.

Figure 18-24 Example of Branch Collapsing in Abstract Clock Graph

Before Collapsing Selected Branches



After Collapsing Selected Branches



See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
 - [Displaying and Hiding Clock Path Elements](#)
 - [Collapsing Duplicate Clock Gates](#)
-

Collapsing Duplicate Clock Gates

You can compress repeated data in an abstract clock graph, when the details are not of interest, by collapsing cloned and potentially cloned clock gates into a single representation. Cloned clock gates are duplicate clock trunks that use the same clock gating. Collapsing cloned and potentially cloned clock gates gives you more room for the clock tree branches that you need to analyze or debug.

To collapse clock gates in an abstract clock graph,

1. Click the Settings tab on the View Settings panel.
2. Select the “Collapse gates” option.
3. Click Apply.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
 - [Displaying and Hiding Clock Path Elements](#)
 - [Collapsing Selected Side Branches](#)
-

Viewing Clock Latency Over Time

You can analyze and debug clock network timing by displaying clock latency in an abstract clock graph view. The clock latency layout arranges clock elements by their scheduling relationships over time. You can examine clock skew, bottlenecks, insertion delays, and the intrinsic path delays caused by insertion of integrated clock-gating (ICG) cells.

To display clock latency in an abstract clock graph view,

1. Click the Settings tab on the View Settings panel.
2. Select the “Show latency” option.
3. Click Apply.

The clock latency layout displays concise and consistent timing information for the clock trees. The x-axis represents clock tree logic relative to time. The graph positions logic gates based on the latency of their input clock pins.

- The clock root appears at arrival time 0 and increases the arrival time on each pin monotonically.
- Logic gates are aligned on their left borders based on the latency of their input clock pins. Clock input pins with nonmonotonically increased arrival times are not displayed.

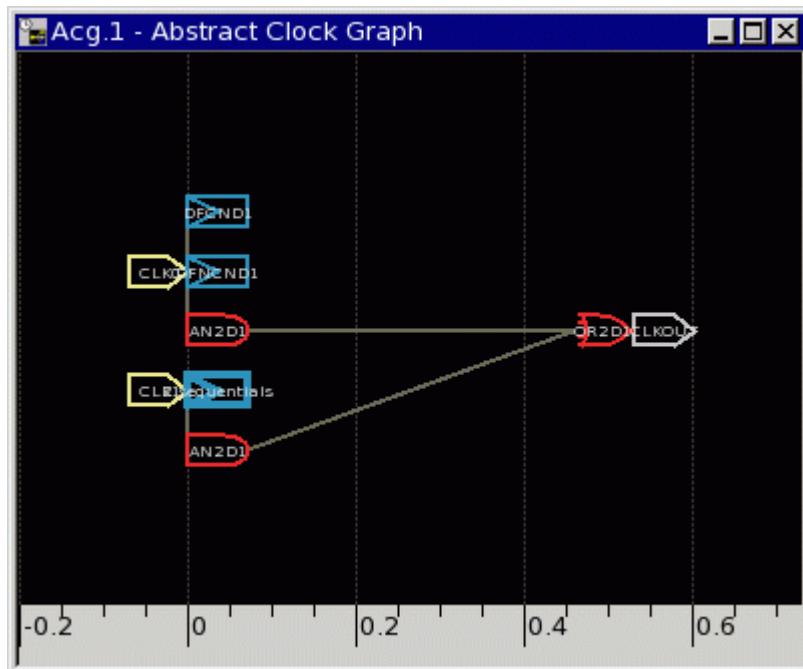
If more than one input has a clock pin attribute, the cells are aligned with the worst case arrival time. If you display multiple clocks in the same graph, the clock positions are based on the maximum latency among the delays on the clock pins of the selected clocks.

- All nets shown entering or leaving a gate converge at a single point, regardless of whether they connect to the same pin.

Similarly, if multiple outputs are in the clock path, their nets are shown emerging from a single point.

[Figure 18-25](#) shows an example of the clock latency layout in an abstract clock graph view.

Figure 18-25 Clock Latency Layout in an Abstract Clock Graph View



If you hold the pointer over an object to preview its attributes, the InfoTip includes the arrival time. If you click an object with the Query tool, the Query panel displays the detailed timing information, such as the arrival and transition times and the delays at each pin.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Displaying and Hiding Clock Path Elements](#)

Displaying the Clock Trees Hierarchically

By default, the abstract clock graph view shows a single, flat representation of each clock path from source to sinks. You can display a hierarchical representation of the clock trees by collapsing the graph to the top-level design, and then expanding individual hierarchical cells. You can also control whether the hierarchical display represents the logic design hierarchy, which is the default, or the physical design hierarchy.

To collapse the abstract clock graph to the top-level design,

1. Select the Hierarchy option in the Show column on the View Settings panel.
2. Click Apply.

To expand a hierarchical cell, you can

- Double-click the cell.
- Select the cell, and then right-click and choose Expand Selected.

To set the hierarchy display to represent the physical design hierarchy,

1. Click the Settings tab on the View Settings panel.
2. Select the “Physical hierarchy” option.
3. Click Apply.

You select the “Physical hierarchy” before or after you select the Hierarchy>

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Displaying and Hiding Clock Path Elements](#)

Measuring Distances in an Abstract Clock Graph View

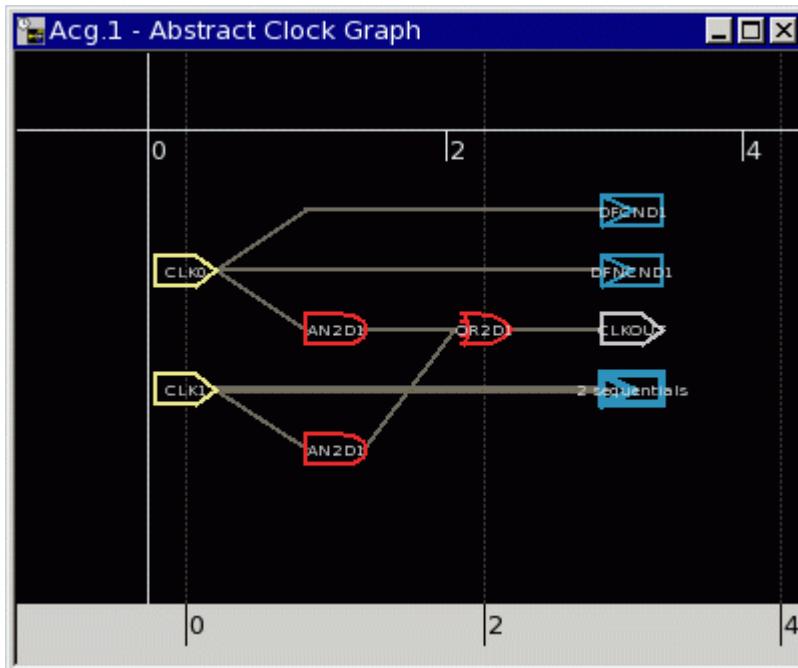
You can determine the deltas between points in an abstract clock graph view by using the Ruler tool. The rulers are similar to rulers in a layout view. However, the rulers in an abstract clock tree graph view are always horizontal. The vertical axis serves more as a cursor than as a measuring aid.

To draw rulers in an abstract clock graph view,

1. Click the  button on the Mouse Tools toolbar, or choose View > Mouse Tools > Ruler Tool.

By default, a cross hair ruler appears attached to the pointer in the abstract clock graph view.

Figure 18-26 Ruler Tool in an Abstract Clock Graph View

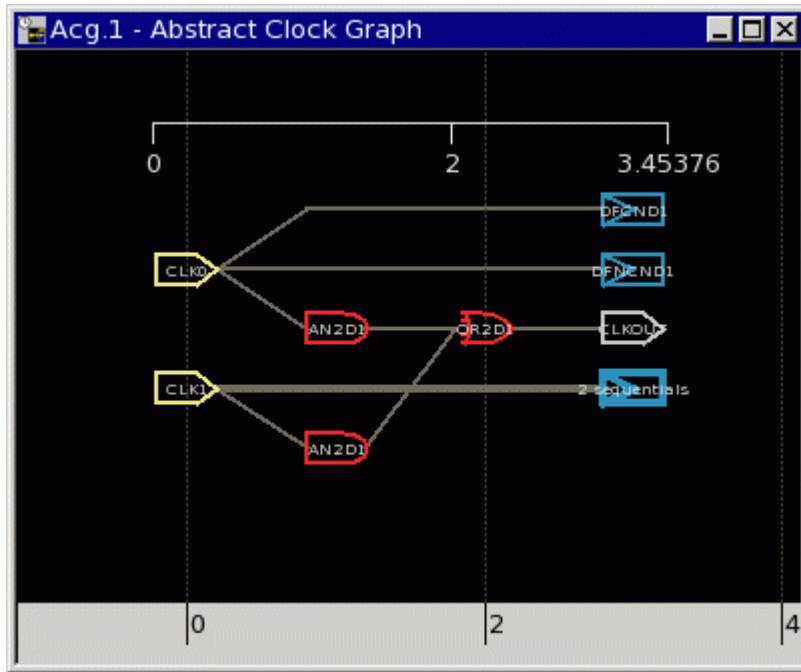


2. Drag the pointer in the abstract clock graph view to begin drawing a ruler.

As you move the pointer, a preview of the ruler appears and expands or contracts to indicate the horizontal distance between the starting point, the ruler origin, and the current pointer position. You can cancel a pending ruler by pressing the Esc key.

3. Release the pointer where you want to end the ruler.

Figure 18-27 Ruler in an Abstract Clock Graph View



4. (Optional) Repeat steps 2 and 3 if you want to draw another ruler.

To remove the ruler nearest the pointer position,

1. Move the pointer over the ruler you need to remove.
2. Right-click and choose Remove Nearest Ruler.

To remove all rulers from the abstract clock graph view,

- Click the button on the Mouse Tools toolbar or right-click and choose Clear Rulers.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)

Analyzing Clock Domains and Clock-to-Clock Relationships

Use the clock analyzer to identify clock-to-clock relationships in the design and drive the clock analysis. This can help you to determine which clock domains communicate with other clock domains and identify the clock domain crossings.

A clock domain consists of a primary or generated clock and all the clocks derived from it. A generated clock has its own domain, which is a subset of the master clock domain. You can expand or collapse clocks based on the clock domains you are analyzing.

To open the clock analyzer,

- Choose Clock > Clock Analyzer.

The clock analyzer window consists of a hierarchical clock tree view on the left side and a clock matrix view on the right side.

- The clock tree view displays the names of the primary clocks in the design.

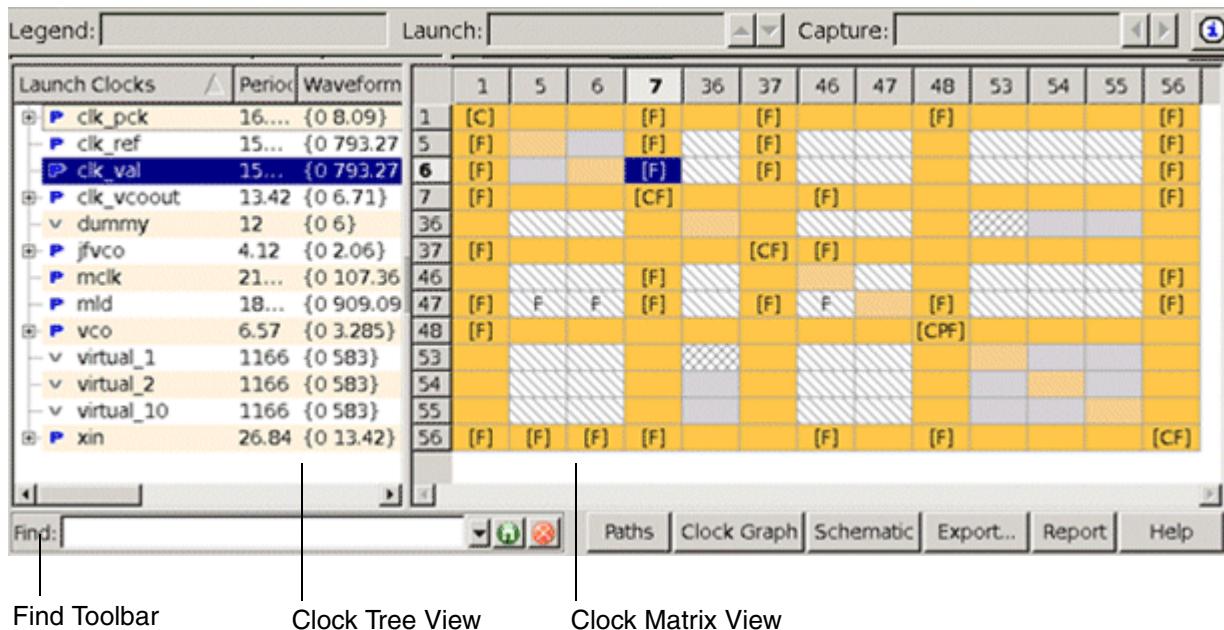
You can select one or more clocks in the clock tree view. By selecting a master clock, you also select the entire clock domain, including the generated clock domains beneath it.

- The clock matrix identifies the clock-to-clock relationships.

Each row and column of the matrix represents a clock in the design. Each clock has a corresponding number that is used to label its row and column.

[Figure 18-28](#) shows an example of the clock analyzer. You can locate specific clocks by using the Find toolbar at the bottom of the window.

Figure 18-28 Clock Analyzer



To locate clock objects and focus on a particular set of clocks, you can expand individual clock trees or all the clock trees. You can also sort the clock tree view by column, search for a clock within a clock domain, and apply filtering criteria.

An expandable clock tree, indicated by the expansion button (+) beside its name, represents the clock hierarchy among the primary clock and the entire set of dependent generated clocks beneath it. You can expand or collapse a clock by clicking its expansion button. A plus sign indicates a collapsed clock. A minus sign indicates an expanded clock.

You can expand or collapse multiple clocks simultaneously.

- To expand one or more selected clocks and display the fanout levels in the clock trees, right-click and choose **Expand all Selected Clocks**.
- To collapse one or more selected clocks and hide the fanout levels beneath them, right-click and choose **Collapse all Selected Clocks**.

As you expand or collapse the clock trees, the clock matrix displays or hides the corresponding clock rows and columns.

When you select a matrix cell, the current clock information appears in the Legend, Launch Clock, and Capture Clock boxes at the top of the clock analyzer window. For explicit paths, the Legend box shows the letter and description that corresponds to the selected cell.

Each matrix cell indicates the clock domain or specific clock-to-clock relationships. The path constraints and synchronous or asynchronous relationship are identified using letters and color associations. When a clock domain is collapsed, a summary of the path constraints appears in the matrix cell, which means that a collapsed cell can have several letters associated with it. The collapsed clock domain is indicated with a light brown cell to differentiate it from other cells. For more details, see [Clock Matrix Symbols and Colors](#).

If you want to examine the timing paths for a specific launch-capture clock pair, you can load the paths into the path analyzer. Select the matrix cell where the clocks intersect, and then right-click and choose “Analyze failing paths from *clock_name* to *clock_name*.”

You can also generate reports for a launch-capture clock pair.

- To generate an exceptions report, select the matrix cell where the clocks intersect and then right-click and choose “Report exceptions from *clock_name* to *clock_name*.”
- To generate a timing report, select the matrix cell where the clocks intersect and then right-click and choose “Report timing from *clock_name* to *clock_name*.”
- To generate a clock timing report, select the matrix cell where the clocks intersect and then right-click and choose “Report clock skew from *clock_name* to *clock_name*.”

The GUI displays the reports in the console log view and in pt_shell.

You can save the clock data in the clock tree view or the clock launch-capture clock data in the clock matrix view by exporting it to a CSV format file.

For more information about working with the clock analyzer, see

- [Querying Launch-Capture Clock Pairs](#)
- [Selecting Clocks or Clock Domains](#)
- [Sorting the Clock Tree View](#)
- [Finding Clocks by Name](#)
- [Filtering Clock Domains](#)
- [Locating Launch and Capture Clocks in the Clock Matrix](#)
- [Saving Clock Attribute or Clock Matrix Constraint Data](#)
- [Clock Matrix Symbols and Colors](#)

Querying Launch-Capture Clock Pairs

You can display information about a launch-capture clock pair in the clock matrix by using the Query tool,

To query a launch-capture clock pair,

1. Click the button in the top-right corner of the clock analyzer window.
The Query panel appears in the PrimeTime main window.
2. Click a cell in the clock matrix.
Information about the launch clock, the capture clock, and the interclock relationship appears on the Query panel.
3. Repeat step 2 to display information about a different launch-capture clock pair.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Selecting Clocks or Clock Domains

You can select clocks in the clock analyzer by clicking them in the clock tree view or the clock matrix view. To select multiple clocks, Ctrl+click each clock. To select a range of clocks, Shift+click the first and last clocks in the range.

To select all of the generated clocks for a given master clock,

1. Select a launch clock in the clock tree view.
2. Right-click and choose Deep Select all Generated Clocks.

The clock analyzer indicates a selected matrix cell by coloring the cell boundary rather than the entire cell so that the contents of the matrix cell remain visible.

You can select clocks or clock domains in the clock tree view that you want to analyze further in other views, such as a clock schematic or abstract clock graph view.

- To display selected clocks in a new abstract clock graph, select the clocks and then right-click and choose Clock Graph of Selected Clocks.
- To display a clock domain in a new abstract clock graph, select the clock domain and then right-click and choose Clock Graph for Clock Domain.
- To display selected clocks in a new clock schematic, select the clocks and then right-click and choose Schematic of Selected Clocks.
- To select the source pins or ports of selected clocks, select the clocks and then right-click and choose Select Source Pins or Ports.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Sorting the Clock Tree View

A design can have hundreds of clocks with some clocks deeply embedded in the tree. To assist in locating clock objects so that you can focus on a set of clocks, the clock analyzer provides alphanumeric sorting by attributes. By default, the clocks are sorted alphabetically in the clock tree.

To sort the clocks in descending order by the contents of a column,

- Click the column header.

To resort the clocks in ascending order,

- Click the same column header again.

Regardless of the sorting order, the clock hierarchies are retained.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Finding Clocks by Name

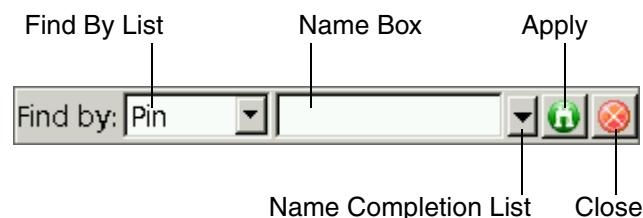
The clock analyzer provides a Find toolbar that can help you to locate a specific clock. You can perform multiple selections and select a range of clocks, and you can locate launch clocks based on the functional definition such as pins along the clock path.

To display the Find toolbar in the clock analyzer window,

1. Select a clock in the clock tree view.
2. Right-click and choose Find Clock(s).

The Find toolbar, shown in [Figure 18-29](#), appears below the clock tree view at the bottom of the clock analyzer window.

Figure 18-29 Find Toolbar



To locate a specific clock,

1. Right-click in the clock tree view and choose Find Tool to display the Find toolbar if it is currently hidden.

The Find toolbar is visible by default.

2. Select the type of objects to search for by name in the “Find by” list.

The choices are Name, Pin, Port, Cell, and LibCell. When you Name is selected, the tool locates the clock by its original clock name.

3. Type the name or name pattern in the Name box.

You can type a portion of the name and then press Enter or

click the button or press Enter.

The clock analyzer displays all matching clocks.

You can also type multiple clock patterns using a space as the delimiter. In this situation, the results of the searches for each different pattern are added together. You can begin typing a clock name and then press the Tab key to automatically complete the clock name if it is unique. If there are multiple names, a list with possible matches is displayed.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Filtering Clock Domains

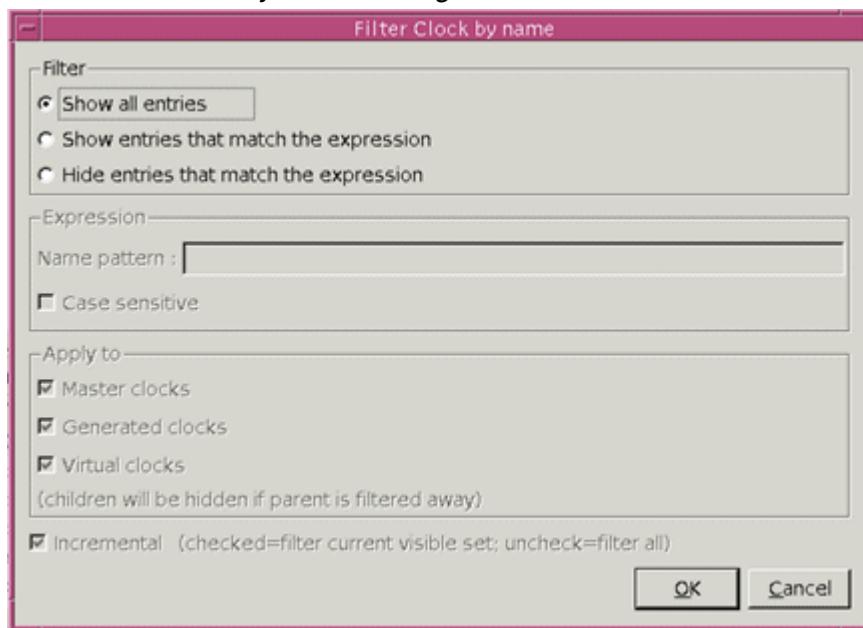
You can filter the clock domains to reduce the number of visible clocks by including or excluding clocks whose names match a certain pattern or type.

To filter the clocks in the clock analyzer,

1. Right-click in the clock tree view and choose Filter.

The Filter Clock by Name dialog box appears as shown in [Figure 18-30](#).

Figure 18-30 Filter Clock by Name Dialog Box



2. Select a filter option.

You can set the filter to either show or hide the clocks that match the filter criteria.

- To display just the clocks that match the filter criteria, select the “Show entries that match the expression” option.

If a parent clock does not match the criteria, the associated children are not displayed.

- To hide the clocks that match the filter criteria, select the “Hide entries that match the expression” option.

If a parent clock matches the criteria, the associated children are not displayed.

3. Enter a clock name or name pattern in the “Name pattern” box.

Use an asterisk (*) to match any string or a question mark (?) to match a single character. Patterns are matched using substring matching within the clock name. If a parent clock does not match the pattern, the associated children are not displayed.

4. Set options to specify the types of clocks that you want to filter.

You can filter master clocks, generated clocks, virtual clocks, or a combination of these clocks and their associated children. These options are all selected by default.

- To prevent filtering of clocks with derived clocks, deselect the “Master clocks” option.
- To prevent filtering of leaf-level derived clocks, deselect the “Generated clocks” option.
- To prevent filtering of generated clocks with their master clocks, deselect the “Virtual clocks” option.

5. Set other options as needed.

6. Click OK.

To remove the filtering and display all the clocks,

1. Right-click in the clock tree view and choose Filter.

The Filter Clock By Name dialog box appears.

2. Select the “Show all entries” option.

3. Click OK.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Locating Launch and Capture Clocks in the Clock Matrix

You can locate the next capture or launch clock in the clock matrix by using the arrow buttons beside the Launch or Capture boxes or the arrow keys on the keyboard. When a matrix cell is selected, the arrow buttons and arrow keys move the selection to the corresponding cell that contains a relationship and skip the matrix cells with no relationship.

- To move from one capture clock to the previous capture clock, click the Left Arrow button or press the Left Arrow key.
- To move from one capture clock to the next capture clock, click the Right Arrow button or press the Right Arrow key.

- To move from one launch clock to the previous launch clock, click the Up Arrow button or press the Up Arrow key.
- To move from one launch clock to the next launch clock, click the Down Arrow button or press the Down Arrow key.

Note:

If a clock is within a collapsed tree, it is not located because the clock analyzer performs the search only on visible clocks and does not automatically expand the tree.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Saving Clock Attribute or Clock Matrix Constraint Data

You can save the clock attribute data in the clock tree view or save the clock constraint data in the clock matrix view by exporting it from the clock analyzer to a comma separated value (CSV) format file.

To save the clock attribute data,

1. Click the Export Attributes button.

The Export Clock Attributes to File dialog box appears.

2. (Optional) If you want to save all the clock attributes, select the “All clock attributes” option.

By default, the clock analyzer saves only the attributes that are displayed in the clock tree view.

3. Specify the name and location for the file.

Specify a file name with the .csv extension if you want to open the file in Microsoft Excel.

4. Click Save.

To save the clock matrix constraint data,

1. Click the Export Matrix button.

The Export Clock Matrix to File dialog box appears.

2. Specify the name and location for the file.

Specify a file name with the .csv extension if you want to open the file in Microsoft Excel.

3. Click Save.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Clock Matrix Symbols and Colors

The GUI identifies the following types of clock crossings for paths with a launch and capture clock pair:

- Empty cell – No paths

No paths exist between the corresponding launch and capture clock except when the launch and capture clock are the same

- C – Constrained paths

Fully constrained paths exist between the launch and capture clocks of the selected matrix cell

- F – False paths

All paths between the selected launch and capture clock pair have been designated false paths

- P – Partially constrained paths

Only some paths between the selected launch and capture clock pair are designated false paths

Each cell represents a launch and capture clock pair. The cell is colored to visually show the relationship between the clocks. [Table 18-3](#) shows the background cell pattern and color that indicates the synchronous or asynchronous relationships between the launch and capture locks.

Table 18-3 Background Cell Pattern

Pattern	Description
	Diagonal orange lines indicates that the launch and capture clocks are from the same clock domains and asynchronous with respect to each other.
	Diagonal gray lines indicates that the launch and capture clocks are from different clock domains and asynchronous with respect to each other.
	Solid gray indicates that the launch and capture clocks are from different clock domains but synchronous with respect to each other.

Table 18-3 Background Cell Pattern (Continued)

Pattern	Description
	Solid light orange indicates that the launch and capture clocks are from the same clock domain and synchronous with respect to each other.
	Solid dark orange indicates that either the launch and capture clocks corresponding to a particular cell are the same clock.
	Crisscross pattern indicates that the launch and capture clocks are logically or physically exclusive.

See Also

- [Analyzing Clock Domains and Clock-to-Clock Relationships](#)

Examining Timing Paths and Design Logic in a Schematic

An instance schematic can show graphical representations of timing paths, selected design logic or fanin and fanout logic. You can use instance schematics to visually analyze timing and logic in the design and gather information that can help you to guide other analysis tasks.

Instance schematics display design objects (cell instances, pins, nets, ports, buses, bus rippers, and hierarchy crossings) in a flat, single-sheet schematic that can span multiple hierarchy levels. An instance can be a block (a hierarchical cell representing a subdesign) or a leaf cell.

To open an instance schematic,

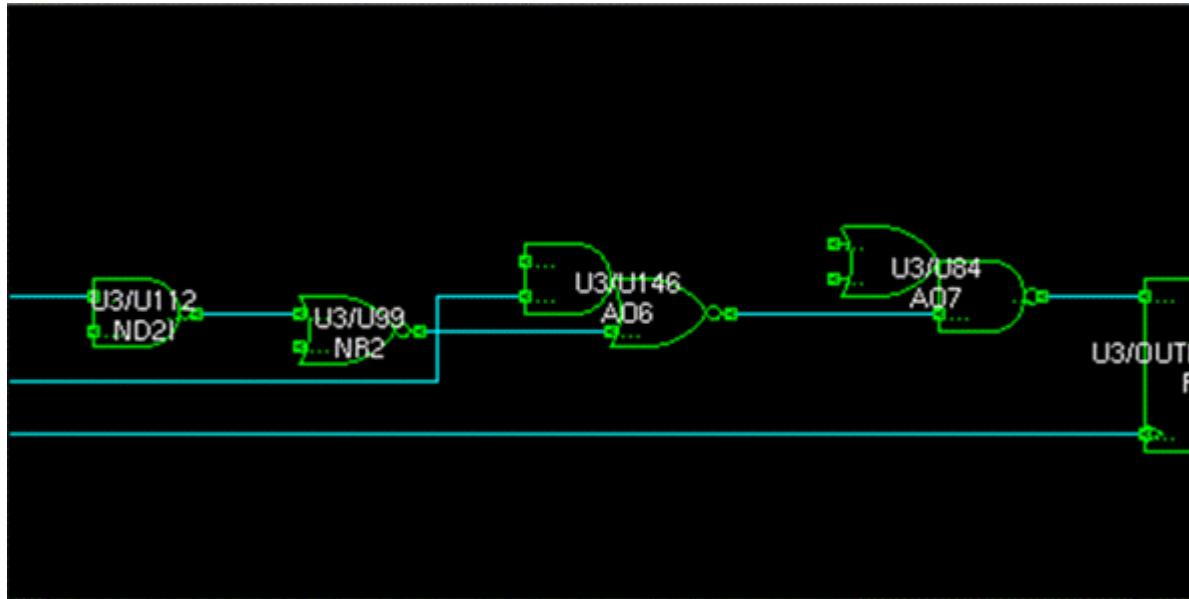
1. Select the design objects or timing path that you want to display in the schematic.

You can select hierarchical cells or design objects in the hierarchy browser or another schematic window. You can select timing paths interactively in a path slack histogram or the timing path table or by choosing **Select > Paths From/To/Through** and setting options in the **Select Paths** dialog box. You can select fanin or fanout logic by choosing **Select > Fanin/Fanout** and setting options in the **Select Fanin/Fanout** dialog box.

2. Click the button on the Schematic toolbar or choose **Schematic > New Schematic View**.

[Figure 18-31](#) shows an example an instance schematic.

Figure 18-31 Instance Schematic



If a schematic contains multiple levels of hierarchy, you can reorganize it so that objects are placed hierarchically with colored boundaries identifying each hierarchical block or cell. You can also move down or up in the hierarchy to display the contents of individual hierarchical cells.

To avoid tediously examining the progression of a signal across buffer and inverter chains or through unimportant blocks in an instance schematic, you can hide some objects by collapsing them into abstract metacells.

You can add or remove selected logic (cells, pins, ports, or nets) in an instance schematic. The objects are added or removed only in the active schematic view. The netlist is not changed and other schematic views are not affected.

You can also add fanin logic, fanout logic, or worst-case timing paths to a schematic, and you can control whether the additions appear in the active schematic view or in a new schematic view. You can

- Add the logic or paths to the schematic in the active schematic view
- Display only the selected objects and the additional logic or paths in the active schematic view
- Add the logic or paths to the schematic and display it in a new schematic view
- Display only the selected objects and the additional logic or paths in a new schematic view

You can reverse and reapply changes that you make in a schematic view, such as expanding or collapsing design logic or adding or removing selected objects.

For more information about working with instance schematics, see

- [Examining Hierarchical Cells](#)
- [Grouping Cells by Hierarchy](#)
- [Moving Down or Up the Design Hierarchy](#)
- [Displaying or Hiding Buffers and Inverters](#)
- [Displaying or Hiding Unconnected Macro Pins](#)
- [Expanding or Collapsing Buses](#)

See Also

- [Viewing and Modifying Schematics](#)
- [Adding or Removing Selected Logic](#)
- [Adding Timing Paths](#)
- [Adding Fanin and Fanout Logic](#)
- [Reversing and Reapplying Schematic Changes](#)
- [Schematic Menu Commands](#)

Examining Hierarchical Cells

By default, an instance schematic displays timing paths and design logic in a flat, single-sheet schematic that can span multiple levels of hierarchy. When you create an instance schematic that contains hierarchical cells, the cells initially appear as collapsed metacells.

You can expand hierarchy metacells to display the objects in the next hierarchy level down in the design hierarchy. If the hierarchical cell includes further levels of hierarchy, they initially appear as hierarchy metacells. You can also collapse the objects to display their parent hierarchical cell at the next level up in the design hierarchy.

To expand all the hierarchy metacells in the schematic,

- Choose Schematic > Expand > All Hierarchy.

To expand individual hierarchy metacells,

1. Select one or more hierarchical cells that you want to expand.
2. Click the  button on the Schematics toolbar or choose Schematic > Expand > Selected Objects.

Alternatively, you can double-click the metacells.

A hierarchy metacell expands to display the objects in the next hierarchy level. If it includes further levels of hierarchy, they appear as hierarchical metacells.

To collapse all the hierarchical cells in the schematic

- Choose Schematic > Collapse > All Hierarchy.

To collapse individual hierarchical cells,

1. Select objects in the hierarchical cells that you want to collapse.

2. Click the  button on the Schematics toolbar or choose Schematic > Collapse > Selected Hierarchy By Parent.

For a set of objects that have a common hierarchical parent, the hierarchy metacell is similar to a hierarchical cell but has a thicker line width and a different color.

Figure 18-32 Expanded and Collapsed Views of a Design Hierarchy



See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Grouping Cells by Hierarchy](#)
- [Moving Down or Up the Design Hierarchy](#)

Grouping Cells by Hierarchy

By default, a schematic that contains multiple levels of hierarchy displays the objects in a flat, single-sheet schematic with input ports on the left and output ports on the right. Hierarchy crossings (diamond shapes) indicate where nets traverse a level of hierarchy.

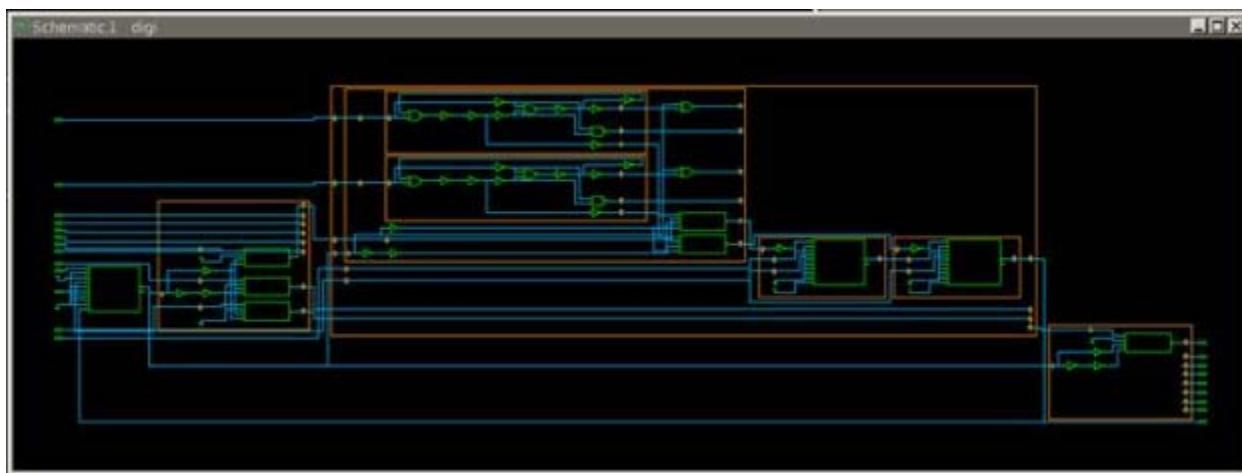
You can reorganize an instance schematic hierarchically so that objects are placed hierarchically, which puts objects that share the same hierarchical parent near each other. The schematic displays an orange rectangular boundary for the top-level design each hierarchical cell.

To group cells hierarchically in the active schematic view,

- Choose Schematic > Show Logical Hierarchy in the Schematic window.

[Figure 18-33](#) shows an example of an instance schematic with the objects grouped hierarchically.

Figure 18-33 Schematic Organized by Logic Hierarchy



See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Hierarchical Cells](#)

- [Moving Down or Up the Design Hierarchy](#)
 - [Displaying or Hiding Buffers and Inverters](#)
 - [Displaying or Hiding Unconnected Macro Pins](#)
 - [Expanding or Collapsing Buses](#)
-

Moving Down or Up the Design Hierarchy

You can traverse the design hierarchy within a schematic view by moving down into any block (subdesign) at the next lower level of the hierarchy or by moving up (from a subdesign) to the hierarchical cell (parent) at the next higher level of the hierarchy.

To move into a hierarchical cell at the next level down in the design hierarchy,

1. Select the hierarchical cell.

You can select the cell in the schematic view or select the cell in the hierarchy browser and then click the title bar in the schematic view window to make it the active view.

2. Click the  button on the Schematics toolbar or choose Schematic > Move Down.

Alternatively, you can double-click the hierarchical cell in the design schematic.

To move up to the next level in the design hierarchy,

- Click the  button on the Schematic toolbar or choose Schematic > Move Up.

Note:

When you move down or move up in the design hierarchy, the GUI changes the design name displayed on the tab for the schematic view window.

See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
 - [Examining Hierarchical Cells](#)
 - [Grouping Cells by Hierarchy](#)
-

Displaying or Hiding Buffers and Inverters

By default, an instance schematic that contains multiple levels of hierarchy displays design objects and timing paths in a flat, single-sheet schematic with hierarchy crossings (diamond shapes) showing where nets traverse a level of hierarchy. Each timing path consists of the objects (cells, pins, and nets) that make up the path. A path can include long chains of buffers or inverters and multiple hierarchy crossings.

To avoid tediously examining the progression of a signal across buffer and inverter chains or through unimportant blocks, you can hide some objects by collapsing them into abstract metacells. You can hide buffer and inverter chains or buffer and inverter trees.

To hide all the buffer and inverter chains in the schematic,

- Choose Schematic > Collapse > All Buffers/Inverters/Crossings By Chain.

To hide all the buffer and inverter trees in the schematic,

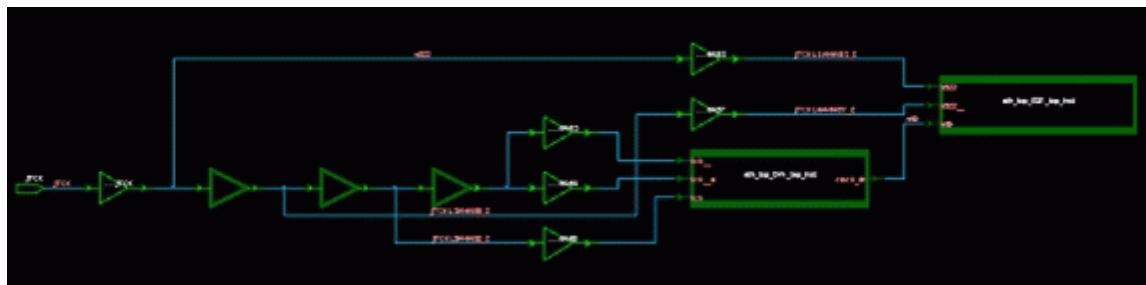
- Choose Schematic > Collapse > All Buffers/Inverters/Crossings By Tree.

To hide individual buffers, inverters, and hierarchy crossings,

- Select the objects that you want to hide.
- Click the  button on the Schematic toolbar or choose Schematic > Collapse > Selected Buffers/Inverters/Crossings.

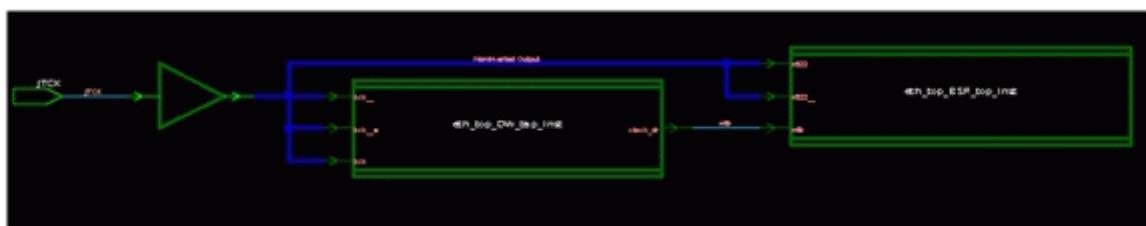
When a buffer or inverter chain or a buffer or inverter tree results in a noninverted output, the metacell is similar to a buffer but has a thicker line width and darker color.

Figure 18-34 Collapsed View of Buffer Chain Metacells



When a buffer or inverter chain or a buffer or inverter tree results in an inverted output, the metacell is similar to an inverter but has a thicker line width and darker color.

Figure 18-35 Collapsed View of Buffer Tree metacells



When a buffer or inverter tree that results in both noninverted and inverted outputs, the metacell combines the appearance of both an inverter and a buffer. The symbol has both inverted and noninverted outputs with the loads of the chain connected to the appropriate polarity output.

Buffer and inverter metacells expand to display the path containing the buffers, inverters, and hierarchy crossings.

To expand all the buffer and inverter metacells in the schematic,

- Choose Schematic > Expand > All Buffers/Inverters/Crossings.

To expand individual buffer and inverter metacells,

1. Select the metacells that you want to expand.

2. Click the  button on the Schematics toolbar or choose Schematic > Expand > Selected Objects.

Alternatively, you can double-click a metacell.

See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Hierarchical Cells](#)
- [Displaying or Hiding Unconnected Macro Pins](#)
- [Expanding or Collapsing Buses](#)

Displaying or Hiding Unconnected Macro Pins

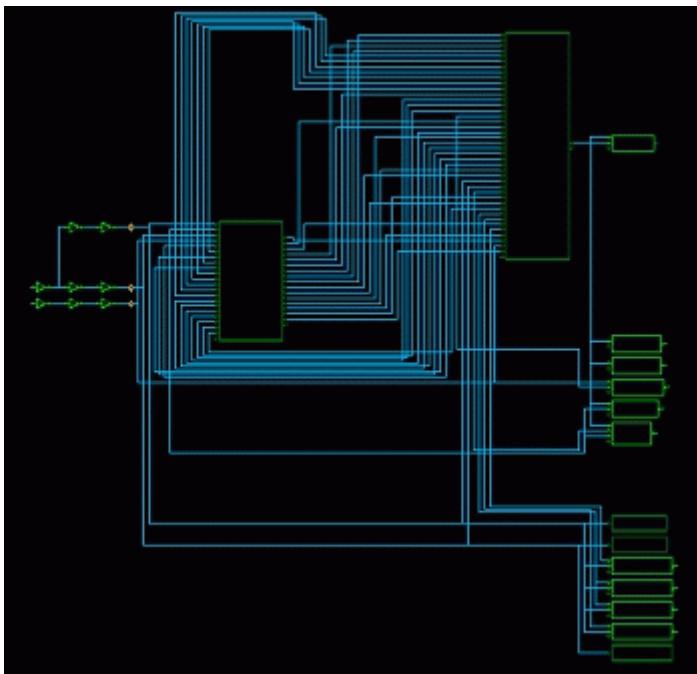
To simplify a schematic and hide unimportant details, you can hide the unconnected pins of macro cells by collapsing them into metapins. The hidden pins are represented by metapins of the same type: in, out, in/out, and bidirectional. A metapin is similar to a design pin but has a thicker line width and a different color.

To hide unconnected macro pins,

- Choose Schematic > Collapse > All Unconnected Pins.

Unconnected pins are represented by metapins of the same type: in, out, in/out, and bidirectional. A metapin is similar to a design pin but has a thicker line width and a different color. [Figure 18-36](#) shows an example.

Figure 18-36 Collapsed View of Metapins



To expand unconnected metapins,

- Choose Schematic > Expand > All Unconnected Pins.

See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Hierarchical Cells](#)
- [Displaying or Hiding Buffers and Inverters](#)
- [Expanding or Collapsing Buses](#)

Expanding or Collapsing Buses

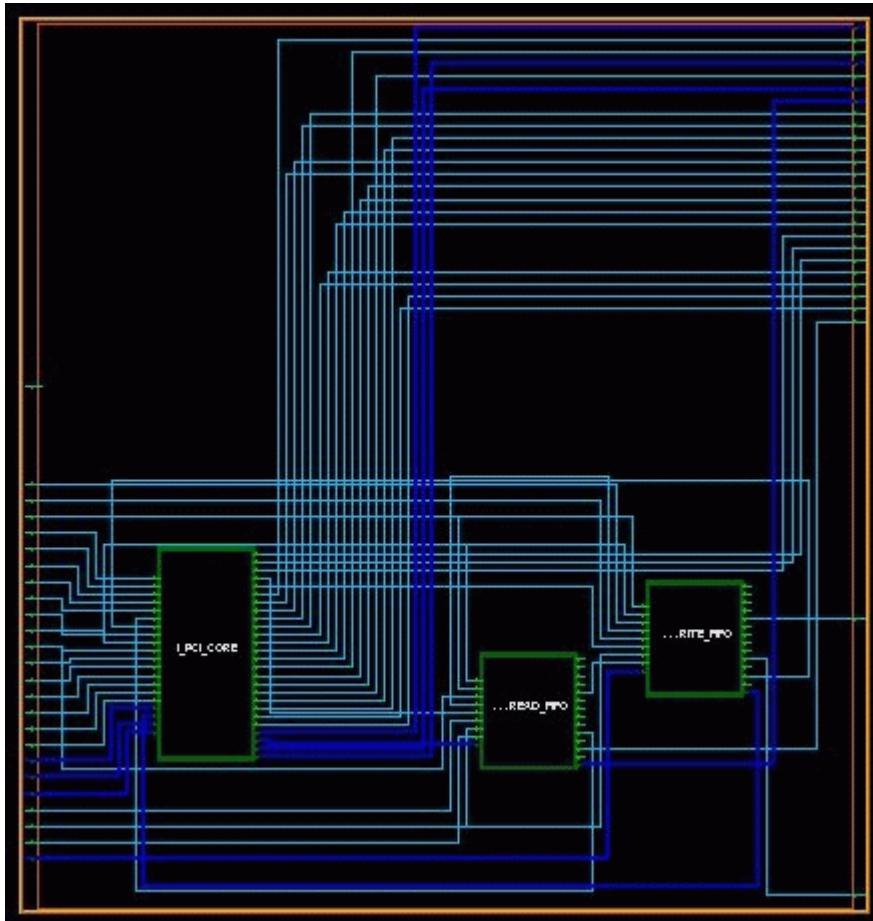
You can save space in a schematic by collapsing nets that are members of a bus and their associated pins. This allows you to focus on the nets that are of interest while keeping the overall context in view as you debug timing problems.

To collapse bus pins and nets into buses,

- Choose Schematic > Collapse > All Bussed Pins/Nets.

The collapsed nets are rendered in a darker blue with a thicker line width, and the pins are rendered in a darker green, as shown in [Figure 18-37](#).

Figure 18-37 Collapsed View of Bus Nets and Pins



You can view the names of a collapsed net or pin in an InfoTip or by using the Query tool.

To expand bus nets and pins,

- Choose Schematic > Expand > All Bussed Pins/Nets.

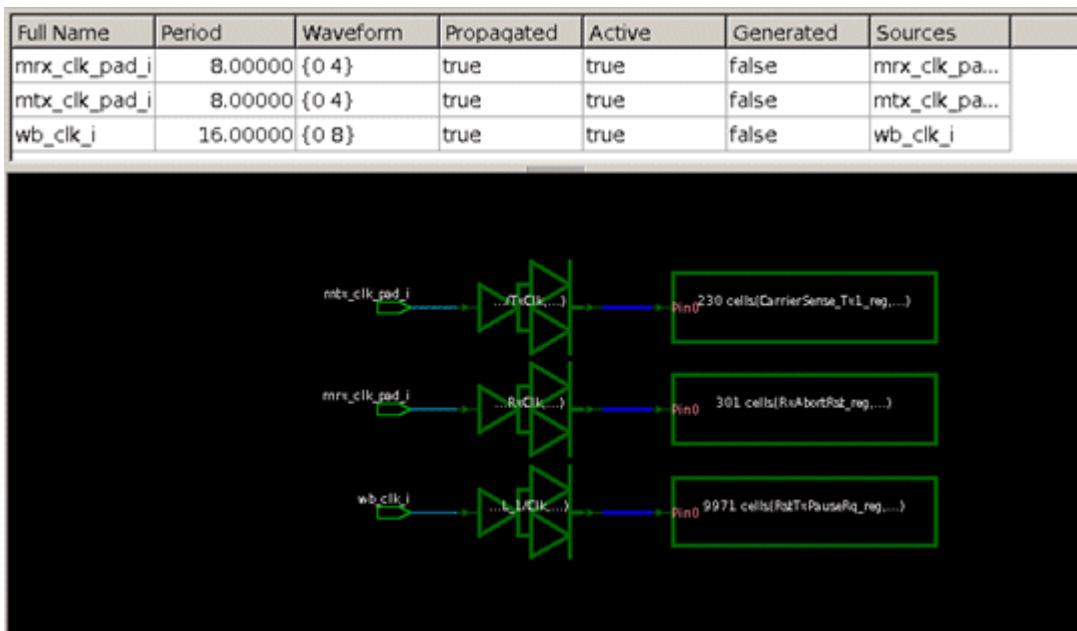
See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Hierarchical Cells](#)
- [Displaying or Hiding Buffers and Inverters](#)
- [Displaying or Hiding Unconnected Macro Pins](#)

Examining Clock Paths in a Schematic

There are several ways to access the Clock Schematic window. To access the Clock Schematic window, from the GUI choose Clock > Clock Schematic for Selected Clocks or choose Clock > Clock Schematic for Unclocked Pins. You can also access the Clock Schematic window from the Clock Analyzer window by right-clicking and choosing Schematic of Selected Clocks. Another way to access the clock schematic is to use the Schematic menu from the Abstract Clock Graph window. [Figure 18-38](#) shows an example of the Clock Schematic window.

Figure 18-38 Clock Schematic Window



For ease of use, clock schematics collapse buffer and inverter clouds in the clock tree to reduce the size of the schematic. Sequential load cells that share the same functional clock signals are also collapsed together. To help with the analysis, the propagation paths of one or more clocks can be highlighted by color.

Metaobjects are representations of multiple objects of the same type. The following terms are used to identify metaobject symbols in clock schematics:

- Metabuffer – A cluster of buffers and inverters collapsed together. This is a single symbol with one or two outputs: inverted with a circle at the output pin or noninverted without a circle at the output pin, or both.
- Metanet – A cluster of nets collapsed together because the connected objects were collapsed.

- Metahierarchy – A cluster of multiple hierarchy symbols collapsed together.
- Metaflop – A cluster of sequential load cells collapsed together.

For more information about working with clock schematics, see

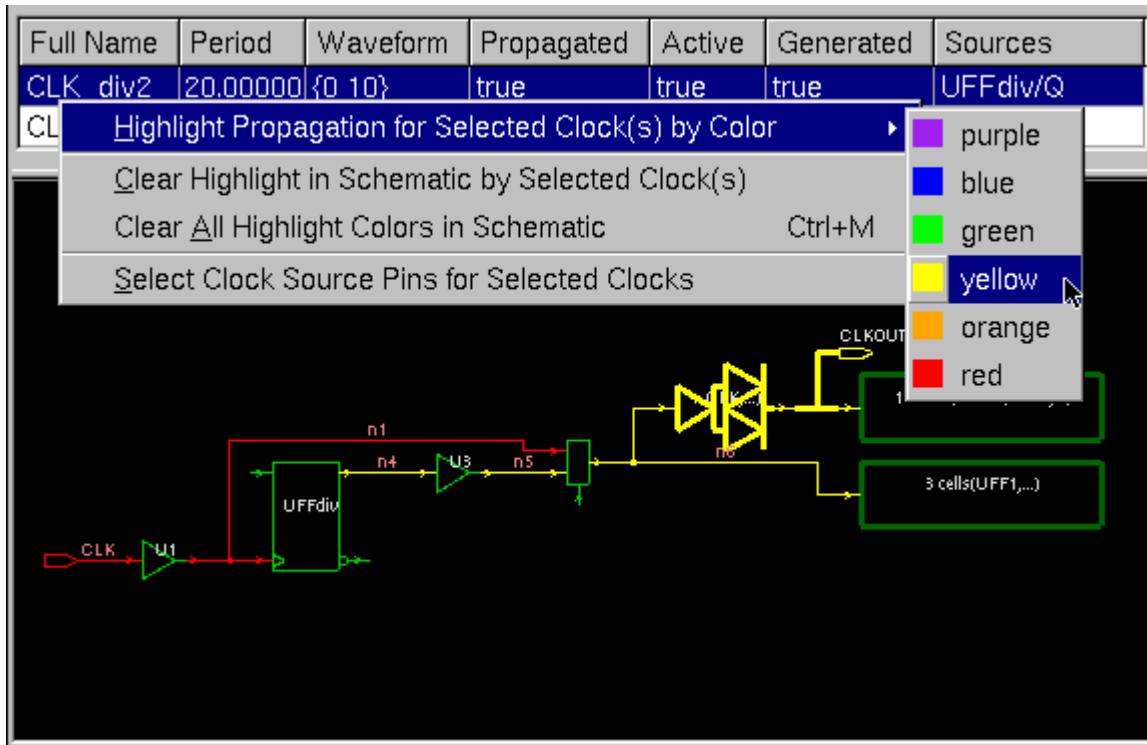
- [Highlighting Clock Propagation Paths](#)
- [Displaying or Hiding Buffers and Inverters](#)
- [Querying and Selecting Metaobjects](#)
- [Collapsing Clock Gating Cells](#)

See Also

- [Viewing and Modifying Schematics](#)
- [Clock Menu Commands](#)
- [Schematic Menu Commands](#)

Highlighting Clock Propagation Paths

When multiple clocks are shown in the clock schematic, you can highlight the propagation path of selected clocks in different colors by choosing commands on the pop-up menu. Right-click and choose Highlight Propagation for Selected Clock(s) by Color > *color_name*, where *color_name* is purple, blue, green, yellow, orange, or red. You can use this feature to highlight selected clocks in the color of your choice. For an example of clock highlighting, see [Figure 18-39](#).

Figure 18-39 Clock Highlighting

When a net is in the propagation of multiple highlighted clocks, the last color applied is kept. You can also clear clock highlighting by clock using “Clear Highlight in Schematic by Selected Clock(s)” or clear all highlighted colors by using “Clear All Highlight Colors in Schematic.”

Normal schematic tools can also be used in the clock schematic. You can select the load or driver of selected objects by choosing the Select > Driver of Selected or Select > Load of Selected option. You can also select the entire fanin or fanout of selected objects by right-clicking and choosing Select Fanin Cone or Select Fanout Cone.

See Also

- [Examining Clock Paths in a Schematic](#)
- [Adding Fanin and Fanout Logic](#)

Displaying or Hiding Buffers and Inverters

For ease of use, you can reduce the size of a schematic by collapsing buffer and inverter clouds in a clock tree. Sequential load cells that share the same functional clock signals are also collapsed together.

To expand selected metaobjects,

1. Select the metaobjects that you want to expand.
2. Choose Clock > Expand > Expand Selected.

Alternatively, you can expand metaobjects that are not selected.

To expand metaobjects that are not selected,

1. Select any metaobjects that you do not want to expand.
2. Choose Clock > Expand > Expand Unselected.

For example, you can obtain a fully unexpanded schematic by choosing Clock > Expand > Expand Unselected on the clock schematic with nothing selected.

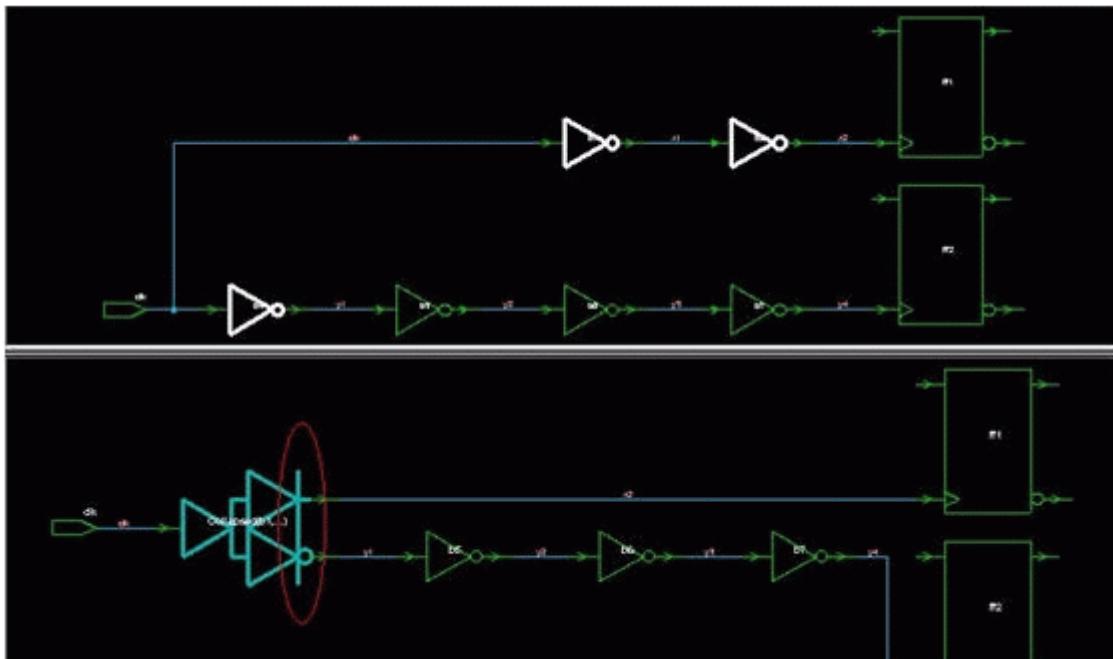
To collapse selected objects into metaobjects,

1. Select the objects in the expanded schematic.
2. Choose Clock > Collapse > Collapse Selected.

In addition, you can selectively expand the first or last fanin or fanout logic level from selected metaflops by choosing Clock > Expand > Expand First Level or Clock > Expand > Expand Last Level.

[Figure 18-40](#) shows how the schematic changes when you expand metaobjects or collapse objects. The top image shows the expanded schematic, and the bottom image shows the schematic after only the selected items are collapsed.

Figure 18-40 Collapse of Selected Buffer or Inverter Cells



Note:

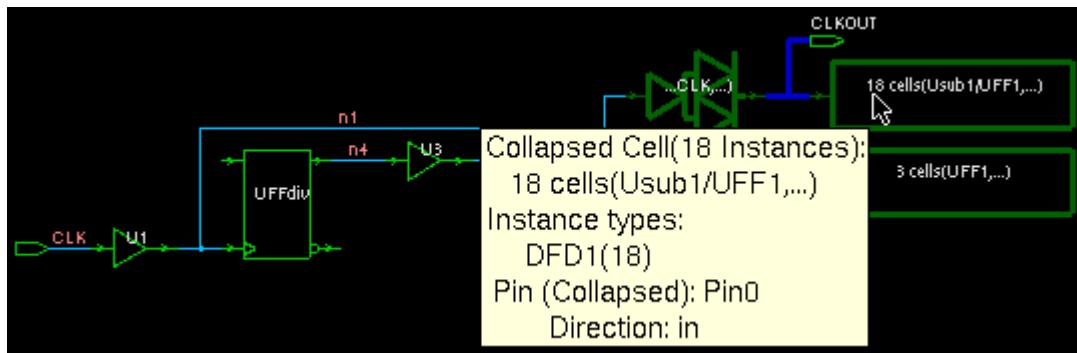
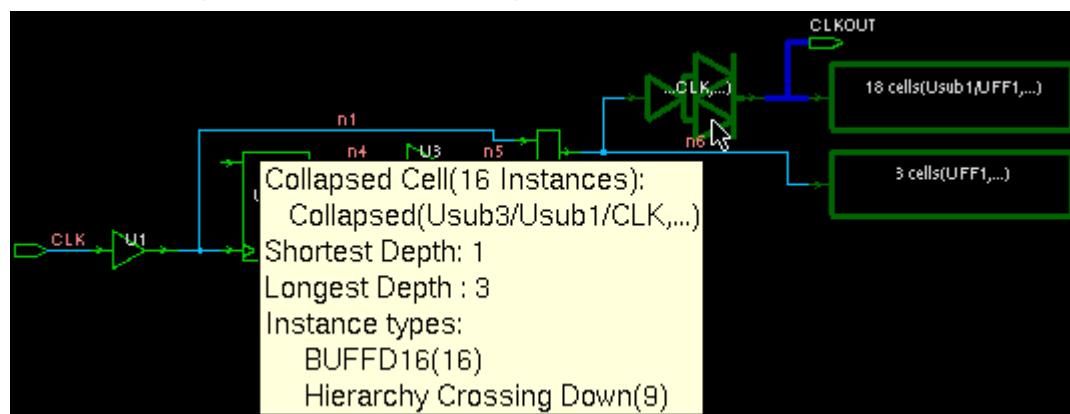
You can add logic to the schematic by using the Add Fanin/Fanout tool. This method can be useful for exploring the side inputs of control logic cells.

See Also

- [Examining Clock Paths in a Schematic](#)
- [Adding Fanin and Fanout Logic](#)

Querying and Selecting Metaobjects

You can get metaobject information through InfoTips as shown in [Figure 18-41](#) and [Figure 18-42](#).

Figure 18-41 Collapsed Cells With InfoTip*Figure 18-42 Collapsed Buffers With InfoTip*

Selected metaobjects are visible in the Selection List dialog box (choose Select > Selection List). When a metaobject is selected, all of the items it contains are selected implicitly and appear in the selection list. Metaobject names start with “Collapsed” and contain their children.

Note that simplified clock schematic functions are enabled only when a clock is selected in a clock table. These functions are not available if a clock is selected in a separate schematic or hierarchy browser view.

See Also

- [Examining Clock Paths in a Schematic](#)

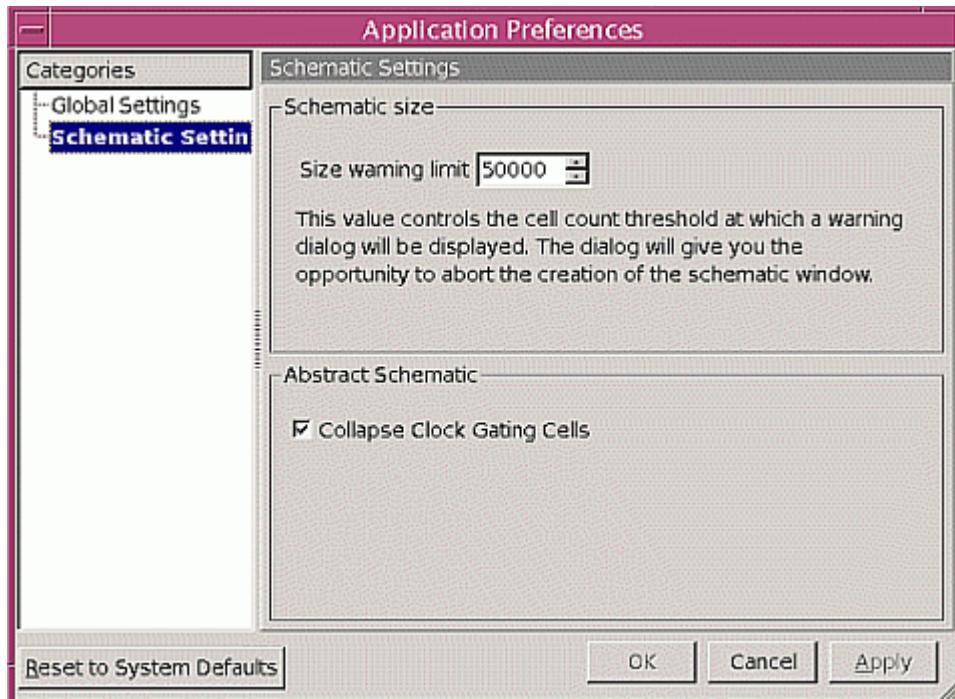
Collapsing Clock Gating Cells

Clock schematics allow you to collapse clock-gating cells into a metabuffer. This feature allows the tool to consider clock-gating cells to be extended buffer cells for the purpose of buffer-tree collapsing.

To enable the collapsing of clock-gating cells,

1. Choose View > Preferences to open the Application Preferences dialog box.
2. Click Schematic Settings in the Categories tree.
3. Select the Collapse Clock Gating Cells option as shown in [Figure 18-43](#).

Figure 18-43 Schematic Settings in Application Preferences Dialog Box



For more information about using the Application Preferences dialog box, see [Setting GUI Preferences](#).

See Also

- [Examining Clock Paths in a Schematic](#)

Viewing and Modifying Schematics

Schematics show graphical representations of timing paths, design objects, or clocks in your design. When you create a schematic, the GUI displays the schematic view in a new schematic window.

The GUI provides two types of schematic views: instance schematics and clock schematics.

- An instance schematic shows design objects and timing paths and lets you traverse the hierarchy of the design.

You can create an instance schematic for the top-level design, a hierarchical cell, selected design objects, or selected timing paths. When you create a schematic that includes timing paths, the schematic shows the cells and nets in each path.

- A clock schematic shows the clocks in a design and lets you traverse the clock hierarchy.

Note:

Abstract clock graphs are similar to schematics but have a higher level of abstraction.

You can add or remove selected objects (cells, ports, or nets) to a schematic. The objects are added or removed only in the active schematic view. The netlist is not changed and other schematic views are not affected. You can also add fanin logic, fanout logic, or timing paths to a schematic, and you can control whether the additions appear in the active schematic view or in a new schematic view.

You can print an image of the active schematic view by choosing File > Print and setting options in the Print dialog box. Alternatively, you can save the image PDF file or a PostScript file.

For more information about working with schematic views, see

- [Schematic Window](#)
- [Selecting or Highlighting Objects By Name](#)
- [Highlighting Schematics By Using Filtering Rules](#)
- [Annotating Schematic Pins and Ports](#)
- [Adding or Removing Selected Logic](#)
- [Adding Timing Paths](#)
- [Adding Fanin and Fanout Logic](#)
- [Reversing and Reapplying Schematic Changes](#)
- [Printing Images of Schematics, Abstract Clock Graphs, and Timing Waveforms](#)

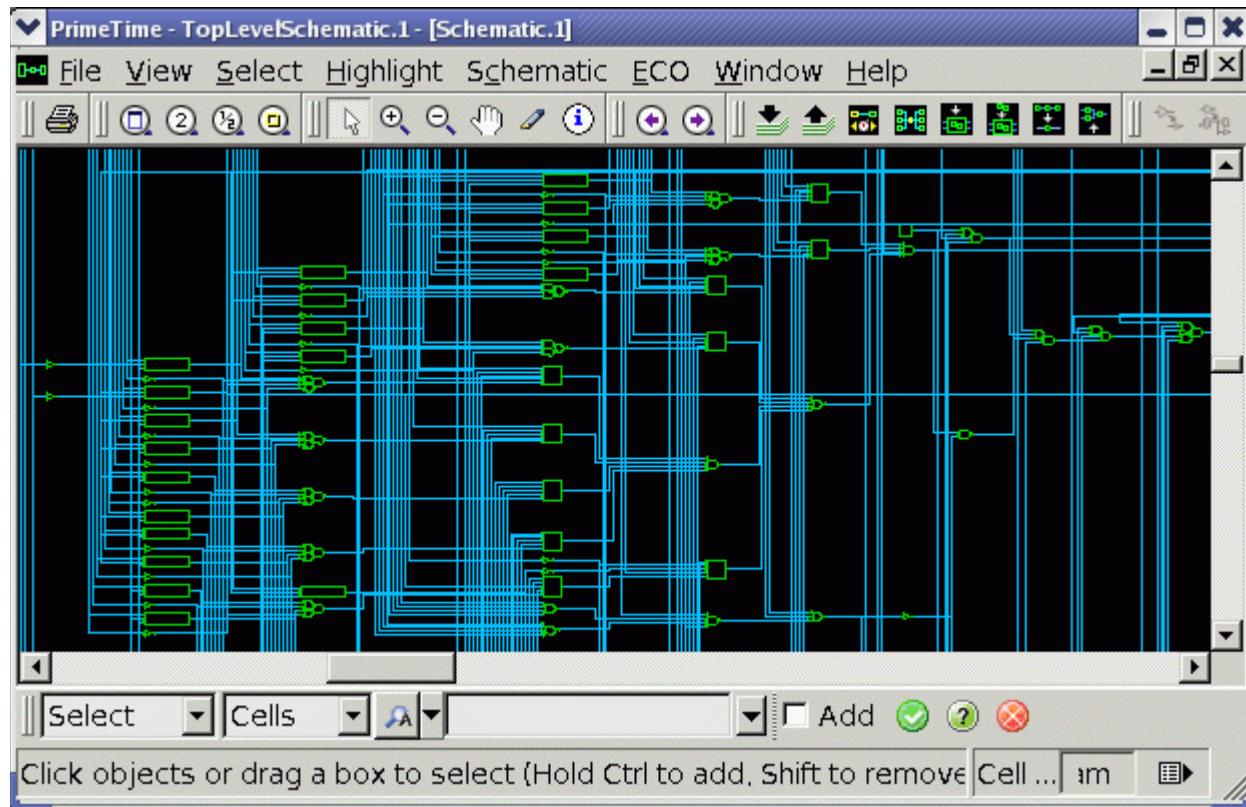
See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Examining Clock Paths in a Schematic](#)

Schematic Window

The schematic window contains its own menu bar, toolbars, and status bar. [Figure 18-44](#) shows an example of an instance schematic in a schematic window with the schematic view maximized in the workspace area.

Figure 18-44 Schematic Window



Initially, the full schematic is visible. You can modify the viewing range and scale by scrolling and resizing the schematic view window and by clicking buttons on the View Zoom/Pan toolbar or choosing commands zoom and pan on the View menu. Choose View > Zoom to access these commands.

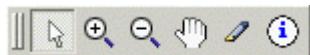
Figure 18-45 View Zoom/Pan Toolbar



You can also use the zoom and pan interactive mouse tools to magnify and traverse the view interactively. You can activate these tools by clicking buttons on the View Tools toolbar or choosing commands on the View menu. Choose View > Mouse Tools to access these

commands. In addition, you can use the arrow keys to scroll vertically or horizontally through the view.

Figure 18-46 View Tools Toolbar



Note:

Text does not appear in a schematic view when it is below a certain size in pixels.

Magnify the view if necessary to see object names in an instance schematic.

You can use interactive mouse tools to select, highlight, and query objects interactively in a schematic view. Mouse tools control the actions performed by the GUI when you click or drag the pointer in a graphic view such as a schematic or layout view.

- To select objects interactively in a schematic view, click them or drag the pointer around them with the Selection tool, which is the default interactive mouse tool.

Selected objects are displayed in white with a thicker line width. Objects that you select in a schematic view are also selected in other views, such as another schematic view or the hierarchy browser.

- To highlight objects in a schematic view, click them or drag the pointer around them with the Highlight tool. To activate this tool, click the button on the View Tools toolbar or choose View > Mouse Tools > Highlight Tool.

You can also highlight objects by selecting them and choosing a command on the Highlight menu. For example, to highlight the critical path (the default path reported by the `report_timing` command), choose Highlight > Critical Path. Objects that you highlight in a schematic view are also highlighted in other graphic views, such as another schematic view or an abstract clock graph view.

- To view information about an object (such as a cell, pin, or net) in a schematic view, hold the pointer over the object. The object information about the object appears in an InfoTip, which is a small, temporary box that displays information about the object at the pointer location.

- To view more detailed information about an object, click the object with the Query tool.

The object information appears on the Query panel. To activate this tool, click the button on the View Tools toolbar or choose View > Mouse Tools > Query Tool.

See Also

- [Viewing and Modifying Schematics](#)
- [View Settings Panel](#)
- [View Menu Commands](#)

- [Highlight Menu Commands](#)
- [Schematic Menu Commands](#)

Selecting or Highlighting Objects By Name

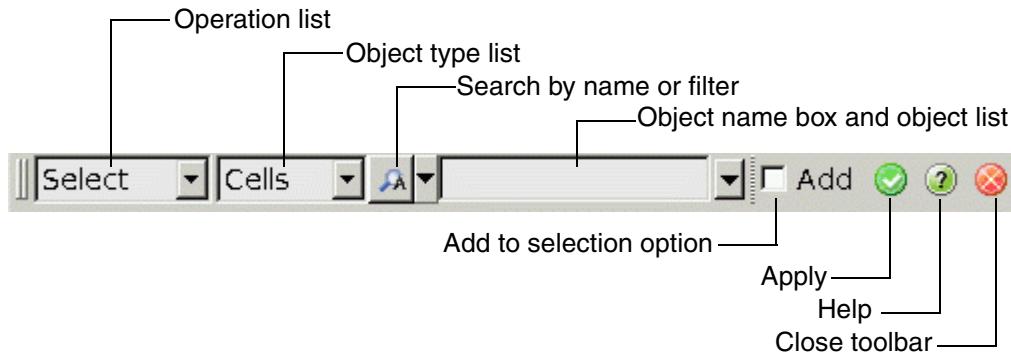
In a schematic window or an abstract clock graph window, you can select or highlight cells, nets, pins, ports, or clocks by name in the current design.

To select or highlight an object,

1. Choose Select > By Name Toolbar.

[Figure 18-47](#) shows the Select By Name toolbar that appears above the status bar at the bottom of the window.

Figure 18-47 Select By Name Toolbar



2. Select an operation option: Select or Highlight.
3. (Optional) Select an object type option: Cells, Nets, Pins, Ports, or Clocks.
- The tool locates the objects that match the criteria.
4. (Optional) Select one of the following search criteria options.

- Search by name

Select this option to search the design for an object that matches the specified name or name pattern. Type part of a name and press Tab to enable the tool to complete the name. If multiple names match the text, a list appears. Use the Up and Down Arrow keys to scroll through the list. Click Enter to select an item.

Type multiple object names separated by a blank space. If an object name contains a space, enclose the name in braces { } to treat it as a single name.

- Search by filter

Select this option to search the design for all objects where the specified filter expression is located. Type part of the filter expression and press Tab to enable the tool to complete the matching value. If multiple values match the text, a list appears.

Note:

Invalid name or filter search patterns are identified by a light red background.

5. Type an object name or select a name in the list of objects.

If the tool locates multiple objects that match the text, the name completion list shows the first 15 names in a list that you can scroll through.

6. (Optional) Select the Add option if you want to add the objects to the current selection.

This option is deselected by default, which means the objects replace the current selection.

7. Click apply to search for objects that match the specified criteria.

See Also

- [Highlighting Schematics By Using Filtering Rules](#)
- [Viewing and Modifying Schematics](#)
- [Examining Clock Paths in an Abstract Clock Graph](#)

Highlighting Schematics By Using Filtering Rules

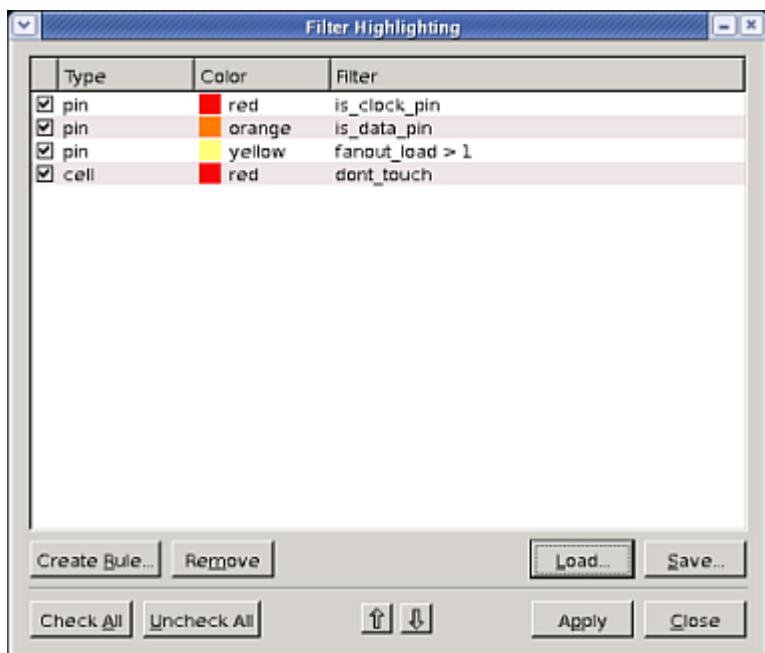
You can highlight elements according to filtering rules that you define.

To open the Filter Highlighting dialog box,

- Choose Highlight > Filter Highlighting.

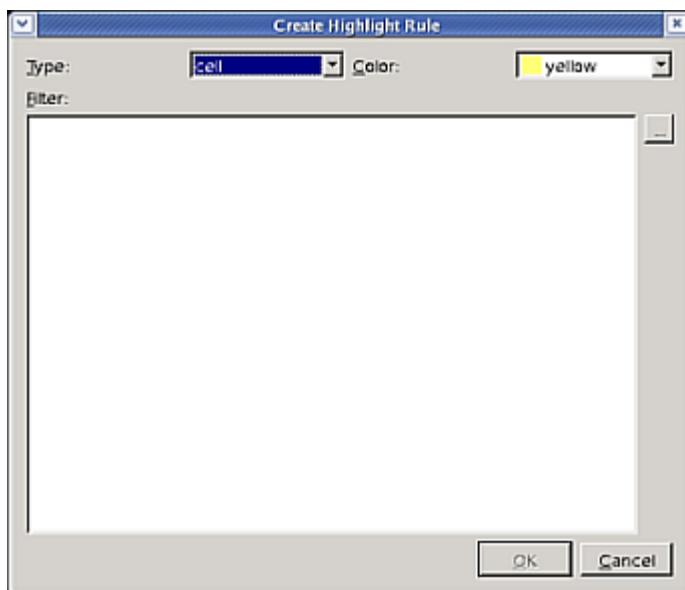
[Figure 18-29](#) shows the Filter Highlighting dialog box after several rules have been defined.

Figure 18-48 Filter Highlighting Dialog Box



In the Filter Highlighting dialog box, you can specify the active highlighting rules and the order in which they are applied. To create a new filtering rule, click Create Rule to open the Create Highlight Rule dialog box, shown in [Figure 18-49](#).

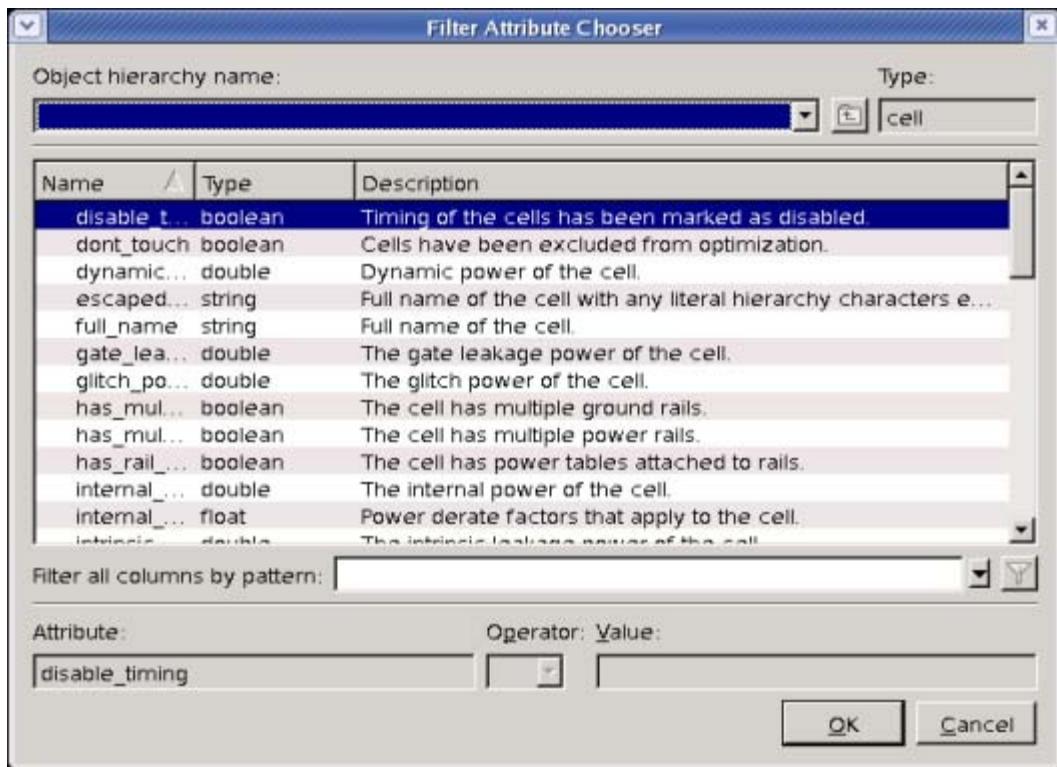
Figure 18-49 Create Highlight Rule Dialog Box



The Create Highlight Rule dialog box allows you to specify the type of object to be highlighted and the highlighting color. You can also specify the filter expression by clicking

the browse button (...) to open the Filter Attribute Chooser dialog box, shown in [Figure 18-50](#).

Figure 18-50 Filter Attribute Chooser Dialog Box



See Also

- [Selecting or Highlighting Objects By Name](#)
- [Viewing and Modifying Schematics](#)

Annotating Schematic Pins and Ports

By default, schematic annotation on pins and ports uses the `case_value` attribute. The schematic annotation feature allows you to annotate pin attributes in an instance schematic. The tool provides the `SchemAnnotAttr` attribute group for pin annotation. You can update this attribute group with other valid attributes. For example, you can use both the name and direction as annotation text for a pin.

The supported attributes are netlist object attributes within the design context. The timing path related context is unsupported. If you specify multiple valid values in the attribute list, ensure each annotation string is delimited by a comma.

You can update the `SchemAnnotAttr` attribute group by using the Attribute Group Manager dialog box. For details, see [Modifying Attribute Groups](#).

See Also

- [Viewing and Modifying Schematics](#)
- [Printing Images of Schematics, Abstract Clock Graphs, and Timing Waveforms](#)

Adding or Removing Selected Logic

You can add or remove selected logic (cells, pins, ports, or nets) in an instance schematic.

- To add logic, select the objects you want to add, make the schematic the active view, and choose Schematic > Add Selected.
- To remove logic, select the objects you want to remove, and then choose Schematic > Delete Logic.

The objects are added or removed only in the active schematic view. The netlist is not changed and other schematic views are not affected.

See Also

- [Adding Timing Paths](#)
- [Adding Fanin and Fanout Logic](#)
- [Viewing and Modifying Schematics](#)

Adding Timing Paths

You can add the timing paths to the schematic in the active schematic view. You can either add the paths to the active view or open a new schematic view. You can add either the timing paths with the worst slack in the current design or the paths with the worst slack from, to, or through selected cells, nets, pins, or ports.

To add timing paths to a schematic,

1. Click the  button on the Schematics toolbar, or choose Schematic > Add Paths From/Through/To.
The Add Paths to Path Schematic dialog box appears.
2. (Optional) Specify the startpoints, throughpoints, or endpoints to generate a histogram for paths from, through, or to specific inputs, registers, or outputs.

You can specify the objects by selecting them or entering their names. For example, if you enter a single pin name in the From box (and leave the Through and To boxes blank), the tool finds all the paths that originate at the specified pin and plots the resulting slack values in the histogram.

To specify the currently selected pins, ports, nets, or clocks in the From, Through, or To box, select an object type option and click the Selection button.

If you want to browse the design hierarchy to find the objects you need to specify, click the browse button next to the From, To, or Through box. This opens the Object Chooser dialog box shown in [Figure 18-7](#).

3. Set options to select the delay type, control the distribution of slack values, and configure the histogram plot.

The default selection is based on maximum delay times. You can select a delay type based on minimum delay times. Alternatively, you can select a delay type based on maximum or minimum delay times on rising or falling clock edges.

You can restrict the number of paths by specifying the number of worst paths per endpoint (Nworst paths box), the total maximum number of path slacks to be reported (Max paths box), or both. You can also specify just the paths in a particular path group (Group name box).

4. Select other options as needed.
5. Set options to control where the tool displays the timing paths.

You can control whether the paths are added to the active schematic view or displayed in a new schematic view. If you display the timing paths in the active schematic view, you can control whether the paths are added to or replace the paths and logic in the path schematic. If you open a new schematic view, you can control whether the paths and logic from the path schematic in the active schematic view are included in the new path schematic.

6. Click OK.

See Also

- [Adding Fanin and Fanout Logic](#)
- [Viewing and Modifying Schematics](#)

Adding Fanin and Fanout Logic

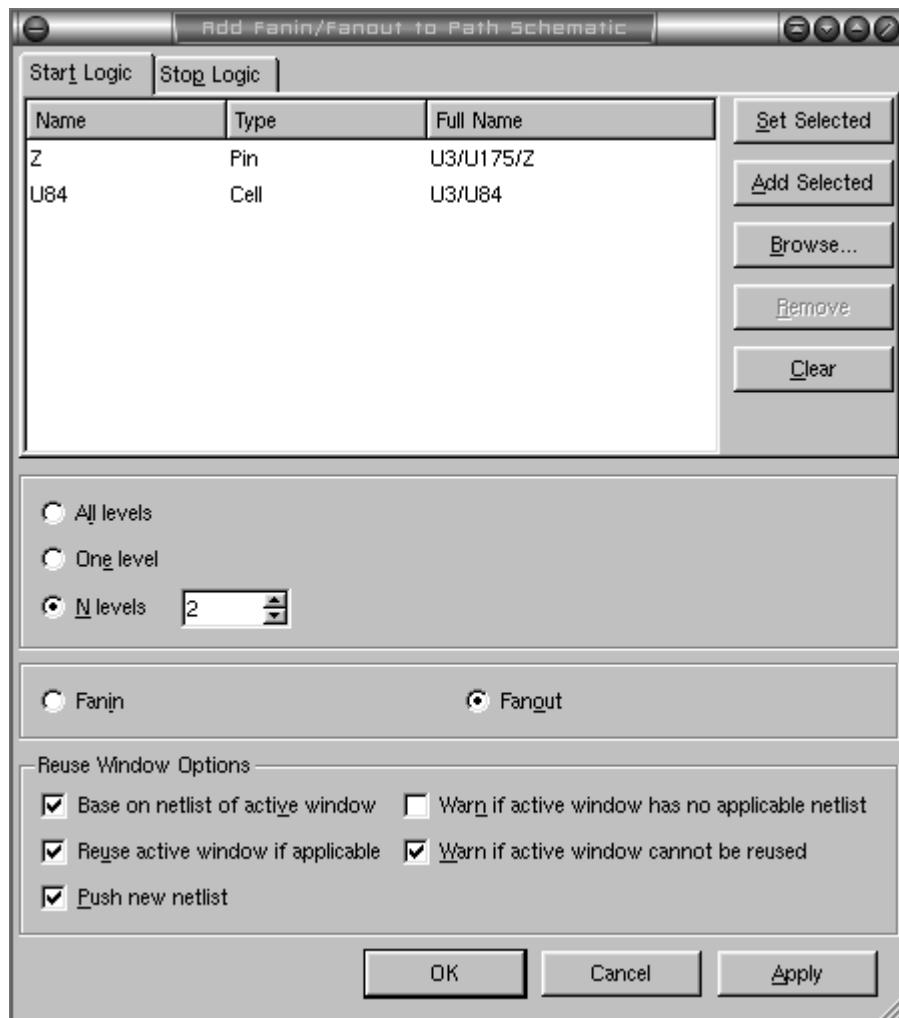
You can increase the scope of a schematic by adding fanin or fanout logic to the schematic in the active schematic view. You can add one or more levels of fanin or fanout logic for a selected cell, pin, port, or net. You can either add the fanin or fanout logic to the active view or open a new schematic view.

To add fanin or fanout logic levels for an object in the active schematic view,

1. Select one or more cells, pins, ports, or nets in the schematic.
2. Click the  button on the Schematics toolbar or choose Schematic > Add Fanin/Fanout.

The Add Fanin/Fanout to Path Schematic dialog box appears as shown in [Figure 18-51](#).
The name, full name (including the path from the top-level design), and type of each selected object appear in the Start Logic list.

Figure 18-51 Add Fanin/Fanout to Path Schematic



3. Select either the Fanin option (to add fanin logic) or the Fanout option (to add fanout logic).

Fanout is selected by default.

4. Set the number of logic levels.

You can add all levels, one level, or a specific number of levels.

5. (Optional) Edit the Start Logic list as needed.

To replace the objects in the list with other objects, select the objects and click the Set Selected button. To add objects to the list, select the objects and click the Add Selected button. To remove objects from the list, select their names and click the Remove button. To remove all objects from the list, click the Clear button.

You can also add objects to the list by clicking the Browse button to browse for them in the Object Chooser dialog box.

6. (Optional) If necessary, click the Stop Logic tab and specify object names in the Stop Logic list.

7. Set other options as needed.

- If you do not want the fanin or fanout cones to stop at sequential logic objects, deselect the “Stop at sequential cells” option.
- Set reuse window options to control whether the fanin or fanout logic is added to the active schematic view or displayed in a new schematic view.

8. Click OK.

See Also

- [Adding Timing Paths](#)
- [Viewing and Modifying Schematics](#)

Reversing and Reapplying Schematic Changes

You can reverse and reapply changes that you make in a schematic view, such as expanding or collapsing design logic or adding or removing selected objects.

To reverse changes in a schematic,

1. Choose Schematic > Back.
2. Repeat step 1 to reverse the next most recent change.

To reapply changes in a schematic,

1. Choose Schematic > Forward.
2. Repeat step 1 to reapply the next most recent change.

See Also

- [Viewing and Modifying Schematics](#)

Printing Images of Schematics, Abstract Clock Graphs, and Timing Waveforms

You can print an image of a schematic view, an abstract clock graph view, or a waveform diagram in the path inspector, or save the image in a file that you can print later. You can save the image as a PDF file or a PostScript file.

To print or save the contents of the active view,

1. Make sure the active view displays the schematic or clock graph information that you want to print.
2. Click the  button on the File toolbar or choose File > Print to open the Print dialog box.
3. Select a print destination in the Name list.
 - To print the image, select a printer name.
 - To save the image in a PDF file, select the “Print to file (PDF)” option.
 - To save the image in a PostScript file, select the “Print to file (Postscript)” option.
4. If you selected a PDF or PostScript file print destination, type the path and file name in the “Output file” box.

Alternatively, you can click the browse button and type or select the file name in the dialog box that appears.

5. (Optional) Click the Properties button and set printer options as needed in the dialog box that appears.
6. (Optional) Click the Options button to expand the dialog box, and set options as needed.
 - To display the print range and copy options, click the Copies tab.
 - To display the duplex printing and color mode options, click the Options tab.
7. Click Print.

See Also

- [Viewing and Modifying Schematics](#)
- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Viewing Timing Waveforms](#)

Examining Timing Path Details in a Data Table

Path data tables provide detailed information about timing paths in your design. You can examine path details, such as attribute values, generate histograms based on path attributes, and select paths for further analysis with other tools in a schematic, a timing report, or the path inspector.

You can select timing paths in another tool, such as a histogram or the path analyzer, and load them into a new path data table.

To display selected timing paths in a path data table,

- Choose Timing > New Timing Path Table View for Selected.

The GUI opens a new data table view and displays the path names and other path information in the table.

Alternatively, you can load timing paths in a new path data table by specifying path criteria and setting report options. For details, see [Loading Timing Paths in a New Path Data Table](#).

The path data table view consists of a timing path table and a button bar.

- The table displays information about the paths, including their startpoint and endpoint names, path groups, and timing attribute values.
- The button bar provides commands you can use to reload the paths, configure the table columns, save the path details in a file, generate path data histograms, and access other timing analysis tools.

By default, the table lists the paths by increasing order of their slack values. You can use the Up Arrow and Down Arrow keys to scroll up or down in the table. If some of the text in a column is missing because the column is too narrow, you can hold the pointer over the column to display the text in an InfoTip.

You can sort the table by the alphanumerical order of the contents in any column and adjust the widths of individual columns.

- To sort a column, click the column heading.
Click again if you want to reverse the order.
- To resize a column, drag the right edge of the column to the left or right.

You can also filter the paths based on a character string or regular expression that you define. For details, see [Filtering Tables and Lists](#).

You can examine specific paths or path details side-by-side with the path data table by clicking buttons on the button bar.

- To view paths in an instance schematic, select the paths and click Schematic.
- To view paths in the path inspector, select the paths and click inspector.
- To view a timing report for a path, select the path, click Report, and set report options in the Report Timing for Selected Path dialog box.
- To generate a path data histogram based on the values in a specific column, click the Histogram button and set options in the Table Histogram dialog box.

You can also click buttons on the button bar to

- Configure the table by hiding, displaying, or reordering columns
- Save the path data in a comma separated value (CSV) format file
- Reload the timing paths

Note:

If the window is too narrow to display all the buttons, you can click the  button and choose a command on the menu that appears.

For more information about working with path data tables, see

- [Loading Timing Paths in a New Path Data Table](#)
- [Creating Path Data Histograms](#)
- [Viewing Timing Reports](#)
- [Configuring Data Table Columns](#)
- [Saving the Timing Path Data](#)
- [Reloading Timing Paths in a Path Data Table](#)

See Also

- [Recalculating Timing Paths](#)

Loading Timing Paths in a New Path Data Table

When you open a new path data table, you can set timing report options to specify the path criteria, control the type of analysis the tool performs, and limit the maximum number of paths. For more information about these options, see the man page for the `get_timing_paths` command.

You can set the criteria for individual timing paths by specifying path startpoints, endpoints, or throughpoints. If you specify more than one startpoint or endpoint, all paths that start at any of the startpoints or end at any of the endpoints (and meet the other criteria) are considered.

To open a path data table view and load timing paths,

1. Choose Timing > New Timing Path Table View.

The New Timing Path Table View dialog box appears.

2. (Optional) Set the criteria for the paths that you want to view.

By default, the tool loads the paths with the worst slack values based on options that you set in step 3.

3. Set the timing report options as needed.

- To control the type of analysis that the tool performs, select a path delay type.
- To control the maximum number of paths in the data table, you can set the maximum number of paths through an endpoint, set the maximum number of paths in a path group, and select a path group.

4. Click OK.

See Also

- [Examining Timing Path Details in a Data Table](#)
- [Reloading Timing Paths in a Path Data Table](#)

Creating Path Data Histograms

When you examine timing paths in a path data table, you can create path data histograms to analyze the distribution of certain types of path details. Each histogram you create shows the distribution of values for the paths in the data table. You can create a histogram for values from any visible column that contains numeric data.

To display a path data histogram,

1. Click the Histogram button in a path data table.

The Table Histogram dialog box appears.

2. Select the histogram data type in the Column list.

For timing path data, the default column is Slack.

Note:

Some of the labels and option settings in the dialog box might change depending on which option you select in the Column list.

3. Set the histogram configuration and appearance options as needed.

4. Click OK.

For details about setting histogram configuration and appearance options, see [Viewing and Customizing Histograms](#).

See Also

- [Examining Timing Path Details in a Data Table](#)

Viewing Timing Reports

You can select a timing path in a path data table and view a timing report about the path. You can specify the report format and set other timing report options.

To view a timing report for a selected path,

1. Select the path in the data table.
2. Click the Report button to open the Report Timing for Selected Path dialog box.
3. Select a report format option in the “Path report type” list.

The choices are Full, Short, Summary, End, Full clock, and Full clock expanded. The default is Full.

4. Set other report and output options as needed.
5. Click OK.

The GUI displays the timing report in a new report view.

For more information about the report options, see the man page for the `report_timing` command

See Also

- [Examining Timing Path Details in a Data Table](#)
- [Viewing Reports](#)

Saving the Timing Path Data

You can save the timing path data by exporting it from the path data table to a comma separated value (CSV) format file.

To save the timing path data,

1. Click the Export button below the path data table at the bottom of the window.
The Export to File dialog box appears.
2. Specify the name and location of the CSV file.
Specify a file name with the .csv extension if you want to open the file in Microsoft Excel.
3. Click Save.

See Also

- [Examining Timing Path Details in a Data Table](#)
-

Reloading Timing Paths in a Path Data Table

You can reload timing paths in a path data table by setting timing report options and optionally setting the criteria for specific paths. For more information about these options, see the man page for the `get_timing_paths` command.

To reload timing paths in a path data table,

1. Click the Load Paths button on the button bar at the bottom of the data table window.
The Get Timing Paths dialog box appears.
2. (Optional) Set the criteria for the paths that you want to view.
By default, the tool loads the paths with the worst slack values based on options that you set in step 3.
3. Set the timing report options as needed.
 - To control the type of analysis that the tool performs, select a path delay type.
 - To control the maximum number of paths in the data table, you can set the maximum number of paths through an endpoint, set the maximum number of paths in a path group, and select a path group.
4. Click OK.

See Also

- [Examining Timing Path Details in a Data Table](#)
- [Loading Timing Paths in a New Path Data Table](#)

Inspecting Timing Path Elements

Use the path inspector to analyze various aspects of one or more timing paths. You can

- View path profiles that represent the relative delay and slack contributions of various path components
- Examine timing report information that includes a path summary, path element data, and path slack details
- Perform additional analysis tasks such as selecting the path, highlighting the path, or finding text in the report information

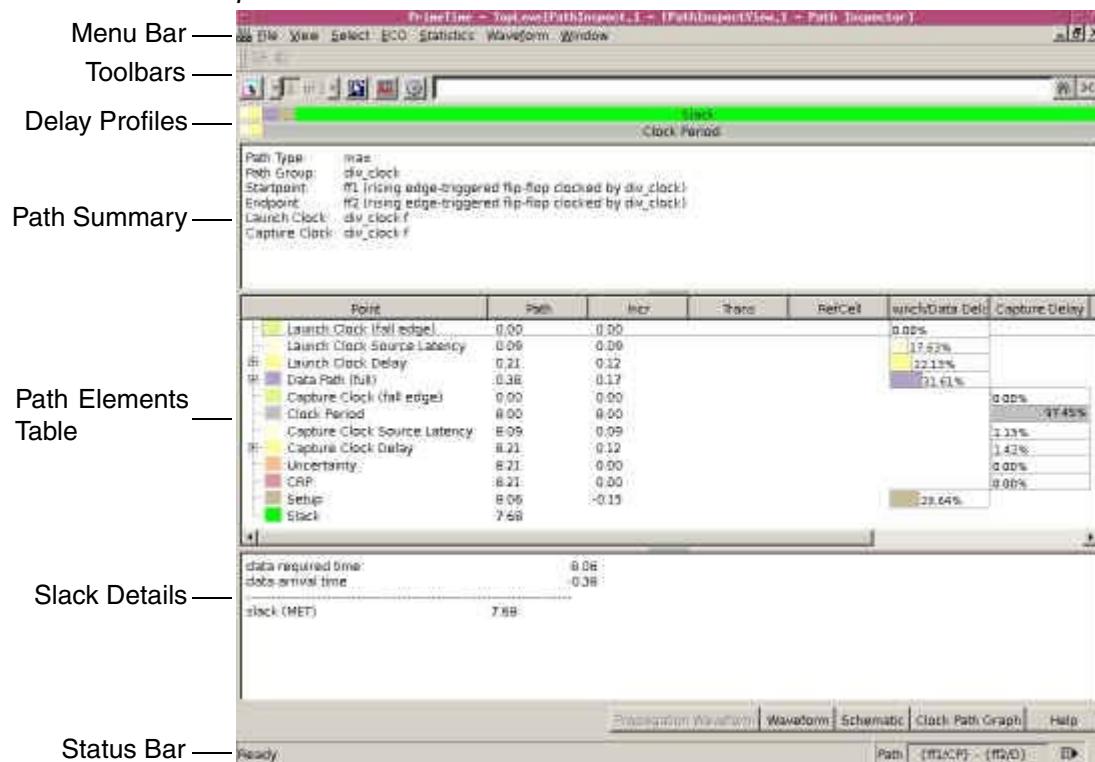
You can load multiple paths and display the information for each path sequentially by moving between the paths. You can also select and load different paths at any time.

The Path Inspector window is an application window with its own menu bar, toolbar, a status bar, and two views: a path inspector view and a waveform view. The path inspector view displays path delay profiles and timing report details.

Note:

If you are using the PrimeTime SI tool, the path inspector also provides a propagated waveform view. For details about using this view, see [Viewing Propagated Waveforms for Recalculated Paths](#).

[Figure 18-52](#) shows an example of the path inspector view in a path inspector window.

Figure 18-52 Path Inspector Window

The path delay profiles display the relative contributions of various components to the delay and slack calculations of the timing path. The width of the bar for each component represents its relative delay contribution. The bars are color-coded to match elements in the path elements table.

To view the delay contribution of a component in the path profile, hold the pointer over the component. An InfoTip appears showing the element name and the delay value. Viewing these components can help you to locate the reasons for timing path failures.

The timing report details appear in three panes: path summary, path elements table, and slack details.

- The path summary provides information about the path attributes that identify the path, such as its path group, startpoint and endpoint, and delay type.
- The path elements table displays information about the elements of the path and their contributions to the path delay and slack calculation.
- The slack details include information about the required and actual arrival times and the slack values.

You can copy the summary and paste them into text files, export the tables to text files, customize the element tables, and generate delay calculation reports for selected cells or nets.

The path inspector toolbar displays the number of timing paths that you have loaded.



If you load multiple paths, you can

- Move to the previous path by clicking the Left Arrow button
- Move to the next path by clicking the Right Arrow button

You can search for text in the timing report information. You can also select text in the path summary or the slack details and copy it for use in another tool, such as a text editor.

The path elements table displays information about timing path elements, such as the data path, the clock period, and clock uncertainty. You can expand some elements by double-clicking the element row or clicking the expansion button (plus sign) beside the element name.

Use the path elements table to view data for

- A clock, pin, or net design object associated with a path element
- The launch clock path
- The capture clock path
- The data path

You can view information about a path element in an InfoTip by holding the pointer over its row in the table. You can also select and highlight path elements in the table.

- To highlight selected path elements, right-click and choose Highlight Object.
- To remove highlighting from selected path elements, right-click and choose Clear Highlight from Object.

To adjust the width of a column in the path elements table, drag the right edge of the column to the left (to decrease the width) or right (to increase the width).

You can configure the path elements table by clicking the button and setting options in the Configure the Path Inspector dialog box. For details, see [Configuring the Path Inspector](#).

You can select or highlight the path you are currently inspecting.

- To select the path and make it the current selection in the tool, click the button on the path inspector toolbar.

- To highlight the path with the current highlight color, click the  button on the path inspector toolbar.

You can also view timing waveforms in the path inspector and display the path in a new schematic window.

- To view timing waveforms in the waveform view, click the Waveform button above the status bar at the bottom of the path inspector window.
- To view propagated waveforms in the propagated waveform view, click the Propagation Waveform button above the status bar at the bottom of the path inspector window.

Propagated waveforms are available only for recalculated timing paths.

To display the path in a new schematic view,

- Click the Schematic button above the status bar at the bottom of the path inspector window.

For more information about working with the path inspector, see

- [Loading Paths Into a Path Inspector Window](#)
- [Finding Text in the Path Inspector](#)
- [Viewing Timing Waveforms](#)
- [Configuring the Path Inspector](#)
- [Saving the Path Element Data](#)

See Also

- [Recalculating Timing Paths](#)
- [Viewing Propagated Waveforms for Recalculated Paths](#)

Loading Paths Into a Path Inspector Window

You can load multiple paths and display the information for each path sequentially by moving between the paths. You can also select and load different paths at any time.

To open the path inspector,

1. (Optional) Select one or more timing paths that you want to inspect.

You can select the paths when you open the path inspector, or you can open the path inspector first and then load the timing paths.

2. Choose Timing > Inspect Normal Path or Timing > Inspect Recalculated Path.

Alternatively, if you can select the paths in a path data table, you can click the Inspector button.

A new path inspector window appears. For assistance using the path inspector, click the Help button above the status bar at the bottom of the window.

See Also

- [Inspecting Timing Path Elements](#)

Finding Text in the Path Inspector

You can search for text in the timing report information. You can also select text in the path summary or the slack details and copy it for use in another tool, such as a text editor.

To search for text,

1. Enter the text in the text box on the path inspector toolbar.
2. Click the  button on the path inspector toolbar.

The tool colors the text with a pale blue background everywhere it occurs and selects the first occurrence. Repeat this step to select successive occurrences of the text.

To clear the search coloring, click the  button.

To select and copy text,

1. Select the text you want to copy by dragging the pointer over it, or right-click and choose Select All.
2. Right-click and choose Copy.

See Also

- [Inspecting Timing Path Elements](#)

Viewing Timing Waveforms

You can view timing waveforms for the path you are inspecting.

To display timing waveforms in the path inspector,

- Click the Waveform button above the status bar at the bottom of the path inspector window.

The path inspector switches to the waveform view and displays the timing waveforms.

You can view information about a waveform segment by holding the pointer over it. The information appears in an InfoTip.

You can display or hide input pin and output pin waveforms. These waveforms are hidden by default.

- To display or hide input pin waveforms, right-click and choose In Pin.
- To display or hide output pin waveforms, right-click and choose Out Pin.

When you need to return to the path inspector view, click the PathInspectorView tab.

See Also

- [Inspecting Timing Path Elements](#)
- [Viewing Propagated Waveforms for Recalculated Paths](#)

Configuring the Path Inspector

You can control how much information the path elements table displays for the data path element by choosing a format option. In addition, you can configure the path elements table by setting options in the Configure the Path Inspector dialog box.

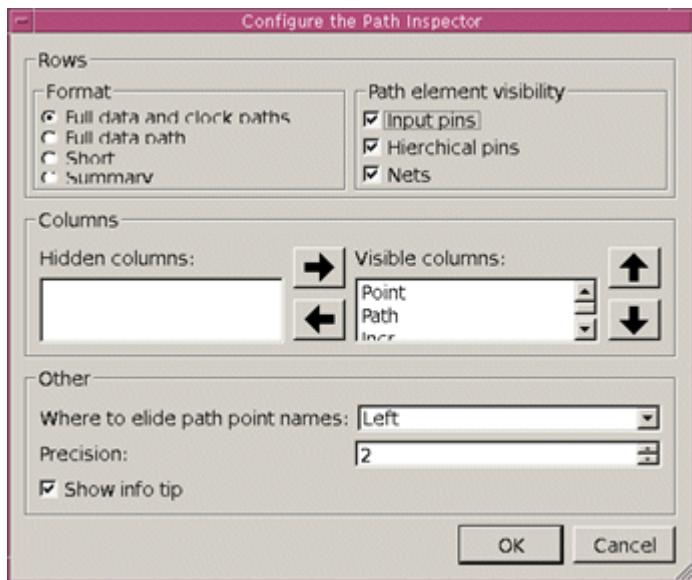
To reformat the path element table,

1. Right-click in the table and choose Format to open the Format menu.
2. Choose a format option.
 - To display full data and clock path details, select the “Full data and clock paths” option.
This option is the default. Note that clock path details might not be available for some paths.
 - To display full data path details but not clock path details, select the “Full data path” option.
 - To display only the startpoint and endpoint of the data path, select the Short option.
 - To hide the path elements and slack panes and display just the report summary, select the Summary option.

The “Full data and clock paths” option is selected by default.

[Figure 18-53](#) shows the Configure the Path Inspector dialog box.

Figure 18-53 Configure the Path Inspector Dialog Box



To configure the path inspector,

1. Click the button on the path inspector toolbar.

The Configure the Path Inspector dialog box appears.

2. Select a row format option.

This option is equivalent to choosing a format option in the path inspector window.

- The “Full data and clock paths” option corresponds to the `report_timing -path_type full_clock` or `report_timing -path_type full_clock_expanded` command.

This format displays the full data path and the full clock paths associated with the timing path, if any, in the path elements section. Whether the clock path is fully expanded with the clock path between a primary clock and a related generated clock depends on how the timing path being inspected was queried. The Path Inspector window shows the clock paths only if you queried the loaded timing path by choosing the “Full data and clock paths” option.

- The “Full data paths” option corresponds with the `report_timing -path_type full` command.
- The Short option corresponds with the `report_timing -path_type short` command.

- The Summary option is similar to the `report_timing -path_type summary` command.

However, the path inspector displays the textual header and the end summary sections with no path elements section. You might find this useful to quickly iterate over a fairly large number of loaded timing paths to locate a specific path to inspect.

3. Set the path visibility options to display or hide elements in the path elements table.

When a full path is displayed, the timing path is shown with a collapsed parent node indicating data, launch clock, or capture clock that can be expanded to show the full timing path elements. Select the options to include rows for that particular data. To exclude the data, deselect the options.

- To hide input pin data and display only driver pin data, deselect the “Input pins” option.
- To hide hierarchical pin data and display only leaf-cell pin data, deselect the “Hierarchical pins” option.
- To hide net data and display only pin data, deselect the Nets option.

These options are all selected by default, which means that the input pin, hierarchical pin, and net data appears in the path elements table.

4. Display or hide columns in the path elements table.

All the columns are displayed by default.

- To hide one or more columns, select the column labels in the “Visible columns” list and click the Left Arrow button.
- To display one or more columns, select the column labels in the “Hidden columns” list and click the Right Arrow button.
- To reorganize columns, select the columns you want to move left or right in the table and click the Up Arrow button or the Down Arrow button.

Repeat this step as needed to configure the columns.

5. Set other options as needed.

- To control how the path element table shortens element names that are too long to fit in the Name column, select an option in the “Where to elide path point names” list.

The choices are Left, Right, and Middle. For example, suppose a pin is named “abcde” but the column is only wide enough for three characters. If you select the Right option, the name is displayed as “abc....” If you select the Left option, the name is displayed as “...cde.” If you select the Middle option, the name is displayed as “ab...e.”

- To set the precision for floating point numbers in the table, enter a value in the Precision box.
 - To enable or disable InfoTips in the path element table, select or deselect the Show info tip option.
6. Click OK.

See Also

- [Inspecting Timing Path Elements](#)

Saving the Path Element Data

You can save the path element data by exporting it from the path inspector to a comma separated value (CSV) format file.

The expanded Data, Launch, and Capture path elements are exported to the same CSV file. If you want to avoid saving the detailed element data for specific path elements, do not expand these elements in the table.

To save the path element data,

1. Choose File > Export CSV Data in the path inspector window.
The Export Path Inspector CSV to File dialog box appears.
2. Specify the name and location of the file.
Specify a file name with the .csv extension if you want to open the file in Microsoft Excel.
3. Click Save.

See Also

- [Inspecting Timing Path Elements](#)

Viewing Object Attributes

You can view attribute values for design objects (cells, pins, ports, nets, or clocks) by selecting the objects and viewing the selection list, by querying the objects in a schematic or abstract clock graph view, or by viewing object attributes in the Properties dialog box or an attribute data table. You can also view, modify, create, and remove attribute groups by using the Attribute Group Manager dialog box. For information about performing these tasks, see

- [Viewing the Current Selection](#)
- [Querying Objects](#)

- [Viewing Object Properties](#)
- [Managing Attribute Groups](#)
- [Examining Object Attributes in a Data Table](#)

Viewing the Current Selection

To get a summary report on the current selection,

- Choose Select > Query Selection.

The report appears in the console log view and in the pt_shell terminal window.

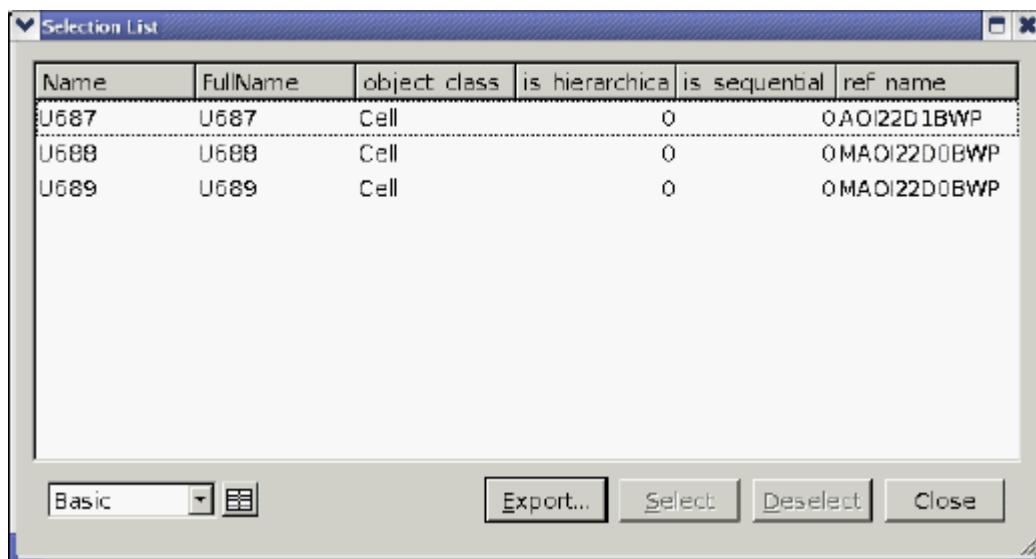
To see a detailed list of the currently selected objects,

- Click the  button on the status bar or choose Select > Selection List.

This opens the Selection List dialog box.

[Figure 18-54](#) shows an example of a selection list in this dialog box.

Figure 18-54 Selection List Dialog Box



You can deselect objects (and remove their names from the list) by

- Selecting names in the list and clicking the Select button to deselect all other selected objects
- Selecting names in the list and clicking the Deselect button to deselect those objects

You can also filter the selection list, limiting it to information based on a character string or regular expression that you define. For details, see [Filtering Tables and Lists](#).

To control which attributes appear in the selection list,

- Select an attribute group name in the list at the bottom-left corner of the dialog box.
This list is available only when the objects in the selection list all have the same object type.

By default, the Selection List dialog box displays values for the attributes in the Basic attribute group. You can modify the contents of some predefined attribute groups and create or modify custom attribute groups by using the Attribute Group Manager dialog box.

To open the Attribute Group Manager dialog box,

- Click the  button in the Selection List dialog box.

For more information about attribute groups, see [Managing Attribute Groups](#).

If you want to save the object or timing path information in the selection list, you can export the entire list to a text file. The file is formatted with each object or path on a separate line and the column data delimited by commas.

To save the selection list information in a text file,

1. Click the Export button to open the Export to File dialog box.
2. Select or type the file name.
3. Click Save.

To close the Selection List dialog box, click Close.

See Also

- [Object Selection](#)
- [Selecting or Highlighting Objects By Name](#)

Querying Objects

You can display object information interactively in the active schematic or abstract clock graph view by using the Query tool. When you click an object in the active view, information about the object, such as design and timing attributes, appears on the Query panel. You can copy the object information to the session transcript in the console log view.

To enable object queries in the active view,

- Click the  button on the Mouse Tools toolbar, or choose View > Mouse Tools > Query Tool.

The pointer becomes the Query Tool pointer (), and by default the Query panel appears.

When you move the Query tool over an object in a schematic or abstract clock graph view, the tool shows the object in the preview color, which is white by default but with a thinner line width than selection coloring. If InfoTips are enabled, information about the object appears in an InfoTip.

To display information an object on the Query panel,

- Click the object.

Each time you click an object with the Query tool, information about the new object replaces the information about the previous object on the Query panel.

You can select and copy text on the Query panel. You can also send the query text to the session transcript in the console log view. For details, see [Copying Text on the Query Panel](#).

You can set options on the Options menu to enable the automatic logging mechanism, wrap long lines of text on the Query panel, and control whether the Query panel opens automatically when you activate the Query tool. In addition you can open the Customize Query Toolbar dialog box to customize the information content of the queries for particular types of objects. For details, see [Configuring the Query Panel](#).

The information that the Query panel displays varies depending on the type of object. By default, you can configure the query information for a design, cell, port, pin, net, or netlist by adding or removing attributes in the QueryText attribute group. For more details, see [Modifying Attribute Groups](#).

See also:

- [Examining Object Attributes in a Data Table](#)

Copying Text on the Query Panel

You can select and copy text on the Query panel. You can also send the query text to the session transcript in the console log view.

To copy text on the Query panel,

- Select the text that you need to copy.

If you want to copy all the text on the panel, right-click and choose Select All.

2. Right-click and choose Copy.

To copy the current object information into the session transcript in the console log view,

- Click the  button at the top of the Query panel.

Alternatively, you can enable the automatic logging mechanism to automatically copy the information to the session transcript.

See Also

- [Querying Objects](#)

Configuring the Query Panel

You can configure the Query panel to control whether it opens automatically when you activate the Query tool, wraps long lines of text, or automatically copies the query text to the session transcript in the console log view. You can also open the Customize Query Toolbar dialog box to customize the information content of the queries for particular types of objects.

To control whether the Query panel appears automatically when you activate the Query tool,

- Click the  button and select or deselect the Show When Query Mode Enabled option.
This mechanism is enabled by default. Note that if you click an object with the Query tool when the Query panel is hidden, it appears automatically regardless of whether this option is selected

To enable text wrapping on the Query panel,

- Click the  button and select the Wrap Text option.

To enable the automatic logging mechanism,

- Click the  button and select the Auto Print Text to Log option.

The default attribute groups for a design, cell, net, port, pin, or netlist are `QueryText` for single-object queries and `Basic` for multiple-object queries. For object types that do not have `QueryText` or `Basic` attribute groups, the default attribute group is `All`.

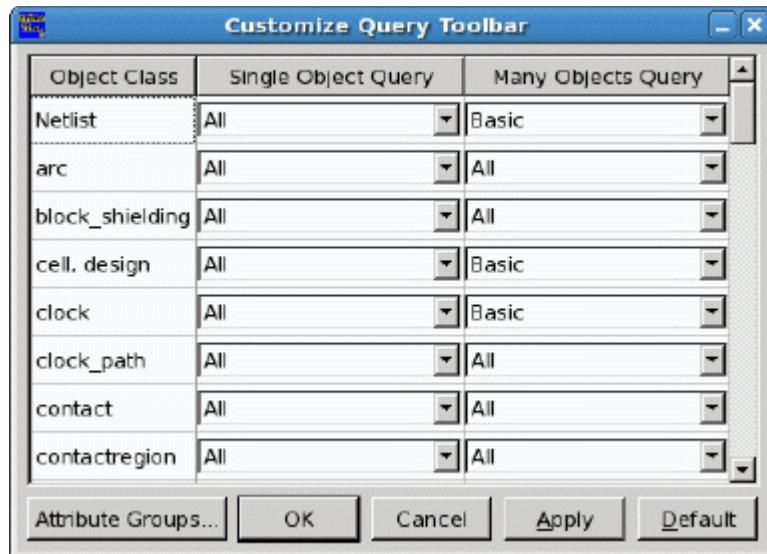
The `All` attribute group is available for every object and query type. The availability of other attribute groups varies depending on the object and query type. You can select any attribute group that appears in the list for a particular object and query type.

To configure the query content for one or more object queries,

- Click the  button and choose Customize Text.

The Customize Query Toolbar dialog box appears as shown in [Figure 18-55](#).

Figure 18-55 Customize Query Toolbar Dialog Box



2. Select attribute groups as needed in the lists for one or more object and query types. If you need to restore the dialog box options to their default values, click Default.
3. (Optional) Click the Attribute Groups button to open the Attribute Group Manager dialog box if you need to modify the `QueryText` attribute group. You can add or remove attributes in the attribute groups for one or more object types. For details, see [Managing Attribute Groups](#).
4. Click OK or Apply in the Customize Query Toolbar dialog box.

See Also

- [Querying Objects](#)

Viewing Object Properties

You can view attributes and other object properties for selected objects by using the Properties dialog box. In this context, an *object* can be a design instance, a leaf-level design object (cell, pin, port, net, or clock), or a timing path. You can also set, change, or remove values for certain types of properties.

Note:

The Properties dialog box displays properties for the objects that are currently selected. If you change the current selection when the Properties dialog box is open, the dialog box changes to display the properties for the newly selected objects.

To open the Properties dialog box,

1. Select one or more objects or timing paths.

You can select a single object, multiple objects of the same type, or multiple objects of different types.

2. Choose Edit > Properties.

If you select multiple objects, the number of selected objects displayed above the property list table. The property values for the first selected object appear in the table.

The Properties dialog box lists the properties in a table with two columns: property names and property values. The properties you can view include object names, attribute values, and certain timing and placement values. The list of properties differs depending on the types of objects that you select in the design and the attribute group that you select in the Properties dialog box.

Note:

Timing attribute values do not appear until you perform an operation that updates timing information, such as generating a timing report or opening a histogram.

The attributes are organized into attribute groups. For most object types, you can display all attributes, which includes user-defined attributes, or application (predefined) attributes. For nets, pins, or ports, you can also display basic attributes, which are the most frequently used attributes for the object type, or timing attributes. For cells or designs, you can display basic attributes, timing attributes, or placement attributes.

The default attribute group depends on which type of object you select. If you select multiple objects of different types, only the attributes in the basic attribute group appear. You can modify the contents of some attribute groups and create custom, user-defined attribute groups by using the Attribute Group Manager dialog box.

- To control which attributes appear in the Properties dialog box, select an option in the “Attribute group” list.
- To open Attribute Group Manager dialog box, click the  button in the Properties dialog box.

If you select multiple objects, the Properties dialog box can display a separate property list for each object. You can move back and forth sequentially between these property lists.

- To display the property values for the next object, click the Right Arrow button.
- To display the property values for the previous object, click the Left Arrow button.

Alternatively, you can combine the properties for all the selected objects together in a single list.

To display the properties for all selected objects,

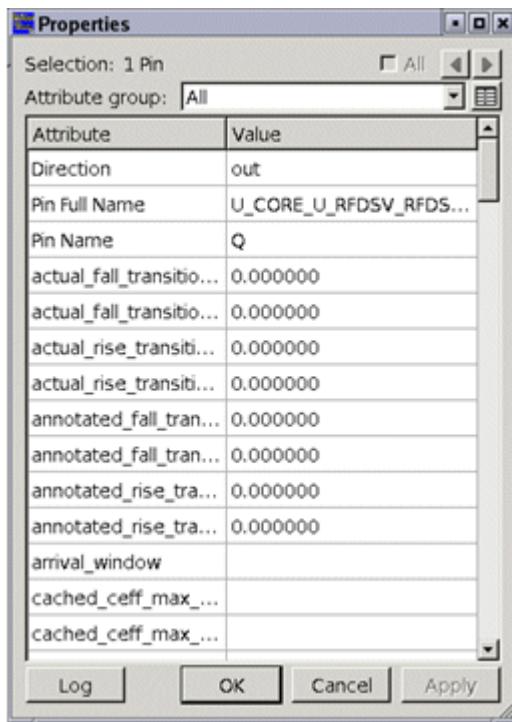
- Select the All option at the top of the Properties dialog box.

If you try to display properties for all objects of the same type, the dialog box shows the values that are identical for all the objects and displays “<Multiple values>” for other values. If you try to display properties for all objects of more than one type, the dialog box does not display any properties.

You can copy the property information to the console log view and the shell by clicking the Log button.

[Figure 18-56](#) shows an example of the Properties dialog box for a selected pin.

Figure 18-56 Properties Dialog Box



See Also

- [Managing Attribute Groups](#)

Managing Attribute Groups

Attribute groups are collections of attributes that the GUI uses to determine which attribute values to display in the Properties dialog box, in the Selection List dialog box, and on the Query panel. The content of these groups varies depending on the object type. When you

view object properties or the select list, you can control which types of attributes are visible in the properties list by selecting an attribute group.

You can view both application attribute groups that are predefined in the GUI and user-defined attribute groups that you create. The GUI provides the following predefined attribute groups:

- `All` contains all the predefined and user-defined attributes for an object
- `Application` contains all the predefined attributes for an object
- `Basic` contains the most frequently used attributes for an object
- `Placement` contains the placement attributes for cells and designs
- `SchemAnnotAttr` contains schematic annotation attributes
- `Timing` contains the timing attributes for cells, designs, nets, pins, ports, and timing paths
- `User` contains all the user-defined attributes that you define by using the `define_user_attribute` command

The tool updates the `All` and `User` groups automatically when you create or remove a user-defined attribute.

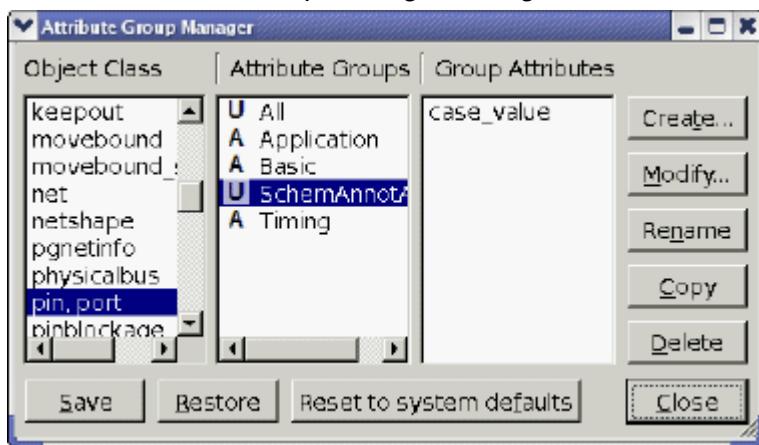
You can use the Attribute Group Manager dialog box to view and edit the attribute groups defined in the tool. Attribute groups allow you to specify a subset of the available attributes for a group of objects.

To open the Attribute Group Manager dialog box,

- Click the  button in the Properties dialog box or the Selection List dialog box, or click the Attributes Group button in the Customize Query Toolbar dialog box.

The Attribute Group Manager dialog box appears as shown in [Figure 18-57](#).

Figure 18-57 Attribute Group Manager Dialog Box



When you select an object type in the Object Class list, the names of the attribute groups for that object type appear in the Attribute Groups list. For port or pin attribute groups, select Pin in the Object Class list.

When you select a group name in the Attribute Groups list, the names of the attributes in the group appear in the Group Attributes list.

You can save attribute groups in your preferences file, load them from the preferences file, or reset the predefined attribute groups to their system defaults.

The Attribute Groups list displays a symbol beside each attribute group name indicating whether the attribute group is predefined in the GUI (A) or a user-defined attribute (U).

You can create, modify, and remove user-defined attribute groups. You cannot modify or remove predefined attribute groups. You can also copy an existing attribute group to create a new user-defined attribute group with the same content, and then rename and modify the group to meet your needs. You can also rename an attribute group that you create.

For more information about saving, creating, modifying, copying, and renaming attribute groups, see

- [Saving and Restoring Attribute Groups](#)
- [Creating Custom Attribute Groups](#)
- [Modifying Attribute Groups](#)
- [Copying and Renaming Attribute Groups](#)

See Also

- [Viewing Object Properties](#)
- [Viewing the Current Selection](#)
- [Querying Objects](#)

Saving and Restoring Attribute Groups

You can save attribute groups in your preferences file, load them from the preferences file, or reset the application attribute groups to their system defaults.

To save attribute groups in your preference file,

- Click the Save button.

To restore attribute groups from your preferences file,

- Click the Restore button.

To reset attribute groups to their system default configurations,

- Click the “Reset to system defaults” button.

See Also

- [Managing Attribute Groups](#)

Creating Custom Attribute Groups

You can create a new attribute group by specifying the group name, selecting one or more attributes, and ordering the attributes as needed.

To create a new attribute group,

1. Select an object type in the Object Class list.
2. Click the Create button.

The Create Attributes Group dialog box appears.

3. Type a unique name for the new attribute group.
4. Click OK.

The Attribute Group dialog box appears. The names of the available attributes appear in the attributes list at the left side of the dialog box.

5. Select one or more attribute names in the “All attributes” list, and then click the Right Arrow button.

The selected names move to the Group list. You can change the position of a name in the list by selecting the name and clicking the Up Arrow button or the Down Arrow button. To remove names from the list, select the names and click the Left Arrow button.

6. Click OK.

See Also

- [Managing Attribute Groups](#)
- [Modifying Attribute Groups](#)

Modifying Attribute Groups

You can modify an attribute group by specifying the group name and adding, removing, or reordering attributes as needed. You can modify the Basic, Placement, SchemAnnotAttr, and Timing attribute groups but not the All, Application, or User attribute groups.

To modify an attribute group,

1. Select the object type in the Object Class list.
2. Select the group name in the Attribute Groups list.
3. Click the Modify button to open the Attributes Group dialog box.
4. Modify the group as needed by adding, reordering, or removing attributes.
 - To add attributes to the group, select the attribute names in the “All attributes” list, and click the Right Arrow button.
 - To reorder attributes in the group, select an attribute name in the Group list, and click the Up Arrow button or the Down Arrow button.
 - To remove attributes from the group, select the attribute names in the Group list, and click the Left Arrow button.

5. Click OK.

See Also

- [Managing Attribute Groups](#)
- [Creating Custom Attribute Groups](#)

Copying and Renaming Attribute Groups

You can copy an existing attribute group to create a new user-defined attribute group with the same content. You can rename any attribute group that you have created.

To copy an attribute group,

1. Select the object type in the Object Class list.
2. Select the group in the Attribute Groups list.
3. Click the Copy button.

The name of the copy appears in the Attribute Groups list. This name consists of the name of the original group with “_copy” appended at the end. You can rename and modify the group to meet your needs.

To rename an attribute group,

1. Select the object type in the Object Class list.
2. Select the group in the Attribute Groups list.
3. Click the Rename button.
4. Edit the name as needed in the Attribute Groups list.

See Also

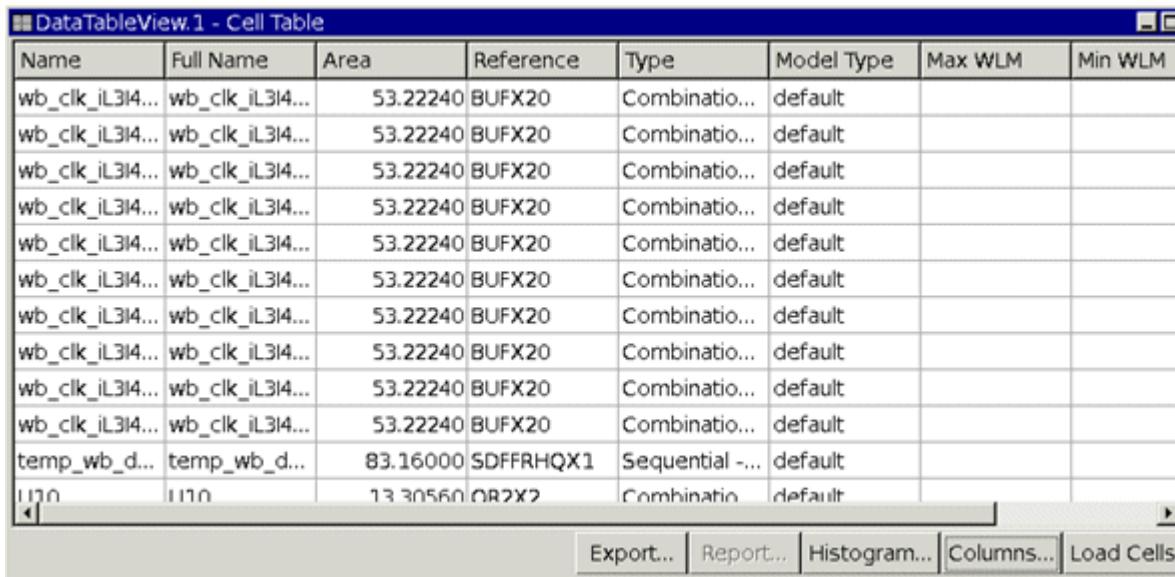
- [Managing Attribute Groups](#)

Examining Object Attributes in a Data Table

The GUI allows you to extract object attribute data from the design and to display the attributes in a data table. You can examine the attribute values, create histograms based on specific attributes, and generate reports for specific objects. You can open an attribute data table for cells, pins, ports, nets, or clocks.

You can select objects in another tool, such as the hierarchy browser or a schematic, and load them into an attribute data table. Alternatively, you can load objects in an attribute data table by specifying the object names or a name pattern, the object class, and an optional filter. For details, see [Loading Objects in an Object Attribute Table](#).

[Figure 18-58](#) shows an example of a cell attribute table.

Figure 18-58 Cell Attribute Data Table


The screenshot shows a Windows application window titled "DataTableView.1 - Cell Table". The window contains a table with the following columns: Name, Full Name, Area, Reference, Type, Model Type, Max WLM, and Min WLM. The table lists several objects, mostly named "wb_clk_iL3I4...", which have various attributes like Area (53.22240), Reference (BUFX20), Type (Combinatio...), and Model Type (default). One row is highlighted with a yellow background. At the bottom of the window is a button bar with the following buttons: Export..., Report..., Histogram..., Columns..., and Load Cells.

Name	Full Name	Area	Reference	Type	Model Type	Max WLM	Min WLM
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
wb_clk_iL3I4...	wb_clk_iL3I4...	53.22240	BUFX20	Combinatio...	default		
temp_wb_d...	temp_wb_d...	83.16000	SDFFRHQX1	Sequential ...	default		
1110	1110	13 30560	OR2X2	Combinatio...	default		

An attribute data table view consists of an object table and a button bar.

- The table displays the attribute values for each object.
- The button bar provides commands you can use to reload the objects, customize the table columns, save attribute values in a CSV file, create object data histograms, and generate object reports.

You can use the Up Arrow and Down Arrow keys to scroll up or down in the table. If some of the text in a column is missing because the column is too narrow, you can hold the pointer over the column to display the text in an InfoTip.

You can sort the table by the alphanumerical order of the contents in any column and adjust the widths of individual columns.

- To sort a column, click the column heading.
Click again if you want to reverse the order.
- To resize a column, drag the right edge of the column to the left or right.

You can also filter the paths based on a character string or regular expression that you define. For details, see [Filtering Tables and Lists](#).

You can create attribute data histograms based on the values in a column of the table. You can also select a cell, net, or port in the table and generate a report for the object.

- To generate a histogram based on the values in a specific column, click the Histogram button on the button bar and set options in the Table Histogram dialog box.

- To view an object report for a cell, net, or port, select the object, click the Report button on the button bar and set report options in the Report Timing for Selected Path dialog box.

You can also click buttons on the button bar to

- Configure the table by hiding, displaying, or reordering columns
- Save the attribute data in a comma separated value (CSV) format file
- Reload the objects

Note:

If the window is too narrow to display all the buttons, you can click the  button and choose a command on the menu that appears.

For more information about viewing object attributes in a data table, see

- [Loading Objects in an Object Attribute Table](#)
- [Creating Object Attribute Histograms](#)
- [Viewing Object Reports](#)
- [Configuring Data Table Columns](#)
- [Saving the Object Attribute Data](#)

See Also

- [Viewing Object Properties](#)

Loading Objects in an Object Attribute Table

You can load selected objects or search for and load objects based on name, class, and an optional filter.

To load selected objects in an attribute data table,

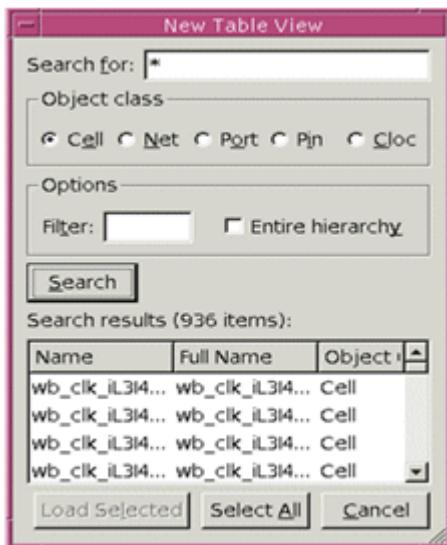
- Choose Design > New Table View for Selection.

The GUI opens a new attribute data table view and displays the object names and attribute values in the table.

To search for objects and load them in an attribute data table,

1. Choose Design > New Table View.

The New Table View dialog box appears as shown in [Figure 18-59](#).

Figure 18-59 New Table View Dialog Box

2. Enter a name or name pattern in the “Search for” box.
3. Select the option for the attribute class of the objects you want to find.
The choices are Cell, Net, Port, Pin, or Clock. The default is Cell.
4. Set options as needed.
You can use the Filter box to restrict the scope of the search.
5. Click Search.
The objects that match the search criteria appear in the “Search results” list.
6. Select the objects that you want to load into the attribute data table.
To select all the objects in the list, click Select All.
7. Click Load Selected.
The GUI opens a new attribute data table view and displays the object names and attribute values in the table.

See Also

- [Examining Object Attributes in a Data Table](#)

Creating Object Attribute Histograms

When you examine object attributes in a data table, you can create table data histograms to analyze the distribution of certain attribute values. Each histogram you create shows the distribution of values for the objects in the data table. You can create a histogram for values from any visible column that contains numeric data.

To display an object attribute histogram,

1. Click the Histogram button in an attribute data table.

The Table Histogram dialog box appears

2. Select the histogram data type in the Column list.

For object attribute data, the default column depends on the object type.

Note:

Some of the labels and option settings in the dialog box might change depending on which option you select in the Column list.

3. Set the histogram configuration and appearance options as needed.

4. Click OK.

For details about setting histogram configuration and appearance options, see [Viewing and Customizing Histograms](#).

See Also

- [Examining Object Attributes in a Data Table](#)

Viewing Object Reports

You can select an object in an attribute data table and generate a report for the object. You can generate reports for cells, nets, and ports.

To view a report for an object,

1. Select the object in the data table.
2. Click the Report button.
3. Click the Selection button in the report dialog box.
4. Set report and output options as needed.
5. Click OK.

The GUI displays the report in a new report view.

Alternatively, you can generate object reports by choosing commands on the Design menu.

See Also

- [Examining Object Attributes in a Data Table](#)
- [Viewing Reports](#)
- [Design Menu Commands](#)

Saving the Object Attribute Data

You can save the object attribute data by exporting it from the attribute data table to a comma separated value (CSV) format file.

To save the object attribute data,

1. Click the Export button below the attribute data table at the bottom of the window.
The Export to File dialog box appears.
2. Specify the name and location of the CSV file.
Specify a file name with the .csv extension if you want to open the file in Microsoft Excel.
3. Click Save.

See Also

- [Examining Object Attributes in a Data Table](#)

Recalculating Timing Paths

You can access and compare a set of normal paths to their recalculated path counterparts by using the recalculated path table. You can also select a specific path and open a path pin comparison table that shows the timing path objects side by side, with the regular and recalculated values as well as the differences between them.

For more information, see

- [Comparing Normal and Recalculated Paths](#)
- [Comparing Normal and Recalculated Path Pins](#)

In addition, you can select recalculated paths and load them into the path inspector.

See Also

- [Inspecting Timing Path Elements](#)
- [Examining Timing Path Details in a Data Table](#)

Comparing Normal and Recalculated Paths

To generate a recalculated path table from the PrimeTime GUI, first select one or more of the paths that you want to recalculate from the Path Analyzer window. You can also use the `change_selection` command to select a set of paths from the command line. Next, choose

Timing > New Recalculate Path Comparison Table. This creates a recalculated path table showing a summary of the normal and recalculated paths.

The information in the table includes the endpoint pin name, path group name, normal (path) slack, and recalculated (path) slack. If the slack is negative (violated), the cell appears red; otherwise, it is green. [Figure 18-60](#) shows an example of a recalculated path table.

Figure 18-60 Recalculated Path Table

Endpoint Pin Name	Path Group	Orig. Slack	Recalc. Slack
ffa/CDN	**async_default**	0.600475	0.600475
ffa/D	CLK	-0.0374383	-0.0360219
ffb/CDN	**async_default**	0.0327012	0.0369549
ffb/D	CLK	-0.0172185	-0.0160167
ffc/CDN	**async_default**	0.534778	0.541809
ffc/D	CLK	-0.0368479	-0.0355718
ffd/CDN	**async_default**	0.600475	0.600475
ffd/D	CLK	-0.0582917	-0.0566103

You can view more information about the exact path differences between the normal and recalculated path. From the path comparison table, highlight the path, and right-click to obtain more options, such as the ability to launch the Path Inspector window to inspect the normal and recalculated path or to generate a path pin comparison table.

Note that the path pin comparison table is only enabled when the normal and recalculated path have the same number of through pins.

See Also

- [Comparing Normal and Recalculated Path Pins](#)

Comparing Normal and Recalculated Path Pins

You can generate a path pin comparison table to see detailed information for normal and recalculated paths side by side. To generate a path pin comparison table from the PrimeTime GUI, you first select a row from the recalculated path table and then choose Timing > New Recalculated Path Pin Comparison Table. Alternatively, after selecting the path, you can right-click and select New Path Pin Comparison Table to generate this table. [Figure 18-61](#) shows an example of a path pin comparison table.

Figure 18-61 Path Pin Comparison Table

#	Pin Name	Normal	Recalc	Edge	Normal	Recalc	Edge	Normal	Recalc	Edge	Normal	Recalc
1	t0:t											
2	t0:t1	0.387303	0.387303		0.23E-01	0.24E-01		0.0311	0.03E-01		0.4m2	4m2
3	t0:t2	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
4	t0:t3	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
5	t0:t4	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
6	t0:t5	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
7	t0:t6	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
8	t0:t7	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
9	t0:t8	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
10	t0:t9	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
11	t0:t10	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
12	t0:t11	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
13	t0:t12	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
14	t0:t13	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
15	t0:t14	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
16	t0:t15	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
17	t0:t16	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
18	t0:t17	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
19	t0:t18	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
20	t0:t19	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
21	t0:t20	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
22	t0:t21	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
23	t0:t22	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
24	t0:t23	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
25	t0:t24	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
26	t0:t25	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
27	t0:t26	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
28	t0:t27	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
29	t0:t28	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
30	t0:t29	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
31	t0:t30	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
32	t0:t31	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
33	t0:t32	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
34	t0:t33	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
35	t0:t34	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
36	t0:t35	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
37	t0:t36	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
38	t0:t37	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
39	t0:t38	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
40	t0:t39	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
41	t0:t40	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
42	t0:t41	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
43	t0:t42	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
44	t0:t43	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
45	t0:t44	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
46	t0:t45	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
47	t0:t46	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
48	t0:t47	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
49	t0:t48	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
50	t0:t49	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
51	t0:t50	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
52	t0:t51	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
53	t0:t52	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
54	t0:t53	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
55	t0:t54	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
56	t0:t55	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
57	t0:t56	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
58	t0:t57	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
59	t0:t58	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
60	t0:t59	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
61	t0:t60	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
62	t0:t61	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
63	t0:t62	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
64	t0:t63	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
65	t0:t64	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
66	t0:t65	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
67	t0:t66	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
68	t0:t67	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
69	t0:t68	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
70	t0:t69	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
71	t0:t70	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
72	t0:t71	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
73	t0:t72	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
74	t0:t73	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
75	t0:t74	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
76	t0:t75	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
77	t0:t76	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
78	t0:t77	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
79	t0:t78	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
80	t0:t79	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
81	t0:t80	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
82	t0:t81	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
83	t0:t82	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
84	t0:t83	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
85	t0:t84	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
86	t0:t85	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
87	t0:t86	-0.387303	-0.387303		-0.23E-01	-0.24E-01		0	0		0	0
88	t0:t87	-0.387303	-0.387303		-0.23E-01	-0.2						

The path pin comparison table shows the pin names, pin transitions, path delays, incremental delays, edge, and delta delays along the pins in the normal and recalculated path, side by side. A positive change appears in green, and a negative change appears in red.

See Also

- [Comparing Normal and Recalculated Paths](#)

Analyzing Signal Integrity

If you are a licensed user of the PrimeTime SI tool, and if crosstalk analysis is currently enabled, you can display histograms of delta delay, bump voltages, and noise analysis results. For example, a typical analysis session might include the following steps:

1. Read in the design, enable crosstalk analysis, read in the parasitic data, and set the crosstalk analysis parameters.
2. Perform a timing analysis by running the `report_timing` or `report_noise` command.
3. Generate histograms for endpoint slack, delta delay, bump voltage, noise slack, and noise bump.
4. In the timing path table, select the worst-case path and click the Inspector button to open the path inspector window.
5. In the path inspector window, view the path element table.

The table has columns showing delta transition times and delta delays along the path.

6. Select the victim net and perform a coupling analysis by choosing Crosstalk > Coupling Analysis for Selected Nets.
7. Generate noise histograms using Noise > Noise Slack Histogram.
8. Check noise bumps against noise immunity curves using Noise > Noise Immunity Curve.
9. Perform crosstalk analysis by choosing Crosstalk > Coupling Analysis for Selected Nets or Crosstalk > Coupling Analysis for SI Bottleneck Nets.

Note:

Some reports and histogram windows use the term “effective” to describe a victim net, aggressor net, or capacitor. This means an object that has not been removed by filtering and has been selected for crosstalk analysis.

For general information about the crosstalk analysis flow, see [Overview of Signal Integrity and Crosstalk](#).

You can display PrimeTime SI crosstalk analysis results in delta delay and bump voltage histograms.

- To display a histogram of delta delay values induced on victim nets in the design, choose Crosstalk > Delta Delay Histogram.
- To display a histogram of delta delay values induced on victim nets in one or more selected paths, choose Crosstalk > Path Delta Delay Histogram.
- To display a histogram of individual voltage bumps induced on one victim net by multiple aggressor nets, choose Crosstalk > Bump Voltage Histogram.
- To display a histogram of the accumulated voltage bumps induced on victim nets by multiple aggressor nets, choose Crosstalk > Accumulated Bump Voltage Histogram.

You can display static noise analysis results in noise slack and noise bump histograms. Upon completion of static noise analysis with the `update_noise` or `report_noise` command, you can display the results in histogram form.

- To display a histogram of noise slack values for nets in the design, considering the worst load pin in each net, choose Noise > Noise Slack Histogram.
- To display a histogram of noise slack values for nets in the design, considering the worst load pin in each net, choose Noise > Noise Bump Histogram.
- To display a histogram of total crosstalk noise bump heights (not including propagated noise) for nets in the design, considering the worst load pin per net, choose Noise > Accumulated Noise Bump Histogram.

See Also

- [Examining Delta Delay](#)
- [Examining Path Delta Delay](#)
- [Examining Bump Voltage](#)
- [Examining Accumulated Bump Voltage](#)
- [Examining Crosstalk Coupling](#)
- [Examining Noise Slack](#)
- [Examining Noise Bump](#)
- [Examining Accumulated Noise Bump](#)
- [Examining Noise Immunity Curves](#)
- [Viewing Propagated Waveforms for Recalculated Paths](#)

Examining Delta Delay

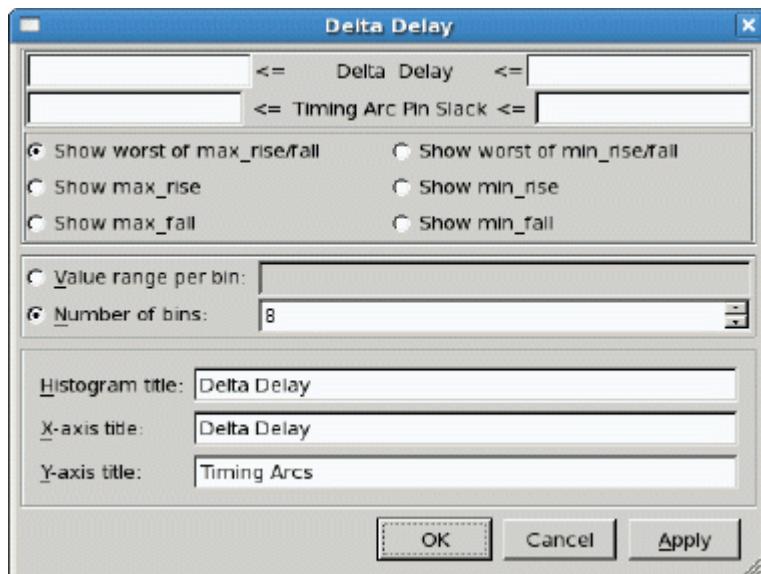
A delta delay histogram shows the distribution of delay changes induced on victim nets by aggressor nets.

To generate a delta delay histogram,

1. Choose Crosstalk > Delta Delay Histogram or click the  button on the Signal Integrity Histograms toolbar.

The Delta Delay dialog box appears as shown in [Figure 18-62](#).

Figure 18-62 Delta Delay Dialog Box

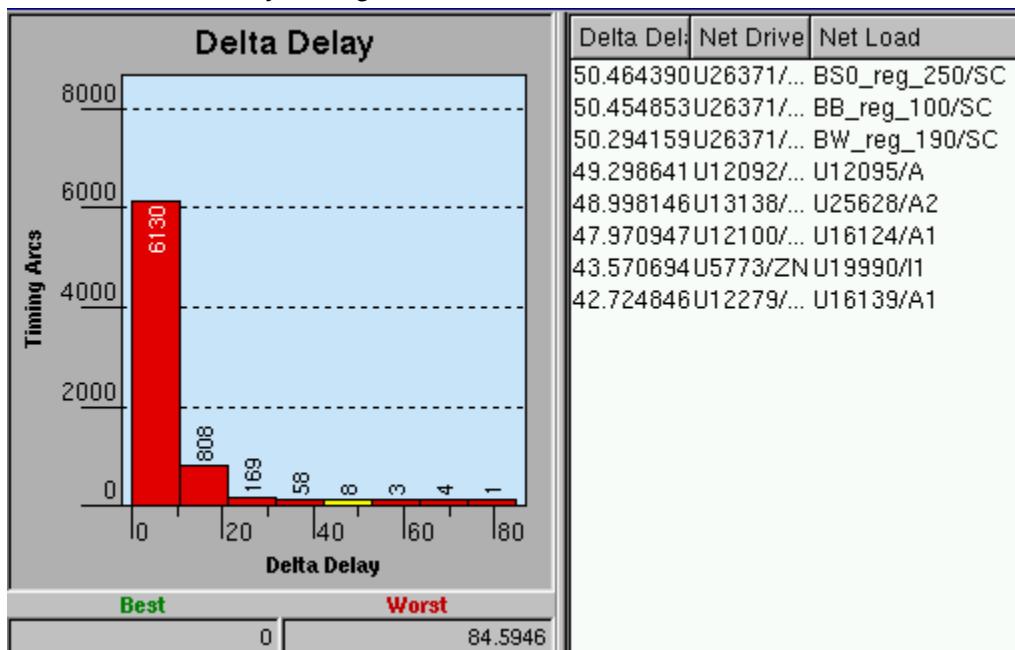


2. Specify the types of delta delays to be included in the histogram plot.

By default, all delta delays are included in the histogram plot. You can restrict the types of data included in the plot by specifying the desired delta delay range, slack range, delay type (minimum or maximum), and transition type (rising or falling).

3. (Optional) Set options to control the distribution of slack values or the number of bins and to configure the plot.
4. Click OK to generate the histogram.

[Figure 18-63](#) is a typical delta delay histogram. In this example, the highest histogram bar is selected and highlighted. The net arcs contained in that bin are listed on the right, along with the corresponding delta delay values.

Figure 18-63 Delta Delay Histogram**See Also**

- [Analyzing Signal Integrity](#)
- [Examining Path Delta Delay](#)

Examining Path Delta Delay

You can select one or more paths and view a histogram of the delta delays of nets along those paths.

Before generating a path delta delay histogram, you must select the paths. You can select paths in a path slack histogram or by choosing **Select > Paths From/Through/To** and specifying the path selection criteria in the **Select Paths** dialog box. You can also select an endpoint in an endpoint slack histogram and enter it in the **To** box in the **Select Paths** dialog box.

To generate a path delta delay histogram,

1. Select one or more paths.
2. Choose **Crosstalk > Path Delta Delay Histogram** or click the button on the **Signal Integrity Histograms** toolbar.

The Path Delta Delay dialog box appears.

3. Specify the types of delta delays to be included in the histogram plot.

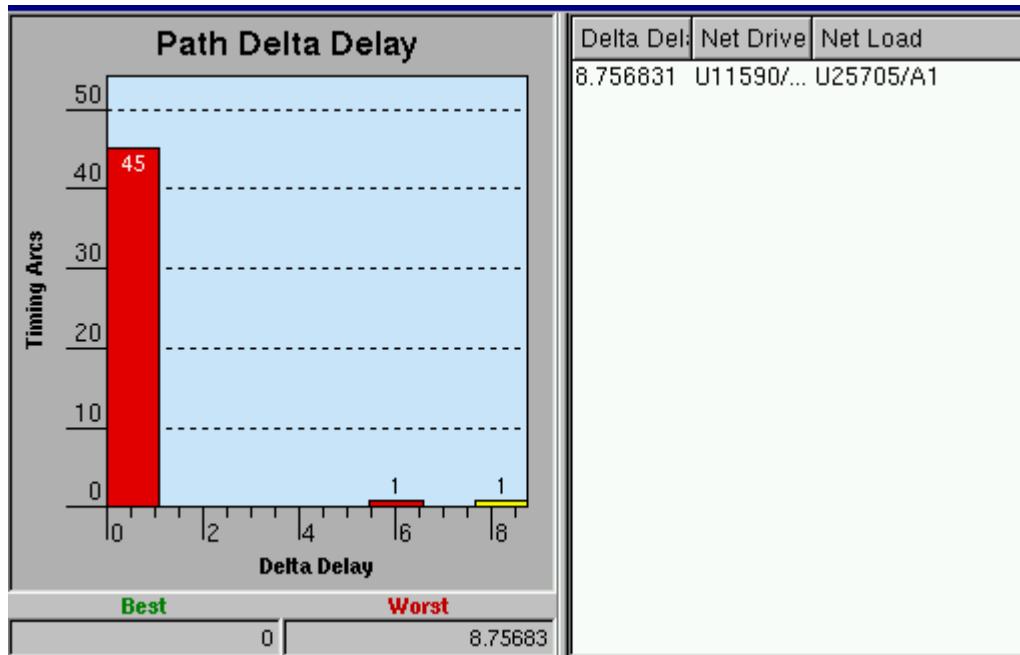
You can restrict the range of delta delays and timing arc pin slacks to be included in the plot.

4. (Optional) Set options to control the distribution of slack values or the number of bins and to configure the plot.

5. Click OK to generate the histogram.

See [Figure 18-64](#) for an example of a path delta delay histogram.

Figure 18-64 Path Delta Delay Histogram



To examine the cause of a delta delay on a particular net,

1. Select the bin in the histogram plot.
2. Select the net in the net list to the right of the plot.
3. Choose Crosstalk > Bump Voltage Histogram.

See Also

- [Analyzing Signal Integrity](#)

Examining Bump Voltage

A bump voltage histogram shows the distribution of individual voltage bumps induced on one victim net by multiple aggressor nets. These bumps are used to calculate delta delay effects, not static noise effects. To display histograms of static noise bump data, choose a command from the Noise menu.

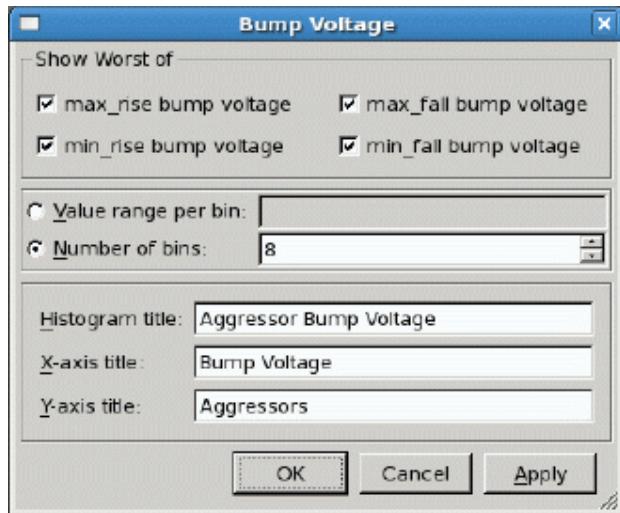
Before generating a bump voltage histogram, you must select a victim net. You can select the net in a schematic, path profile, or a histogram view or by choosing one of the Select > Nets commands.

To generate a bump voltage histogram for a net,

1. Select the victim net.
2. Choose Crosstalk > Bump Voltage Histogram or click the  button the Signal Integrity Histograms toolbar.

The Bump Voltage dialog box appears as shown in [Figure 18-65](#).

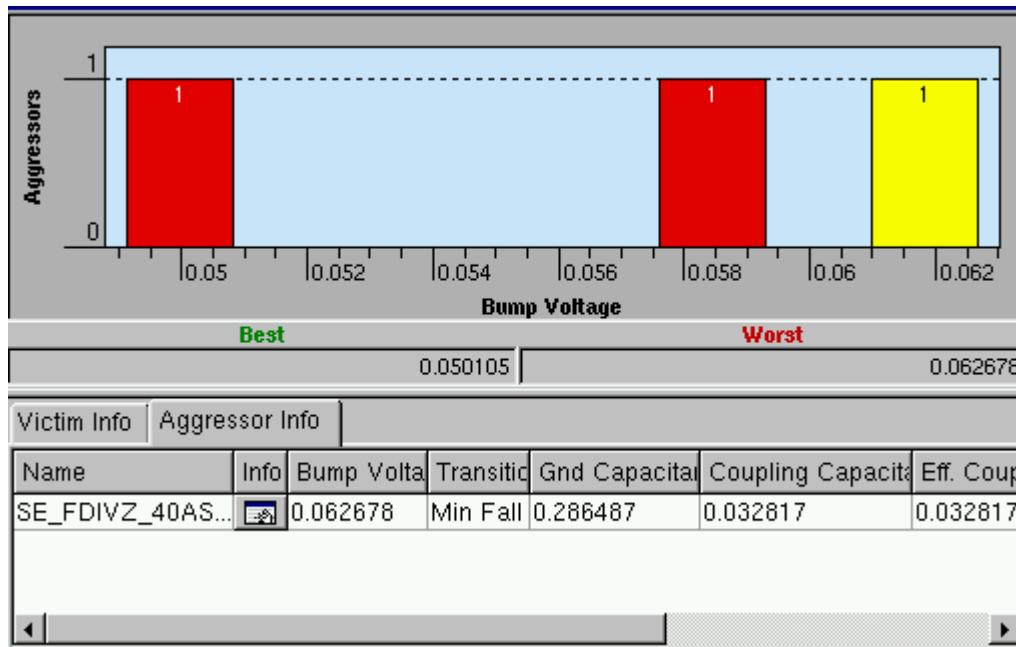
Figure 18-65 Bump Voltage Dialog Box



3. Specify the types of voltage bumps (max_rise, max_fall, min_rise, or min_fall) to include in the plot.
4. (Optional) Set options to control the distribution of bump voltage values or the number of bins and to configure the plot.
5. Click OK to generate the histogram.

[Figure 18-66](#) shows an example of a bump voltage histogram.

Figure 18-66 Bump Voltage Histogram With Victim Info Spreadsheet



The histogram window has two tabbed pages at the bottom, labeled Victim Info, and Aggressor Info.

- Clicking the Victim Info tab displays a spreadsheet containing detailed information about the victim net.
- Clicking the Aggressor Info tab displays a spreadsheet with information about all aggressor nets in the currently selected bin.

The Victim Info spreadsheet shows detailed information about the victim net, including the net name, ground capacitance, coupling capacitance, aggressor information, delta delay, and delta slew.

The Aggressor Info spreadsheet shows information about each aggressor net in the currently selected bin, including the net name, bump voltage contribution, ground capacitance, coupling capacitance, and filtering status. All bump voltages are shown as a fraction of VDD, not an absolute number of volts.

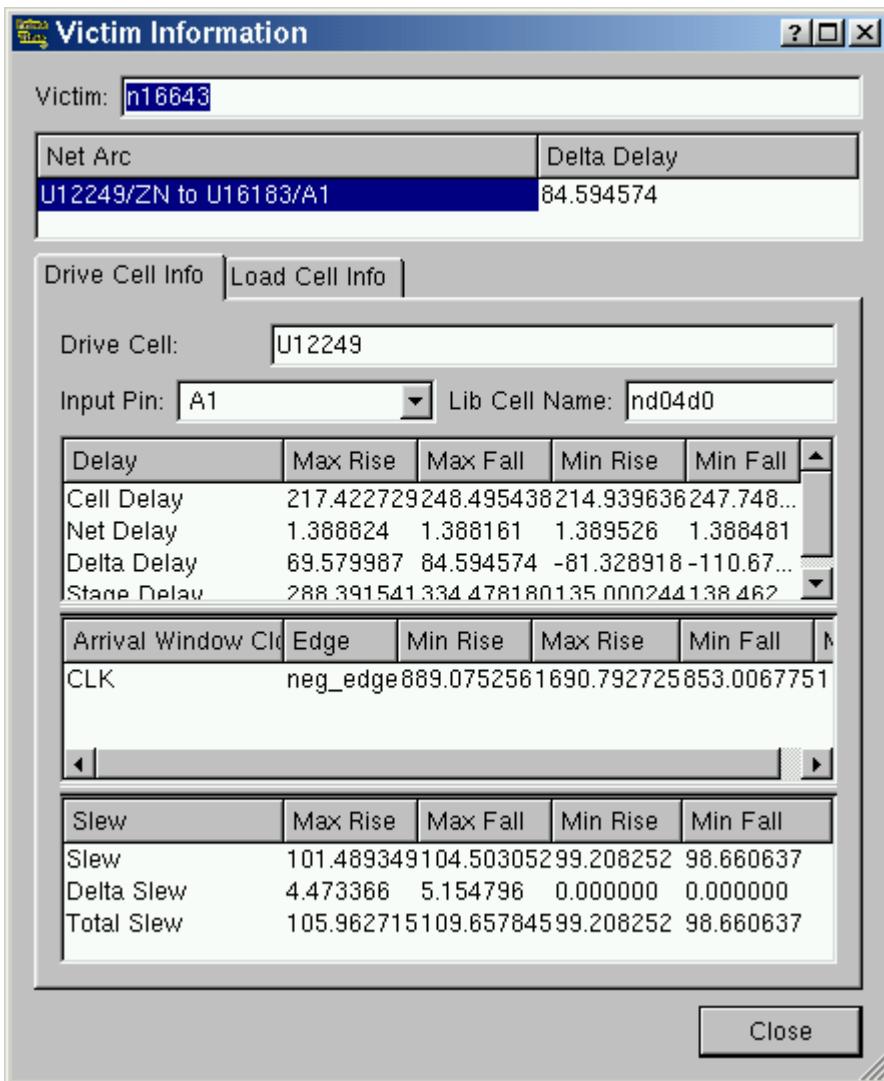
To view detailed information about the victim net,

- Click the  button in the Victim Info spreadsheet.

The Victim Information dialog box appears.

[Figure 18-67](#) shows an example of the information displayed by the Victim Information dialog box.

Figure 18-67 Victim Information Dialog Box



To view detailed information about the aggressor net,

- Click the button in the Aggressor Info spreadsheet. The Aggressor Information dialog box appears.

You must close the Victim Information dialog box or the Aggressor Information dialog box before you can go back to using the main GUI window.

See Also

- [Analyzing Signal Integrity](#)

Examining Accumulated Bump Voltage

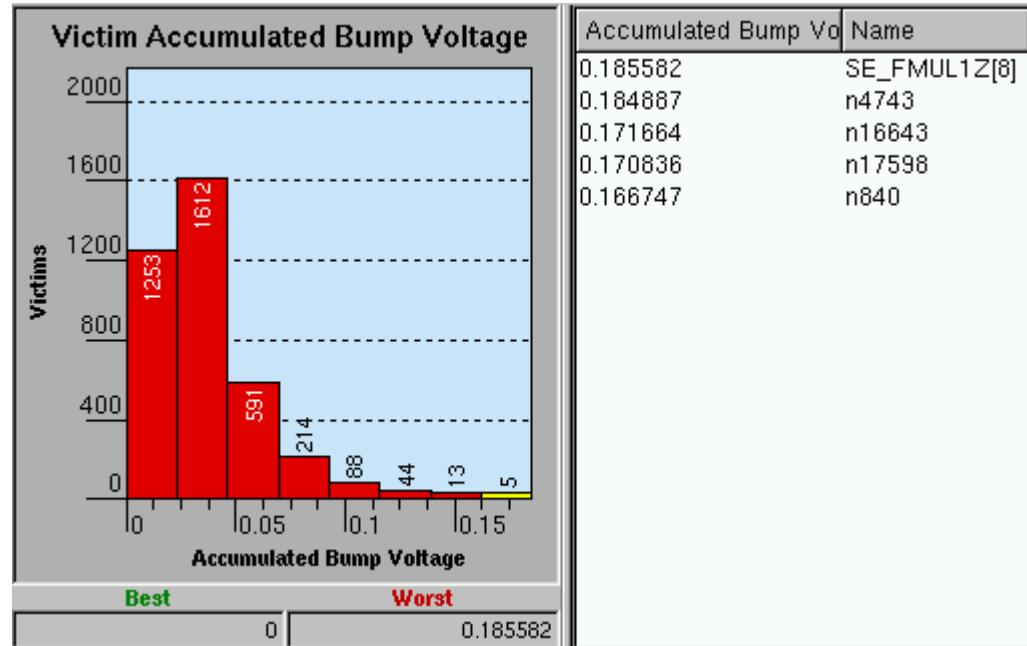
An accumulated bump voltage histogram shows the distribution of the total voltage bumps induced on victim net arcs. The word *accumulated* means the total of all voltage bumps induced on a victim net by multiple aggressor nets. These bumps are used to calculate delta delay effects, not static noise effects. To display histograms of static noise bump data, choose a histogram menu command from the Noise menu.

To generate an accumulated bump voltage histogram,

1. Choose Crosstalk > Accumulated Bump Voltage Histogram or click the  button on the Signal Integrity Histograms toolbar.
2. Specify the types of accumulated voltage bumps (max_rise, max_fall, min_rise, or min_fall) to include in the plot.
3. (Optional) Set options to control the distribution of bump voltage values or the number of bins and to configure the plot.
4. Click OK to generate the histogram.

[Figure 18-68](#) shows a typical accumulated voltage bump histogram. The nets contained in the selected bin are listed on the right, along with the corresponding accumulated bump voltage values.

Figure 18-68 Accumulated Bump Voltage Histogram



See Also

- [Analyzing Signal Integrity](#)

Examining Crosstalk Coupling

You can perform crosstalk coupling analysis on signal integrity bottlenecks, where given nets contribute to multiple crosstalk violations.

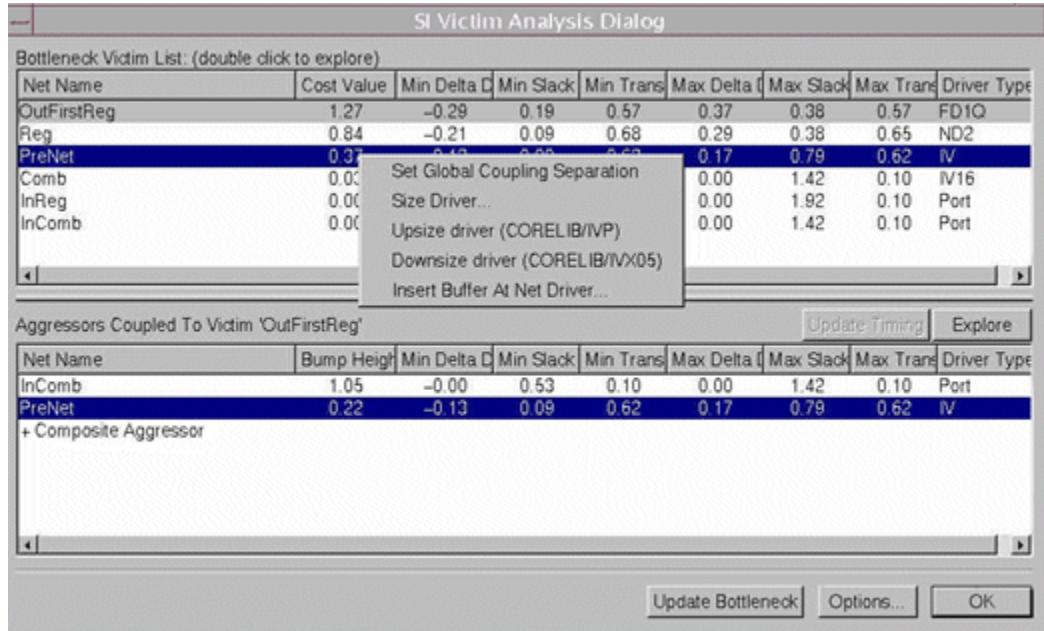
To find crosstalk bottlenecks,

1. Choose Crosstalk > Coupling Analysis for SI Bottleneck Nets.
2. Set analysis options in the SI Bottleneck Options dialog box.
3. Click OK.

This is the same as running the `report_si_bottleneck` command.

The results appear in a victim analysis table in the SI Victim Analysis Dialog dialog box shown in [Figure 18-69](#).

Figure 18-69 Victim Analysis Table

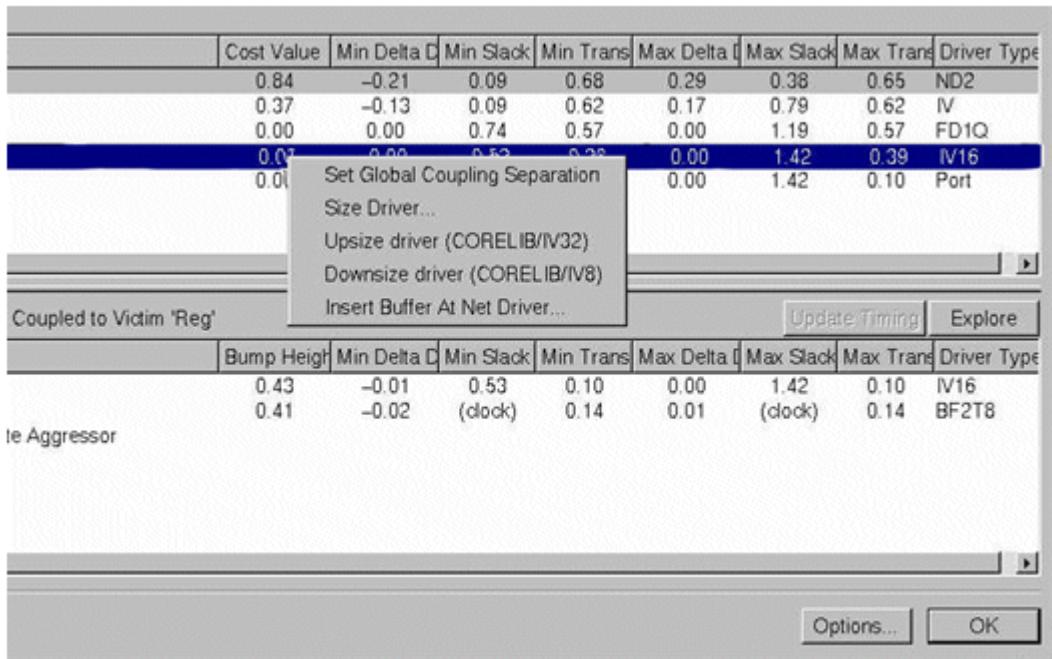


For an analysis on a specified collection of paths or nets, select the paths or nets, and then choose one of the following commands:

- To display a table of victim nets, cross-coupled aggressor nets, and crosstalk information for one or more selected paths, choose Crosstalk > Coupling Analysis For Selected Paths.
- To display a table of victim nets, cross-coupled aggressor nets, and crosstalk information for nets in one or more selected nets, choose Crosstalk > Coupling Analysis For Selected Nets.

[Figure 18-70](#) shows how the Coupling Analysis Dialog dialog box is populated with the information for the selected paths or nets.

Figure 18-70 Coupling Analysis Table



The SI Victim Analysis Dialog and Coupling Analysis Dialog dialog boxes each consist of two sections. The nets of interest are shown in the top section. When you double-click a net in the top section, the coupled nets are shown in the bottom section.

The minimum and maximum delta, slack, and transition information is shown for all nets. You can right-click any net in either section to open a pop-up menu with commands that you can choose to perform the following ECO operations:

- Set coupling separation (either global or pairwise)
- Net driver resizing (upsizing, downsizing, or specifying a replacement cell)
- Buffer insertion

By considering the net information, you can make informed decisions about potential fixes. For example, if an aggressor net has sufficient setup slack, you could downsize its driver. If a victim net has a slow transition time, you can enlarge its driver to make it more immune to crosstalk.

You can set coupling separation between nets by choosing commands from the ECO menu.

To set global coupling separation between nets,

1. Select the nets.
2. Choose ECO > Set Coupling Separation.

To set pairwise coupling separation between nets,

1. Select the nets.
2. Choose ECO > Set Pairwise Coupling Separation.

This opens the Set Pairwise Coupling Separation window, as shown [Figure 18-71](#).

Figure 18-71 Set Pairwise Coupling Separation Table

Set Pairwise Coupling Separation								
Selected Nets :								
Net Name	Cap. Value	Min Delta	Min Slack	Min Trans	Max Delta	Max Slack	Max Trans	Driver Type
InComb	0.04	-0.00	0.53	0.10	0.00	1.42	0.10	Port
OutComb	0.00	0.00	0.53	0.39	0.00	1.42	0.39	IV16
OutReg	0.00	0.00	0.74	0.57	0.00	1.19	0.57	FD1Q
Reg	0.09	-0.21	0.09	0.68	0.29	0.38	0.65	ND2

Nets coupled to InComb								
Net Name	Cap. Value	Min Delta	Min Trans	Min Slack	Max Delta	Max Slack	Max Trans	Driver Type
OutFirstReg	0.04	-0.29	0.19	0.57	0.37	0.38	0.57	FD1Q
net1	0.00	-0.13	----	0.00	0.17	----	0.00	BF2T16

OK Cancel Apply

For more information, see the man pages for the `set_si_delay_analysis` and `set_coupling_separation` commands.

See Also

- [Analyzing Signal Integrity](#)

Examining Noise Slack

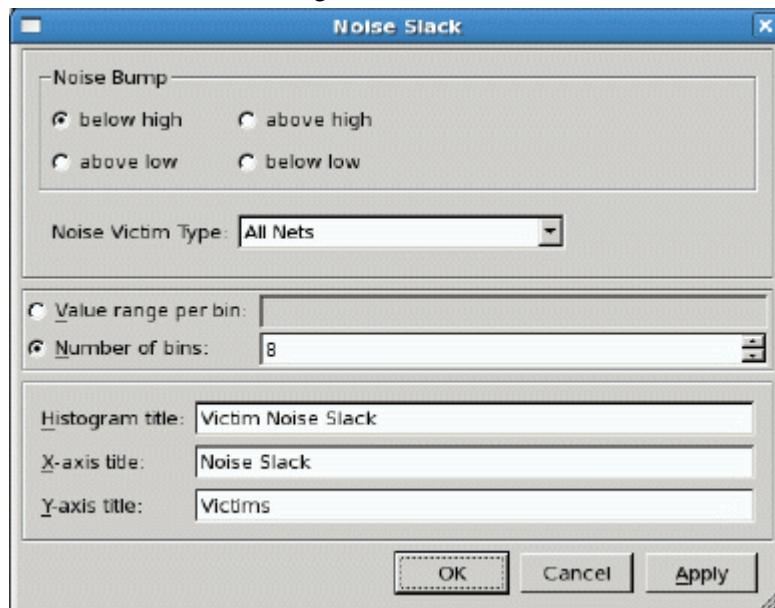
A noise slack histogram shows the distribution of noise slack values for nets in the design, considering the worst load pin in each net. Noise slack is the amount by which a logic failure is avoided at a cell input with the worst-case composite noise bump on that pin. The slack value is the failure threshold voltage minus the bump height. The units are in library voltage units.

To generate a noise slack histogram,

1. Choose Noise > Noise Slack Histogram or click the  button on the Signal Integrity Histograms toolbar.

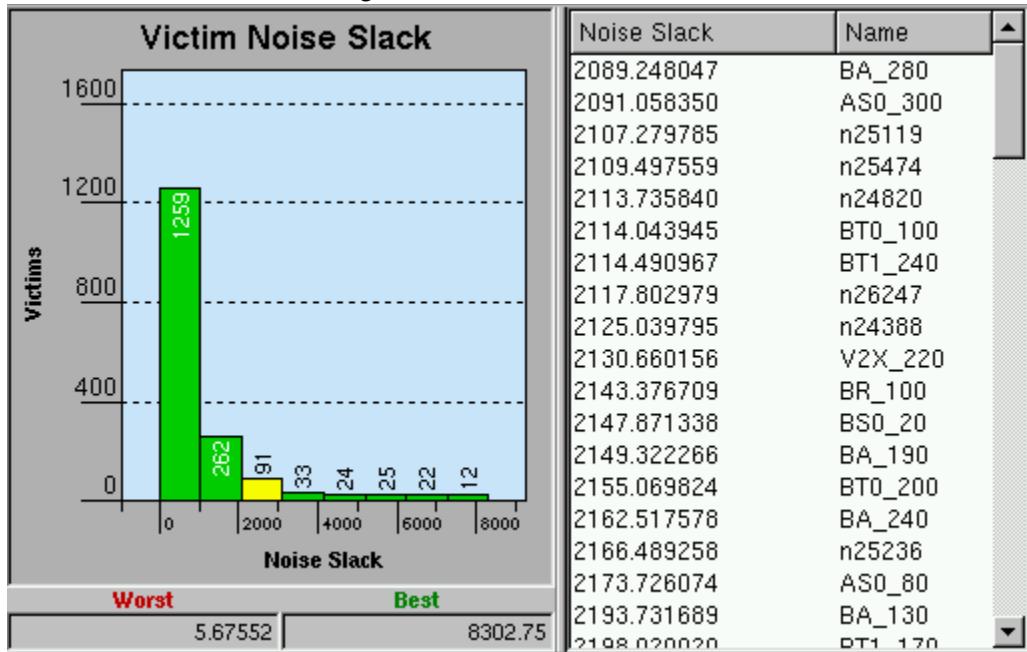
The Noise Slack dialog box appears as shown in [Figure 18-72](#).

Figure 18-72 Noise Slack Dialog Box



2. Specify the bump type and the types of load pins to be included in the report.
3. (Optional) Set options to control the distribution of bump voltage values or the number of bins and to configure the plot.
4. Click OK to generate the histogram.

[Figure 18-73](#) shows an example of a noise slack histogram.

Figure 18-73 Noise Slack Histogram

Click a bin to display a list of the nets in that bin. You can click a net in the list to select the net and display its victim noise bump histogram, which shows the load pins for the net.

See Also

- [Examining Noise Bump](#)
- [Analyzing Signal Integrity](#)

Examining Noise Bump

A victim noise bump histogram shows the distribution of noise bump heights on a load pin resulting from individual aggressor nets. The aggressor nets are distributed in the histogram according to their individual contribution to the total bump height at the load pin, expressed as a fraction of the total rail-to-rail voltage. Two spreadsheets show relevant information about the victim net and aggressor nets.

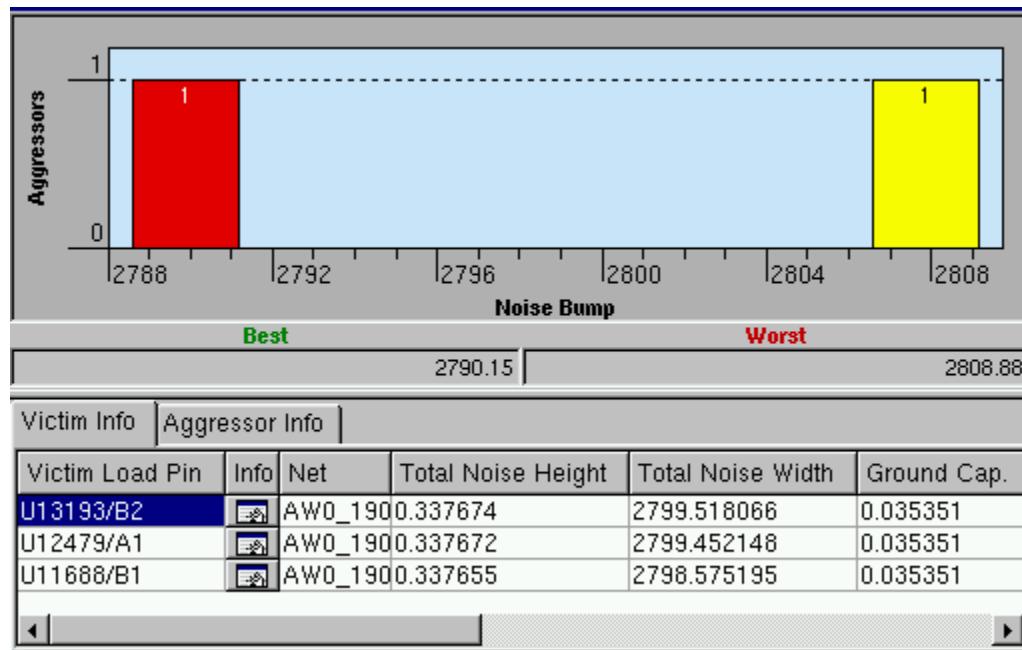
Before generating a noise bump histogram, you must select the net or load pin. You can select the net or pin in a schematic, path profile, or histogram view, in a path inspector window, or by choosing one of the Select > Nets or Select > Ports/Pins commands.

To generate a noise bump histogram,

1. Select the net or load pin.
 2. Choose Noise > Noise Bump Histogram or click the  button on the Signal Integrity Histograms toolbar.
- The Noise Bump dialog box appears.
3. Specify the noise bump type.
 4. (Optional) Set options to control the distribution of bump voltage values or the number of bins and to configure the plot.
 5. Click OK to generate the histogram.

[Figure 18-74](#) shows an example of a noise bump histogram.

Figure 18-74 Noise Bump Histogram With Victim Info Spreadsheet



If you select a net (not a specific load pin) before you generate the histogram, the histogram displays information about the load pin on the net that has the least noise slack, and the Victim Info spreadsheet lists all the load pins on the net. You can select a particular load pin from the list to generate a histogram for that load pin.

The histogram window has two tabbed pages at the bottom, labeled Victim Info and Aggressor Info.

- Clicking the Victim Info tab displays detailed information about the victim net.
- Clicking the Aggressor Info tab displays information about all aggressor nets in the currently selected bin.

The Victim Info spreadsheet shows detailed information about the victim net, including the load pin, noise height and width, ground capacitance, coupling capacitance, noise bump information, and noise slack information.

To open a dialog box showing detailed information about the victim net,

- Click the Info button next to the victim name in the spreadsheet.

The Aggressor Info spreadsheet shows information about each aggressor net in the currently selected bin, including the net name, bump width and height, ground capacitance, coupling capacitance, and filtering status.

To open a dialog box showing more information about an aggressor net,

- Click the Info button next to the aggressor net name in the spreadsheet.

See Also

- [Analyzing Signal Integrity](#)

Examining Accumulated Noise Bump

An accumulated noise bump histogram shows the distribution of total crosstalk noise bump heights (not including propagated noise) for nets in the design, considering the worst load pin per net.

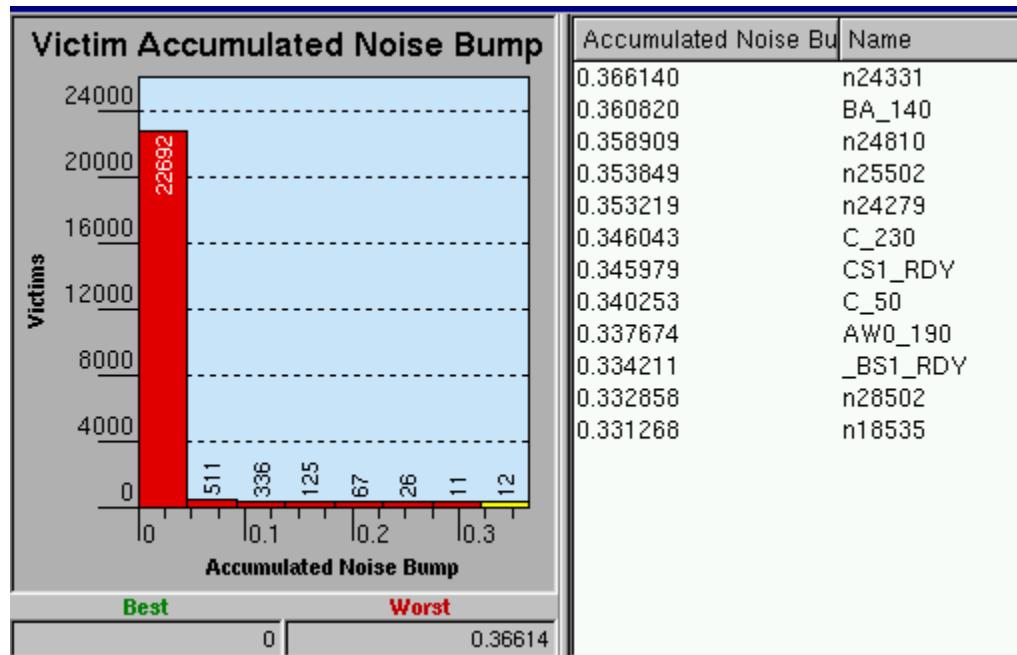
To generate an accumulated noise bump histogram,

1. Choose Noise > Accumulated Noise Bump Histogram or click the  button on the Signal Integrity Histograms toolbar.
The Accumulated Noise Bump dialog box appears.
2. Specify the bump type and the types of load pins to be included in the report.
3. (Optional) Set options to control the distribution of bump voltage values or the number of bins and to configure the plot.
4. Click OK to generate the histogram.

If necessary, the tool runs the `update_noise` command in the background to generate the noise data for the histogram.

[Figure 18-75](#) shows an example of an accumulated noise bump histogram. The load pins are distributed in the histogram according to their accumulated bump voltage values, expressed as a fraction of the total rail-to-rail voltage. The accumulated bump voltage is the height of the highest noise bump at the load pin resulting from all aggressor nets switching at the same time within their respective timing windows.

Figure 18-75 Accumulated Noise Bump Histogram



Click any histogram bin to display a list of the nets in that bin. You can click a net in the list to select that net and display its victim noise bump histogram, which then shows the load pins for that net.

See Also

- [Analyzing Signal Integrity](#)

Examining Noise Immunity Curves

The GUI can display a plot of the noise immunity curve at a cell input pin, showing a plot of the data point corresponding to the worst-case noise bump calculated at that pin.

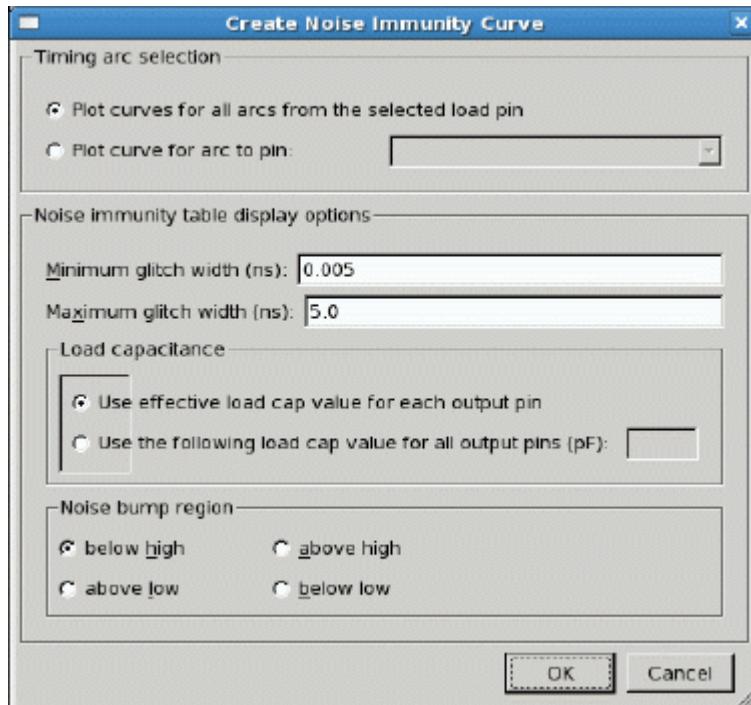
Before generating a plot of a noise immunity curve, you must select the cell input pin. For example, if you know the pin name, you can choose Select > By Name or select the input pin showing the largest delta delay from the path element table in the path inspector window.

To generate a plot of a noise immunity curve,

1. Select the cell input pin.
2. Choose Noise > Noise Immunity Curve.

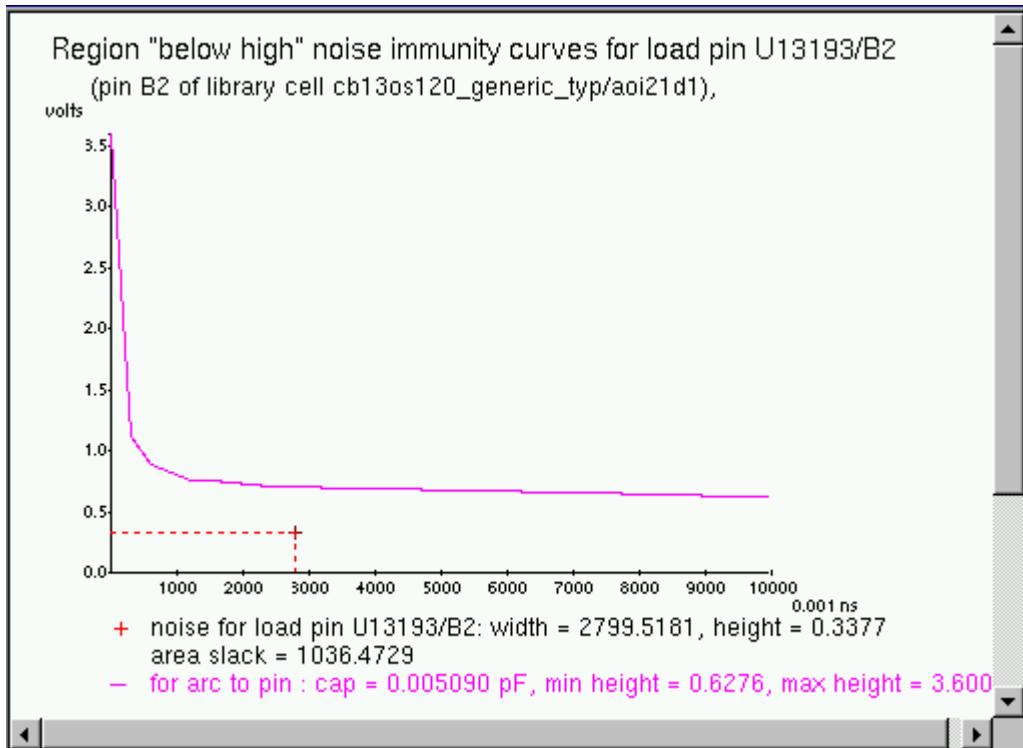
The Create Noise Immunity Curve dialog box appears as shown in [Figure 18-76](#).

Figure 18-76 Create Noise Immunity Curve Dialog Box



3. Set options in the dialog box as desired, including the noise bump type (below high, above low, above high, or below low).
4. Click OK.

The GUI displays the noise immunity curve and the worst-case noise bump data, as shown in [Figure 18-77](#).

Figure 18-77 Noise Detection Display

The curve shows the pass or fail input threshold voltage as a function of noise bump width. A small cross indicates the worst-case noise bump width and height at the pin.

To modify the width range and bump type of an existing plot,

1. Right-click and choose Modify Graph.

The Modify Noise Immunity Curve dialog box appears.

2. Set options in the dialog box as desired.

3. Click OK.

See Also

- [Analyzing Signal Integrity](#)

Viewing Propagated Waveforms for Recalculated Paths

For libraries that have both CCS timing and CCS noise models, the PrimeTime SI tool can apply an advanced, gate-level delay calculation mode for path-based analysis. This mode uses the CCS timing and noise models together with propagated piecewise linear waveforms in place of simplified equivalent waveforms. To use this feature, you must set the

`delay_calc_waveform_analysis_mode` variable to `full_design` before you link the design. For details about this analysis mode, see [Waveform Propagation](#).

When this mode is enabled, you can display the piecewise linear waveforms in the path inspector window.

To display the piecewise linear waveform for a recalculated path,

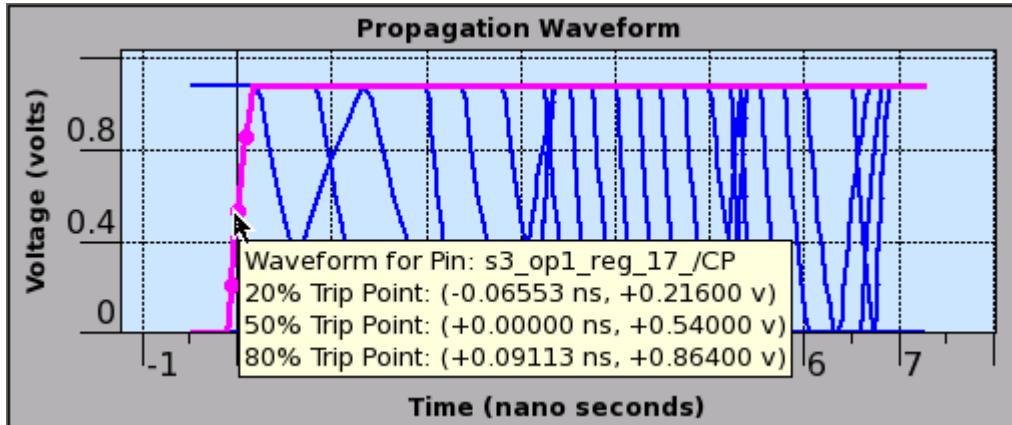
1. Select one or more recalculated paths.

For example, you can select a path in with a single recalculated timing path selected, choose Timing > Inspect Recalculated Path; or in the Path Analyzer or Path Comparison Table, use the pop-up menu and select Inspect Recalculated Path.

2. Choose Timing > Inspect Recalculated Path.
3. Click the Propagated Waveform button in the path inspector window.

[Figure 18-78](#) shows an example of a path-based waveform display in a path inspector window.

Figure 18-78 Path-Based Waveform Display



See Also

- [Analyzing Signal Integrity](#)
- [Inspecting Timing Path Elements](#)
- [Recalculating Timing Paths](#)

Analyzing Timing Paths from Multiple Scenarios

The PrimeTime GUI includes an interactive multi-scenario analysis flow, which allows you to restore timing path collections simultaneously. This flow allows you to analyze and debug timing paths from multiple scenarios, and it is intended to be used as a quick analysis tool to help debug and track designs that are improperly constrained. You can also load path collections and categorize the timing paths by the available attributes.

For information about these subjects, see

- [Interactive Multi-Scenario Analysis Flow](#)
 - [Viewing the Timing Paths in the Path Analyzer Window](#)
 - [Specifying a User-Defined Value for the Slack](#)
-

Interactive Multi-Scenario Analysis Flow

With the interactive multi-scenario analysis flow, you can

- Load and debug multiple collections of timing paths

The interactive multi-scenario analysis tool is designed to postprocess the timing paths across multiple scenarios within the same design.

- Categorize timing path collections

Load timing path collections from the same design and categorize them. You can use existing rules or create new category rules. For more information, see [Analyzing Collections of Timing Paths](#).

- Shift slacks of a category by a user-defined value

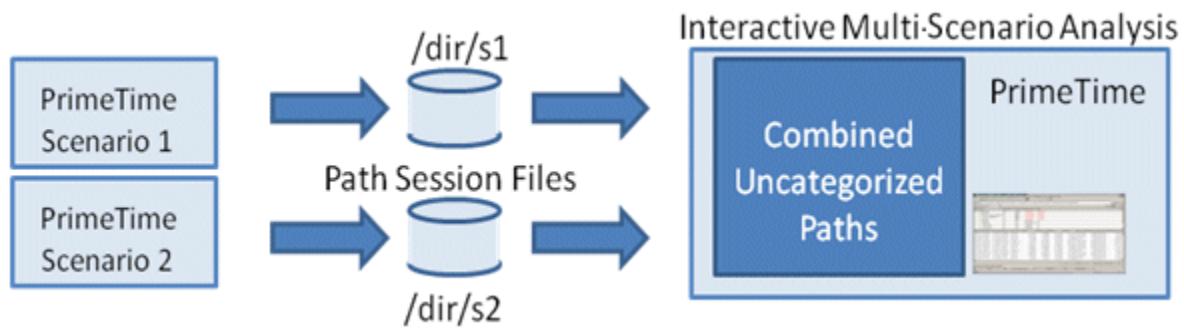
For more information, see [Specifying a User-Defined Value for the Slack](#).

- Reduce the number of times you run timing reports

In the Path Inspector window, you can configure the information presented in the timing report without having to rerun it each time. Change the setting to add or remove a column of information. For more information, see [Inspecting Timing Path Elements](#).

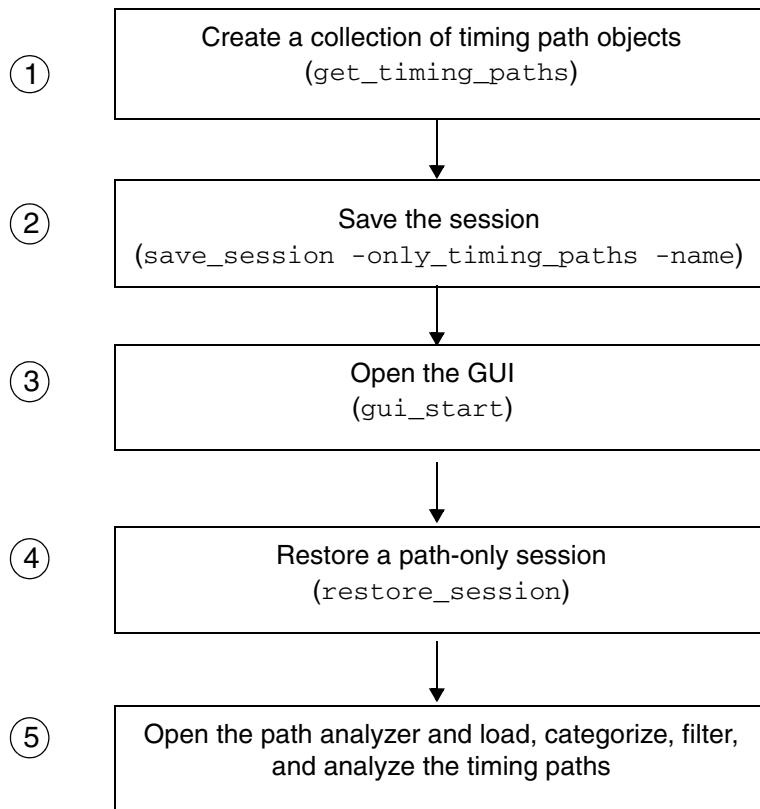
[Figure 18-79](#) shows the high-level flow for the interactive multi-scenario analysis tool. First, save only the timing paths from one or more PrimeTime SI sessions. Next, restore multiple timing paths from the same design into a single session, which automatically changes the tool to the interactive multi-scenario analysis mode. You can now analyze timing paths from multiple scenarios in the same session. You can then categorize the timing paths from multiple sessions using the path analyzer. Use the path inspector, schematics, and path data histograms to further analyze the timing paths.

Figure 18-79 Interactive Multi-Scenario Analysis in the PrimeTime SI Tool



The interactive multi-scenario analysis flow described in [Figure 18-80](#) shows the steps to create and save a collection of timing path objects. It also describes how to restore an analysis session and use the path analyzer to categorize and analyze the timing paths.

Figure 18-80 Interactive Multi-Scenario Analysis Flow in the PrimeTime SI Tool



The following steps explain the interactive multi-scenario analysis flow in more detail:

1. Use the `get_timing_paths` command to create a collection of timing paths objects. For example,

```
pt_shell> set paths [get_timing_paths -start_end_pair \
    -slack_lesser_than 0.0]
```

2. Save the timing path collection sessions using the `save_session` command with the `-only_timing_paths` option. The saved session is a quick representation of the PrimeTime SI session; any session data that is unrelated to the timing paths is omitted.

The `-only_timing_paths` option saves the timing path collection in a representation that is independent of the timing data. The `-name` option assigns a name for the saved path session. You cannot use the `-include` or `-only_used_libraries` option with either the `-only_timing_paths` or `-name` option.

The saved timing collection sessions are smaller than regular saved sessions, and the session size is proportional to the size of the timing path collection. Paths from a DMSA session can be saved using the remote execute feature.

The following example shows the `save_session` command:

```
pt_shell> save_session -only_timing_paths $paths /dir/s1
```

By default, the name assigned to the session is the directory name where the session is saved. You can use the `-name session_name` option to assign a different session name.

3. To start a PrimeTime SI session that includes the GUI, use the `pt_shell -gui` command to start the session or the `gui_start` command in `pt_shell`.
Ensure you have a PrimeTime SI license.
4. Use the `restore_session` command to restore a session. You can restore multiple sessions of the same design in a single interactive multi-scenario analysis GUI session.

Ensure the version of the PrimeTime tool used to restore a session with the `restore_session` command is the same version used when saving the session with the `save_session` command. Locate the version used to save the session from the `README` file located in the session directory.

You can incrementally load timing paths by using several instances of the `restore_session` command. For each restored session, the previously loaded timing path collections are maintained; they are not removed or overwritten. Path collections that are already loaded are kept, and another timing path collection is created. If you attempt to restore a collection containing the same name, the SR-023 error message appears. You can resolve this conflict by using the `-name session_name` option of the `restore_session` command.

By default, the name assigned to the paths is the one used when saving the session. If no name was assigned when saving the session, the directory name where the session

was restored is used. Alternatively, you can specify the name by using the `-name session_name` option. The following is an example of the `restore_session` command:

```
pt_shell> restore_session /dir/s1 [-name paths_s1]
pt_shell> restore_session /dir/s2
```

5. Use the path analyzer to categorize and analyze the timing paths.

In an interactive multi-scenario analysis, only attributes that are associated with the timing paths are available in the path analyzer.

The path analyzer includes timing overviews of the timing paths by their groups, startpoint cell, or endpoint cell. The path analyzer allows you to load multiple collections of timing paths within the same design using the `restore_session` command. This feature is supported only in the interactive multi-scenario analysis flow.

In addition, you can customize new category definitions for timing path collections and then populate the path analyzer with the new grouping criteria. After defining the attribute of the path to be categorized, access the user-defined category from the “View paths by” list.

For more information about the path analyzer, see [Viewing the Timing Paths in the Path Analyzer Window](#).

6. Shift the slack of a category by a user-defined value.

For more information about shifting slack, see [Specifying a User-Defined Value for the Slack](#).

7. Use schematics, the path inspector, and path data histograms to further analyze the timing paths.

If a GUI view or menu item is not supported, it is not available. For example, the abstract clock graph and clock analyzer views are not available.

8. To return to a regular PrimeTime SI session, use the `remove_design -all` command to remove the path collections and exit the interactive multi-scenario analysis mode. An informational message is issued to let you know that the interactive multi-scenario analysis is no longer available.

See Also

- [Viewing the Timing Paths in the Path Analyzer Window](#)

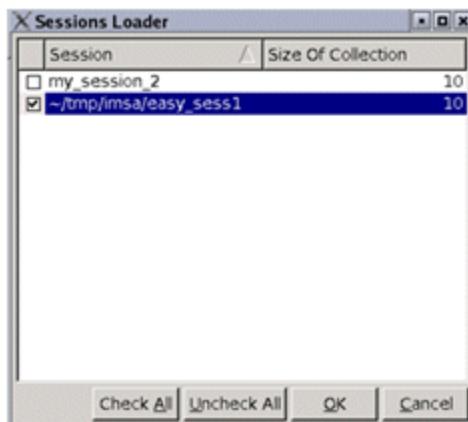
Viewing the Timing Paths in the Path Analyzer Window

To view the timing paths in the path analyzer,

1. Choose Timing > New Path Analyzer to open a new path analyzer window.
2. Click the Sessions button.

The Sessions Loader, shown in [Figure 18-81](#), appears showing all of the available sessions.

Figure 18-81 Sessions Loader Dialog Box



3. Select the sessions you want to analyze.

You can click Check All if you want to select all the available sessions. Click Uncheck All if you want to deselect all the sessions.

4. Click OK to load the information.

5. Click Apply in the path analyzer window.

Although there are separate sessions loaded and the paths have different session attributes, the paths appear in a single category called All. You can see the paths categorized by different sessions nested under the All category.

6. Categorize the timing paths;

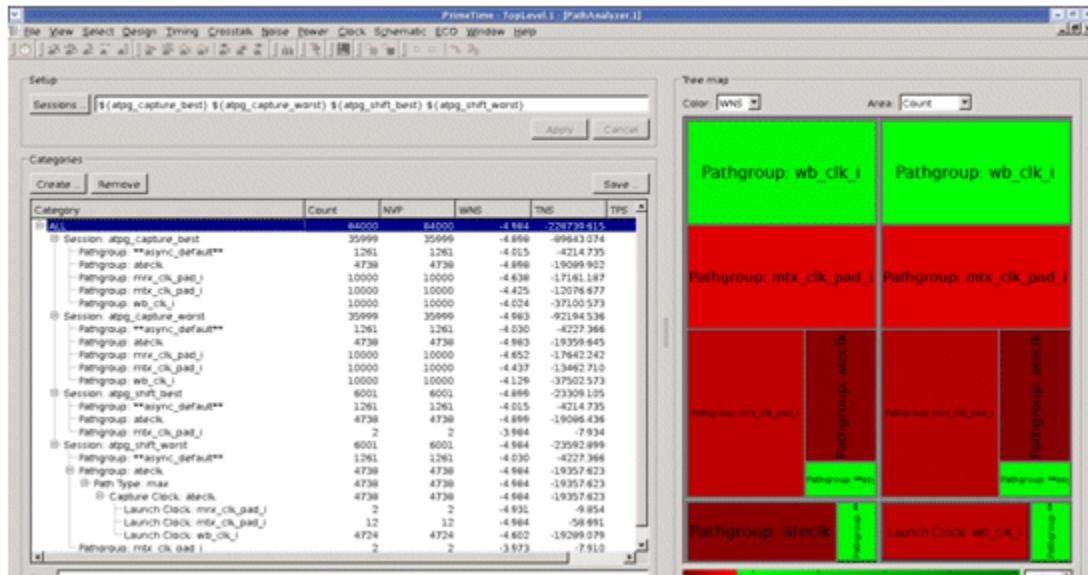
You can apply existing rules or create and apply custom rules. For more information, see [Specifying a User-Defined Value for the Slack](#).

7. Use the treemap view to evaluate the path groups.

The treemap displays hierarchical data in a tree structure as a set of nested rectangles. Each branch of the tree is displayed in a rectangle, which is then tiled with smaller rectangles representing subbranches. A rectangle representing a leaf node has an area proportional to a specified dimension on the data. The leaf nodes are colored to show a separate dimension of the data.

[Figure 18-82](#) shows an example of the path analyzer in the interactive multi-scenario analysis flow.

Figure 18-82 Path Analyzer Window - Interactive Multi-Scenario Analysis Flow



For more information about using the path analyzer, see [Analyzing Collections of Timing Paths](#).

See Also

- [Specifying a User-Defined Value for the Slack](#)
- [Interactive Multi-Scenario Analysis Flow](#)

Specifying a User-Defined Value for the Slack

In the interactive multi-scenario analysis flow, you can shift the slack of a category by a user-defined value in the path analyzer.

To shift the slack of a category,

1. Select a category in the path analyzer.
2. Right-click and choose Shift Category.

The Shift Histogram dialog box appears as shown in [Figure 18-83](#).

Figure 18-83 Shift Histogram Dialog Box



3. Select the slack shift value and parent category of the histogram.

4. Click OK.

The Table Histogram dialog box appears.

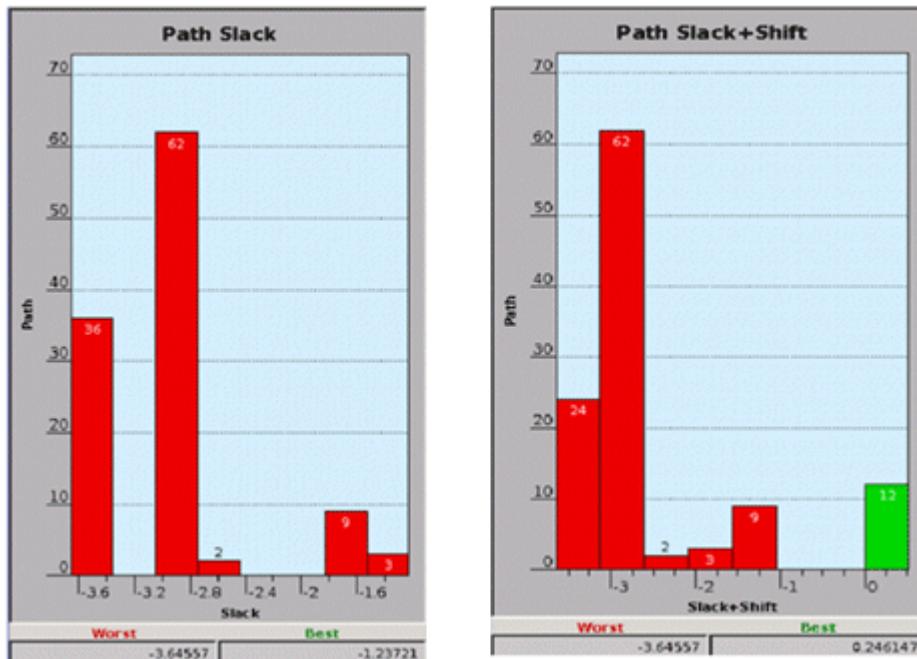
5. Select the type of histogram to view.

In the Column box, select either the Slack+Shift or Slack option. Select other options as needed.

6. Click OK.

Figure 18-84 shows what you might see if you select either the Slack or Slack+Shift option.

Figure 18-84 Path Slack and Path Slack+Shift



See Also

- [Viewing the Timing Paths in the Path Analyzer Window](#)
 - [Interactive Multi-Scenario Analysis Flow](#)
-

Menu Command Reference

The following sections list and briefly describe the commands on the menus in the PrimeTime main window, the schematic window, and the abstract5 clock graph window:

- [File Menu Commands](#)
- [View Menu Commands](#)
- [Select Menu Commands](#)
- [Design Menu Commands](#)
- [Timing Menu Commands](#)
- [Highlight Menu Commands](#)
- [Crosstalk Menu Commands](#)
- [Noise Menu Commands](#)
- [Power Menu Commands](#)
- [Clock Menu Commands](#)
- [Schematic Menu Commands](#)
- [ECO Menu Commands](#)
- [Window Menu Commands](#)
- [Help Menu Commands](#)

File Menu Commands

[Table 18-4](#) lists and briefly describes the commands on the File menu in the main window.

[Table 18-5](#) lists and briefly describes the commands on the File menu in the schematic and abstract clock graph windows.

Table 18-4 File Menu Commands in Main Window

Menu command	Action taken
File > Restore Session	Retrieves a previously saved session.
File > Save Session	Saves an existing session, including existing contents, used libraries, and noise data.
File > Execute Script	Executes a script just as when you execute a script from the source command.
File > Save Screenshot As	Enables you to select the image format and specify the file to save. A check box allows you to specify whether you want a snapshot of the active view window or the application window. The default is to take a snapshot of the active view and save the file in the PNG format.
File > Close GUI	Closes all PrimeTime GUI windows, leaving pt_shell active.
File > Exit	Exits completely from the PrimeTime session.

Table 18-5 File Menu Commands in Schematic and Abstract Clock Graph Windows

Menu command	Action taken
File > Save Screenshot As	Enables you to select the image format and specify the file to save. An option allows you to specify whether you want a snapshot of the active view window or the application window. The default is to take a snapshot of the active view and save the file in the PNG format.
File > Print	Prints the active schematic or abstract clock graph view, or saves the view in a PDF or PostScript file.
File > Close	Closes the currently active application window. If this is the only open window, the command closes the GUI and leaves you at the pt_shell prompt.

View Menu Commands

[Table 18-6](#) lists and briefly describes the commands on the View menu in the main window. [Table 18-7](#) lists and briefly describes the commands on the View menu in the schematic and abstract clock graph windows.

Table 18-6 View Menu Commands in Main Window

Menu command	Action taken
Preferences	Opens the Application Preferences dialog box.
View > Properties	Opens the Properties dialog box, in which you can view and edit properties such as name, type, and attribute values for a selected object, timing path, or clock.
View > Toolbars > <i>toolbar_name</i>	Displays or hides individual toolbars and panels.

Table 18-7 View Menu Commands in Schematic and Abstract Clock Graph Windows

Menu command	Action taken
View > Zoom > Zoom Full	Displays the entire schematic or clock graph in the view window.
View > Zoom > Zoom In	Doubles the magnification of the schematic or clock graph.
View > Zoom > Zoom Out	Shrinks the schematic or clock graph by half the magnification.
View > Zoom > Zoom Fit Selection	Adjusts the magnification level to fit the selected objects in the view.
View > Zoom > Zoom Fit Highlight	Adjusts the magnification level to fit the highlighted objects in the view.
View > Zoom > Pan to Selection	Recenters the schematic or clock graph in the view to display the selected objects.
View > Zoom > Pan to Highlight	Recenters the schematic or clock graph in the view to display the highlighted objects.
View > Zoom > Follow Selection	Enables or disables the mechanism that automatically zooms to an object when you select it, increasing or decreasing the magnification of the schematic or clock graph to fit the object within the view.

Table 18-7 View Menu Commands in Schematic and Abstract Clock Graph Windows (Continued)

Menu command	Action taken
View > Zoom > Back in Zoom and Pan History	Reverses the zoom level and pan position to the last zoom and pan setting in the view.
View > Zoom > Forward in Zoom and Pan History	Reapplies the next (previously reversed) zoom level and pan position in the view.
View > Mouse Tool > Selection Tool	Enables the Selection tool. You can select or deselect objects by clicking them or dragging a rectangle around them. To add objects to the current selection, press the Ctrl key while you click or drag the rectangle. To remove objects from the current selection, press the Shift key while you click or drag the rectangle.
View > Mouse Tool > Highlight Tool	Enables the Highlight tool. You can highlight objects by clicking them or dragging a rectangle around them. To remove highlighting from objects, press the Shift key while you click or drag the rectangle.
View > Mouse Tool > Query Tool	Enables the Query tool and, by default, displays the Query panel. When you click an object with the Query tool, the object appears in the query color, which is white by default but has a thinner line width than selection coloring, and information about the object, such as attribute values, appears on the Query panel.
View > Mouse Tool > Zoom In Tool	Enables the Zoom In tool. You can click where you want to recenter the schematic or clock graph and magnify it by a factor of 2, or drag a line diagonally across a rectangular area that you want to magnify to fill the view.
View > Mouse Tool > Zoom Out Tool	Enables the Zoom Out tool. You can click where you want to recenter the schematic or clock graph and shrink it by a factor of 2, or drag a line diagonally across a rectangular area to shrink the view by fitting it within the area.
View > Mouse Tool > Pan Tool	Enables the Pan tool. You can drag the schematic or clock graph in the direction that you want to move it within the view.
View > Mouse Tool > Fisheye Tool	Enables the Fisheye tool. You can move the tool over portions of an abstract clock graph view that you want to magnify. (This command is available only in the abstract clock graph window.)
View > Mouse Tool > Ruler Tool	Enables the Ruler tool. You can draw rulers by dragging horizontal lines in an abstract clock graph view. (This command is available only in the abstract clock graph window.)

Table 18-7 View Menu Commands in Schematic and Abstract Clock Graph Windows (Continued)

Menu command	Action taken
View > Mouse Tool > Clear Rulers	Removes all rulers from an abstract clock graph view. (This command is available only in the abstract clock graph window.)
View > Refresh	Redraws the active schematic or abstract clock graph view.
View > InfoTip	Enables or disables InfoTips in the active schematic or abstract clock graph window.
View > Property	Opens the Properties dialog box, in which you can view and edit properties such as name, type, and attribute values for a selected object, timing path, or clock.
View > Toolbars > <i>toolbar_name</i>	Displays or hides individual toolbars and panels.

Select Menu Commands

[Table 18-8](#) lists and briefly describes the commands on the Select menu.

Table 18-8 Select Menu Commands

Menu command	Action taken
Select > Clear	Clears (deselects) the current selection.
Select > By Name	Displays the Select By Name toolbar, which lets you interactively select cells, nets, ports, or pins by name, based on matching a specified pattern.
Select > Highlighted	Selects all highlighted objects or paths.
Select > Paths From/Through/ To	Selects paths based on from-to-through criteria or slack criteria.
Select > Fanin/Fanout	Selects objects in the fanin or fanout cones of specified objects.
Select > Driver of Selected	Selects the driver of the selected objects.
Select > Load of Selected	Selects the load of the selected objects.

Table 18-8 Select Menu Commands (Continued)

Menu command	Action taken
Select > Cells > <i>selection_type</i>	Selects cells based on their location in the hierarchy or their pin or net connections.
Select > Ports/Pins > <i>selection_type</i>	Selects ports and pins based on their type or connections.
Select > Nets > <i>selection_type</i>	Selects nets based on their connections to ports, pins, or cells.
Select > Clocks > <i>selection_type</i>	Selects clocks based on those that reach selected ports, pins, or cells. (This command is available only in the main window.)
Select > Selection List	Displays a dialog box that lists the currently selected objects. The selection list is updated automatically when you select or deselect objects.
Select > Query Selection	Generates a summary report for the selected objects.

Design Menu Commands

[Table 18-9](#) lists and briefly describes the commands on the Design menu in the main window.

Table 18-9 Design Menu Commands

Menu command	Action taken
Design > New Hierarchy Browser View	Opens a hierarchy browser view, in which you can traverse the design hierarchy and select objects.
Design > New Table View for Selection	Opens a data table view that shows the characteristics of the currently selected objects. The selected objects must all be the same type of object (for example, all cells).
Design > New Table View	Opens a data table view for a list of cells, nets, ports, pins, or clocks. You specify the object class and name-filtering criteria in a dialog box.
Design > <i>report_type</i>	Generates a text-format report on constraints, designs, cells, ports, nets, clocks, or wire load models in the design.

Timing Menu Commands

Table 18-10 lists and briefly describes the commands on the Timing menu in the main window.

Table 18-10 Timing Menu Commands

Menu command	Action taken
Timing > New Path Analyzer	Opens the path analyzer view window, which consists of the design status view and the timing path table.
Timing > New Timing Path Table View for Selected	Opens a new timing path table view that lists the characteristics of the currently selected timing paths.
Timing > New Timing Path Table View	Opens a new timing path table view. You specify the path criteria (<code>report_timing</code> options) in a dialog box.
Timing > Inspect Normal Path	Opens a path inspector window for the currently selected timing path. The path inspector shows a wide range of information about the path, including delay profiles, a path element table, clocking information, and timing waveforms.
Timing > Inspect Recalculated Path	Opens a path inspector window for the recalculated path. The path inspector shows a wide range of information about the path, including delay profiles, a path element table, clocking information, and timing waveforms.
Timing > Histogram > <i>histogram_type</i>	Displays a histogram of design data: endpoint slack, path slack, net capacitance, design rules, or timing bottleneck.
Timing > New Recalculated Path Comparison Table	Displays a table comparing original and recalculated paths.
Timing > New Recalculated Path Pin Comparison Table	Displays a table showing path through pin value differences between the original and recalculated path.
Timing > Net Capacitance Profile	Displays the relative amounts of net capacitance and pin capacitance for the currently selected net.
Timing > Check Timing	Displays a text-format report on the timing constraints, like the <code>check_timing</code> command.
Timing > Analysis Coverage	Displays a text-format report on the timing check coverage, like the <code>report_analysis_coverage</code> command.

Table 18-10 Timing Menu Commands (Continued)

Menu command	Action taken
Timing > Report Timing	Displays a text-format timing report, like the <code>report_timing</code> command.
Timing > Report Cell Timing	Displays a text-format cell timing report.
Timing > Clock Timing	Displays a text-format clock timing report, like the <code>report_clock_timing</code> command.
Timing > Update Timing	Updates the timing for the design, like the <code>update_timing</code> command.

Highlight Menu Commands

[Table 18-11](#) lists and briefly describes the commands on the Highlight menu in the schematic and abstract clock graph windows.

Table 18-11 Highlight Menu Commands

Menu command	Action taken
Highlight > Selected	Highlights selected objects or paths with the current highlight color.
Highlight > Clear Selected	Removes highlighting from selected objects or paths.
Highlight > Clear > <i>highlight_color</i>	Removes highlighting from objects and paths with the specified highlight color.
Highlight > Clear All	Removes highlighting from all highlighted objects and paths.
Highlight > Set Current Color > <i>highlight_color</i>	Sets the color for the next highlight operation.
Highlight > Next Color	Changes the color for the next highlight operation to the next color in the highlight color cycle.
Highlight > Auto Cycle Colors	Enables or disables automatic cycling through the highlight colors. Automatic cycling is disabled by default.
Highlight > Filter Highlighting	Highlights objects based on filtering rules that you define. (This command is available only in the schematic window.)

Table 18-11 Highlight Menu Commands (Continued)

Menu command	Action taken
Highlight > Critical Path	Highlights the critical path, which is the timing path with the worst slack in the design.
Highlight > Max Path > <i>delay_type</i>	Highlights the maximum delay path from, through, or to the selected objects.
Highlight > Min Path > <i>delay_type</i>	Highlights the minimum delay path from, through, or to the selected objects.
Highlight > Vth Cells > <i>action</i>	Use to highlight threshold voltage (Vt) cells in the PrimeTime PX tool.

Crosstalk Menu Commands

[Table 18-12](#) lists and briefly describes the commands on the Crosstalk menu in the main window.

Table 18-12 Crosstalk Menu Commands

Menu command	Action taken
Crosstalk > Delta Delay Histogram	Displays a histogram of crosstalk delay data.
Crosstalk > Path Delta Delay Histogram	Displays a histogram of crosstalk path delay data.
Crosstalk > Bump Voltage Histogram	Displays a histogram of crosstalk bump voltage.
Crosstalk > Accumulated Bump Voltage Histogram	Displays a histogram of crosstalk accumulated bump voltage.
Crosstalk > Coupling Analysis For Selected Paths	Populates the coupling analysis table with the information for the “n” worst paths, based on all the signal integrity delays in the selected paths.
Crosstalk > Coupling Analysis For Selected Nets	Populates the coupling analysis table with the information for the selected nets.

Table 18-12 Crosstalk Menu Commands (Continued)

Menu command	Action taken
Crosstalk > Coupling Analysis For SI Bottleneck Nets	Populates the signal integrity victim analysis table with the information for the bottleneck nets. (This is analogous to running the <code>report_si_bottleneck</code> command.)

Noise Menu Commands

[Table 18-13](#) lists and briefly describes the commands on the Noise menu in the main window.

Table 18-13 Noise Menu Commands

Menu command	Action taken
Noise > Noise Slack Histogram	Displays a histogram of noise slack.
Noise > Noise Bump Histogram	Displays a histogram of noise bump.
Noise > Accumulated Noise Bump Histogram	Displays a histogram of accumulated noise bump.
Noise > Noise Immunity Curve	Displays a plot of the noise immunity characteristics of the cell at the currently selected input pin, as specified by the noise immunity table in the library definition of the cell.

Power Menu Commands

[Table 18-14](#) lists and briefly describes the commands on the Power menu in the main window.

Table 18-14 Power Menu Commands

Menu command	Action taken
Power > Show Power Analysis Driver	Displays the analysis driver so that you can analyze dynamic power and static power. You view total power consumption for the current design with its distribution between dynamic, switching, internal, and leakage consumption.

Table 18-14 Power Menu Commands (Continued)

Menu command	Action taken
Power > Show Leakage Variation Data	Displays the Leakage Variation Analysis dialog box so that you can analyze the data.
Power > Net Power Design Map > <i>map_type</i>	Displays the total power density as a map of the design. The power density displays as a treemap that is a visualization tool. You can use this tool to help you identify anomalies in large hierarchical data sets.
Power > Histogram of selected > <i>histogram_type</i>	Use to browse the design hierarchy. You can view the cell hierarchical, pin, port, pin of child cell, net, and clock attributes.
Power > View Waveforms	Displays your switching activity in a waveform format.

Clock Menu Commands

Table 18-15 lists and briefly describes the commands on the Clock menu in the main window. **Table 18-16** lists and briefly describes the commands on the Clock menu in the schematic window.

Table 18-15 Clock Menu Commands in Main Window

Menu command	Action taken
Clock > Clock Analyzer	Opens a clock analyzer view window that provides a central place to run various clock analysis tasks.
Clock > Clock Graph for All Clocks	Opens an abstract clock graph view that helps you to identify the relationships between different clocks, such as primary and generated, clock convergence points, classes of clocked elements, and logic levels.
Clock > Schematic for Selected Clocks	Opens a schematic view showing the selected clocks.
Clock > Schematic for Unclocked Pins	Opens a schematic view showing all unclocked pins in the design.

Table 18-16 Clock Menu Commands in Schematic Window

Menu command	Action taken
Clock > Highlight Propagation for Selected Clock(s) by Color > <i>highlight_color</i>	Highlights the propagation paths for selected clocks using the specified highlight color in a clock schematic.
Clock > Clear Highlight Propagation for Selected Clock(s)	Removes highlighting from the propagation paths for selected clocks in a clock schematic.
Clock > Clear All Highlight Colors in Schematic	Removes highlighting from all clock objects and paths in a clock schematic.
Clock > Highlight Collapsing of Selected	Highlights buses containing selected objects using the current highlight color in a clock schematic.
Clock > Expand > <i>expand_type</i>	Expands metaobjects in a clock schematic. You can expand all selected metaobjects, all unselected metaobjects, or the first or last logic level (fanin or fanout) of the selected metaobjects.
Clock > Collapse > <i>collapse_type</i>	Collapses objects into metaobjects of the same type in a clock schematic. You can collapse all selected objects or all unselected objects.

Schematic Menu Commands

[Table 18-17](#) lists and briefly describes the commands on the Schematic menu in the main window. [Table 18-18](#) lists and briefly describes the commands on the Schematic menu in the schematic and abstract clock graph windows.

Table 18-17 Schematic Menu Commands in Main Window

Menu command	Action taken
Schematic > New Schematic View	Opens a new schematic window and displays the selected objects and timing paths in an instance schematic or the selected clocks in a clock schematic.

Table 18-18 Schematic Menu Commands in Schematic and Abstract Clock Graph Windows

Menu command	Action taken
Schematic > New Schematic View	Opens a new schematic window and displays the selected objects and timing paths in an instance schematic or the selected clocks in a clock schematic.
Schematic > Move Down	Displays the instance schematic for the parent cell at the next higher level in the design hierarchy.
Schematic > Move Up	Displays the instance schematic for the selected design instance (hierarchical cell) at the next lower level in the design hierarchy.
Schematic > Back	Reverses the most recent action that you performed in the active schematic view, and redisplays the previous schematic in the active schematic view. You can sequentially reverse a series of actions.
Schematic > Forward	Reapplies the most recently reversed action in the active schematic view. If you have reversed one or more actions, you can reapply the most recently reversed action or a series of actions.
Schematic > Add Paths From/Through/To	Adds worst slack timing paths in the active schematic view, but does not change the netlist or affect other schematic views. You can add one or more paths with the worst slack in the current design, in a path group, or from, to, or through selected objects.
Schematic > Add Fanin/Fanout	Adds one or more levels of fanin or fanout logic in the active schematic view, but does not change the netlist or affect other schematic views. You can add either the fanin logic to selected objects or the fanout logic from selected objects.
Schematic > Add Selected	Adds selected objects in the active schematic view, but does not change the netlist or affect other schematic views.
Schematic > Remove Selected	Removes selected objects from the active schematic view, but does not change the netlist or affect other schematic views.
Schematic > Select Fanin Cone	Selects the fanin logic to the selected objects.
Schematic > Select Fanout Cone	Selects the fanout logic from the selected objects.
Schematic > Expand > <i>object_type</i>	Expands metaobjects in an instance schematic. You can expand selected hierarchical cells, all hierarchical cells, all buffers and inverter metacells, or all unconnected macro metapins.

Table 18-18 Schematic Menu Commands in Schematic and Abstract Clock Graph Windows (Continued)

Menu command	Action taken
Schematic > Collapse > <i>object_type</i>	Collapses objects into metaobjects in an instance schematic. You can collapse all hierarchical cells, hierarchical cells containing selected objects, all buffer and inverter chains, all buffer and inverter trees, or all unconnected macro pins.
Schematic > Show Logical Hierarchy	Organizes the schematic hierarchically and displays the hierarchical cell boundaries in an instance schematic.

ECO Menu Commands

[Table 18-19](#) lists and briefly describes the commands on the ECO menu.

Table 18-19 ECO Menu Commands

Menu command	Action taken
ECO > Insert Buffer	Inserts a buffer at the selected port or pin.
ECO > Insert Buffer at Net Driver	Inserts a buffer at the drivers for the selected nets.
ECO > Size Cell	Resizes the selected cell.
ECO > Size Driver	Resizes the drivers of selected nets.
ECO > Upsize Driver	Enlarges the drivers of the selected nets.
ECO > Downsize Driver	Downsizes the drivers of the selected nets.
ECO > Set Coupling Separation	Sets global coupling separation for selected nets.
ECO > Set Pairwise Coupling Separation	Sets pairwise coupling separation for the selected nets. This is only available for PrimeTime SI flows.

Window Menu Commands

Table 18-20 lists and briefly describes the commands on the Window menu.

Table 18-20 Window Menu Commands

Menu command	Action taken
Window > New Main Window	Opens a new instance of the main window.
Window > New Console	Opens a new console view window for running pt_shell commands, viewing the command history, and viewing errors and warnings.
Window > Close Window	Closes the currently active application window. If this is the only open window, the command closes the GUI and leaves you at the pt_shell prompt.
Window > Close All Windows (Close GUI)	Closes all application windows, which closes the GUI and leaves you at the pt_shell prompt.
Window > Next Window	Cycles to the next application window and brings it to the front.
Window > Previous Window	Cycles to the previous application window and brings it to the front.
Window > Close View	Closes the active view within the active application window. (This command is available only in the main window.)
Window > Close All View	Closes all of the view windows within the active application window. (This command is available only in the main window.)
Window > Tile	Arranges all the view windows side by side within the active application window. (This command is available only in the main window.)
Window > Cascade	Cascades all the view windows into a stack of windows within the active application window. (This command is available only in the main window.)
Window > Next View	Cycles to the next view window within the active application window and brings it to the front. (This command is available only in the main window.)
Window > Previous View	Cycles to the previous view window within the active application window and brings it to the front. (This command is available only in the main window.)
Window > <i>application_window_name</i>	Makes the named application window active and brings it to the front.

Table 18-20 Window Menu Commands (Continued)

Menu command	Action taken
Window > <i>view_window_name</i>	Makes the named view window active and brings it to the front in the active application window.

Help Menu Commands

[Table 18-21](#) lists and briefly describes the commands on the Help menu.

Table 18-21 Help Menu Commands

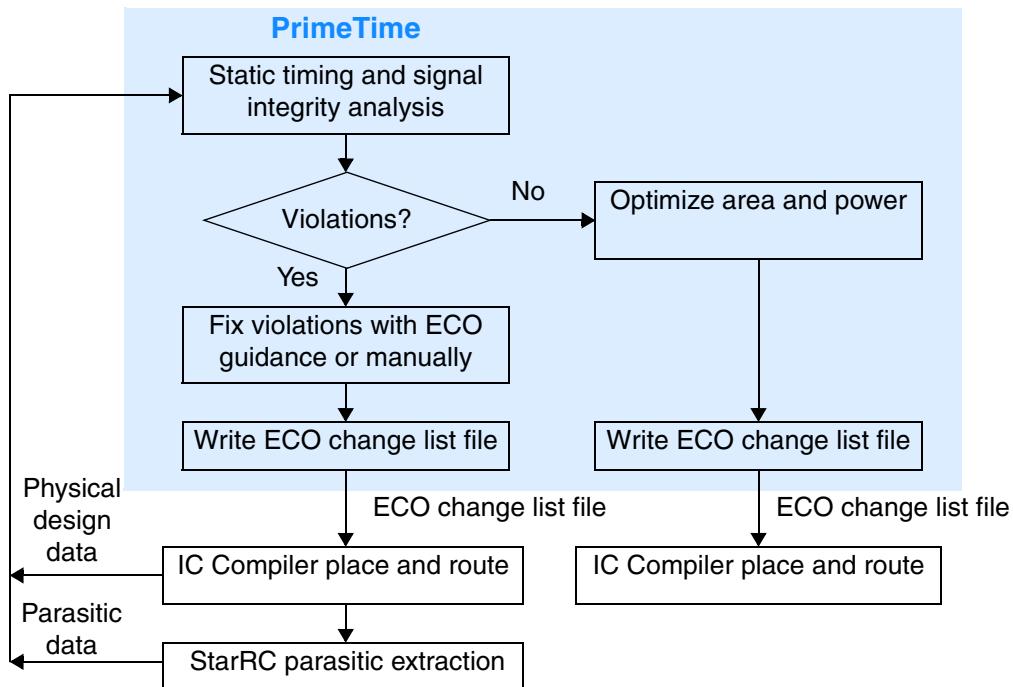
Menu command	Action taken
Help > Man Pages	Opens the man page viewer, in which you can view, search, and print man pages.
Help > Report Hotkey Bindings	Opens a report view listing the keyboard shortcuts that are available in the active application window.
Help > Getting Help	Provides an overview of the different resources available, such as user guides, release notes, and SolvNet articles.
Help > About	Shows the PrimeTime version number and copyright notice.

19

ECO Flow

If your design has timing and design rule violations, you can follow the Synopsys Galaxy engineering change order (ECO) flow to fix the violations, implement the changes, and rerun timing analysis.

Figure 19-1 *Synopsys Galaxy ECO Flow*



To learn about the PrimeTime ECO flow, see

- [Setting Options for ECO](#)
- [Excluding the Usage of Specific Library Cells From ECO](#)
- [Handling Multiply Instantiated Modules](#)
- [Fixing Design Rule and Timing Violations With ECO Guidance](#)
- [Performing Power Recovery](#)
- [Manual Netlist Editing](#)
- [Writing Change Lists](#)
- [Implementing the Changes With the Galaxy Design Platform](#)

Setting Options for ECO

Before you start the ECO, use the `set_eco_options` command to specify the information in [Table 19-1](#).

Table 19-1 ECO Options

To specify this information...	Use this option...
information for physically-aware ECO	<code>-physical_lib_path</code> <code>-physical_design_path</code> <code>-physical_constraint_file</code>
Multiply instantiated module (MIM) groups	<code>-mim_group</code>
Filler cells	<code>-filler_cell_names</code>
Output log file	<code>-log_file</code>

To show the ECO options, use the `report_eco_options` command. To reset the ECO options, use the `reset_eco_options` command.

Configuring the ECO for Multiple Libraries

If the tool must differentiate between multiple libraries with the same logical file name, specify how the tool records library cell names in the ECO change list by setting the `eco_write_changes_prepend_libname_to_libcell` and `eco_write_changes_prepend_libfile_to_libcell` variables. By default, these variables are set to `false`. These variables affect all change list output formats.

Excluding the Usage of Specific Library Cells From ECO

To exclude specific library cells from being used for ECO, set one of the following attributes on those library cells:

- [The dont_use Application Attribute](#)
- [The pt_dont_use User-Defined Attribute](#)

The `dont_use` Application Attribute

To apply the `dont_use` application attribute to a specified cell list, use this command:

```
set_dont_use cell_list [true | false]
```

The `set_dont_use` command sets the `dont_use` attribute to `true` or `false` on the specified library cells; the default is `true`. During ECO, the tool does not use a cell if its `dont_use` attribute is `true`.

The following example shows how to prevent BUFX1 cells from being used:

```
pt_shell> set_dont_use [get_lib_cells BUFX1] true
```

The `pt_dont_use` User-Defined Attribute

To apply the `pt_dont_use` user-defined attribute, use the following Tcl procedure:

```
define_user_attribute -quiet -type boolean \
    -classes lib_cell pt_dont_use
proc set_pt_dont_use_lib_cell { lib_cell } {
    set_user_attribute -quiet -class lib_cell \
        [get_lib_cells -quiet $lib_cell]
    pt_dont_use true
}
```

Handling Multiply Instantiated Modules

PrimeTime ECO handles multiply instantiated modules (MIM) in the following ways:

- Multiply instantiated modules have equivalent netlist changes during ECO.
- Multiply instantiated modules have one change list file for IC Compiler implementation.
- ECO generates block-level and top-level change list files.

To use multiply instantiated modules in the ECO flow, follow these steps:

1. Set the `eco_enable_mim` variable:

```
pt_shell> set_app_var eco_enable_mim true
```

2. (Optional) Define groups of multiply instantiated modules by using the `set_eco_options` command with the `-mim_group` option. To specify multiple groups, use this command multiple times:

```
pt_shell> set_eco_options -mim_group {CPU1 CPU2}
pt_shell> set_eco_options -mim_group {CPU3 CPU4}
```

Note:

Using the `-mim_group` option might use more runtime and memory during ECO. If you do not use this option, PrimeTime ECO automatically detects multiply instantiated modules from the DEF and parasitic files.

3. Report the groups of multiply instantiated modules by using the `report_eco_options` command:

```
pt_shell> report_eco_options
...
Instance groups:
Group Module Instances
-----
1      CPU      CPU1 CPU2
2      CPU      CPU3 CPU4
```

4. Perform ECO fixing by using the `fix_eco_timing`, `fix_eco_drc`, and `fix_eco_power` commands.
5. Generate the ECO change list files by using the `write_changes` command:

```
pt_shell> write_changes -format icctcl -output pteco.tcl
```

This command creates the following change lists, where the change list file names are user-specified file names prepended with their module names:

- `pteco.tcl` – Change list for the top-level module.
- `CPU_pteco.tcl` – Change list for the CPU module for instances CPU1 and CPU2.
- `CPU_0_pteco.tcl` – Change list for the CPU module for instances CPU3 and CPU4.

Note:

This is for a custom configuration example; by default, all changes in the CPU module would be one single ECO file.

Fixing Design Rule and Timing Violations With ECO Guidance

If your design has numerous violations that make manual netlist editing impractical, PrimeTime SI and PrimeTime ADV can provide ECO guidance to automatically fix timing and design rule violations. With this capability, you can evaluate possible fixes quickly and minimize time-consuming iterations in the physical implementation flow.

To learn about automated fixing with ECO guidance, see

- [Recommended Automated Fixing Flow](#)
- [Fixing Design Rule Violations](#)
- [Fixing Setup and Hold Timing Violations](#)

- Physically-Aware ECO
- Prioritizing the Fixing of Timing Violations Over Design Rule Violations
- Multi-Scenario ECO Fixing With Constrained Resources

Recommended Automated Fixing Flow

Setup or hold fixing does not degrade design rule violations, but fixing design rule violations can degrade setup or hold violations because fixing design rule violations has the highest priority. Also, because setup violations are harder to fix, hold fixing preserves setup slack, but setup fixing is permitted to introduce some hold violations. Therefore, you should first fix design rule violations, followed by setup violations, and then hold violations, as shown in the following flow:

1. Ensure that your design is fully placed and routed, with generated clock trees. If these conditions are not met, you might make design changes that prove to be unnecessary later.
2. (Optional) Specify the maximum area ratio increase for cell resizing by setting the `eco_alternative_area_ratio_threshold` variable. The default of this variable is 2.0, which specifies that the resized cell can be no more than twice as large as the current cell. To eliminate the area restrictions on cell resizing, set the variable to 0.
3. (Optional) Exclude specific library cells from being used during ECO by setting the `dont_use` or `pt_dont_use` attribute on those library cells. (See [Excluding the Usage of Specific Library Cells From ECO](#).)
4. Fix design rule violations with the `fix_eco_drc` command. (See [Fixing Design Rule Violations](#).)
5. Fix setup and hold violations with the `fix_eco_timing` command. (See [Fixing Setup and Hold Timing Violations](#).)

Fixing Design Rule Violations

To fix or reduce maximum transition, maximum capacitance, and maximum fanout design rule violations, while minimizing the effect on timing and area, use the `fix_eco_drc` command. Because the tool fixes design rule violations by resizing cells and inserting buffers, the `fix_eco_drc` command automatically specifies the `size_cell` and `insert_buffer` commands. Although the tool prioritizes fixing design rule violations over timing violations, the fixes are chosen to minimize the effect on timing.

The following example shows how to fix maximum-transition design rule violations using cell resizing and generate a verbose fixing report:

```
pt_shell> fix_eco_drc -type max_transition -methods {size_cell} -verbose
```

The following example shows how to fix only maximum transition design rule violations using only buffer insertion with buffers BUFX1, BUFX2, and BUFX3 and generate a verbose fixing report:

```
pt_shell> fix_eco_drc -type max_transition -methods {insert_buffer} \
           -buffer_list {BUFX1 BUFX2 BUFX3} -verbose
```

You can specify a timing context for fixing design rule violations with the `-setup_margin` and `-hold_margin` options, based on your knowledge of the state of the design or phase of the design cycle. If you do not specify these options, then the setup and hold margins default to negative infinity, which gives PrimeTime the freedom to fix design rule violations even if timing degrades.

To minimize the timing effect, set the margins to zero or a small nonzero number. In this case, PrimeTime does not perform a fix that

- Degrades a positive setup or hold slack below zero
- Degrades a negative setup or hold slack further

However, if a negative slack could be improved by fixing a design rule violation, such a fix is applied regardless of any specified margin values.

Note:

The `fix_eco_drc` command tries to preserve—but does not guarantee—the user-specified margin values. This might result in a lower fixing rate for design rule violations.

The fixing report shows the number of design rule violations that remain unfixed. For example:

Remaining Violations:	
Violation Type	Count
Total remaining violations	2
Unfixable violations	1

Fixing Setup and Hold Timing Violations

To fix or reduce timing violations while minimizing new timing or design rule violations, use the `fix_eco_timing` command. This command is similar to the `fix_eco_drc` command, except that it fixes timing violations instead of design rule violations. When the analysis contains crosstalk data, coupling information is also used during the fixing process.

To fix setup and hold timing violations:

1. Fix setup violations with the `fix_eco_timing -type setup` command. For example:

```
pt_shell> fix_eco_timing -type setup -methods size_cell ...
```

Because setup violations are more difficult to fix than hold violations, PrimeTime avoids introducing new design rule violations but introduces new hold violations if necessary. Setup fixing works through cell resizing, which is specified with the `-methods size_cell` option.

2. Fix hold violations with the `fix_eco_timing -type hold` command. For example:

```
pt_shell> fix_eco_timing -type hold \
    -methods {size_cell insert_buffer_at_load_pins} \
    -buffer_list {BUFX1 BUFX2}
```

Examining the ECO Results and the Reasons for Unfixed Violations

If you use the `fix_eco_timing` command with the `-verbose` option, the report shows details about the violation count in each scenario and an ECO summary. For example:

```
Information: 303 violating endpoints located in the
scenario_min scenario...
```

```
Information: 447 violating endpoints located in all scenarios...
```

Violation Totals:

Scenario	Count
<hr/>	
scenario_max	144
scenario_min	303
<hr/>	
Total	447

```
Information: Creating database for fixing...
```

```
Information: Fixing violations...
```

```
Information: Finalizing ECO change list...
```

```
Information: Updating timing...
```

```
Information: Analyzing timing improvement...
```

```
Information: Tuning changes...
```

ECO Summary:

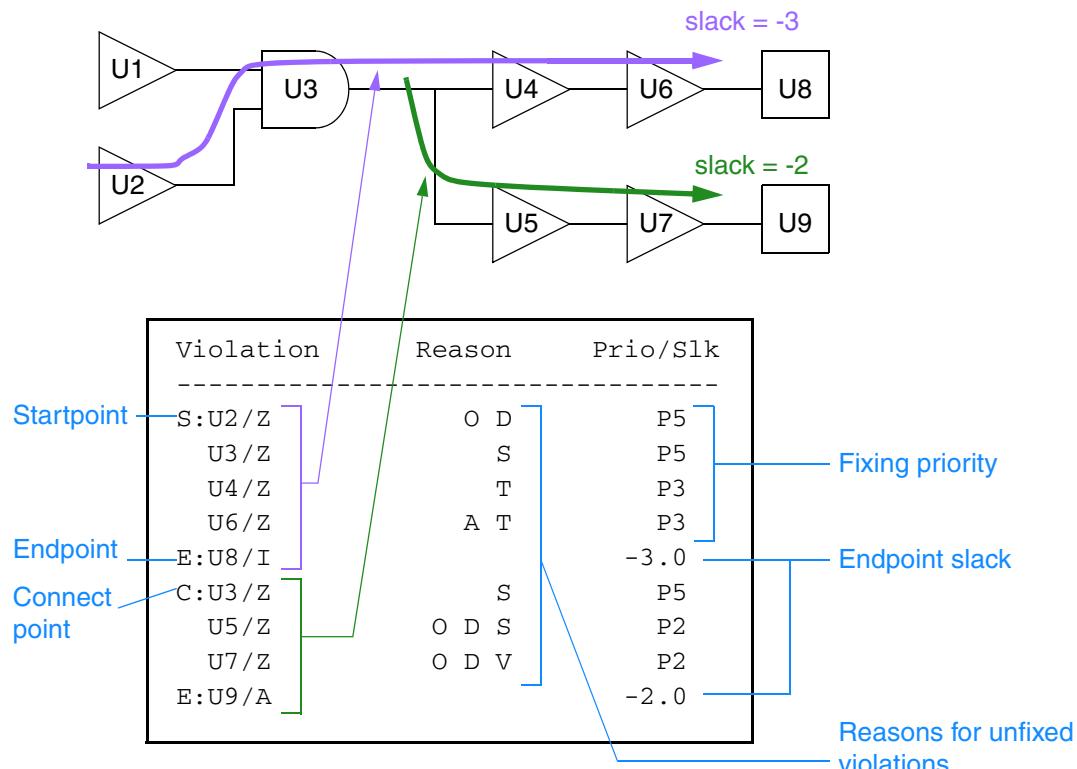
Number of size_cell commands	3
Number of insert_buffer commands	111
Total number of commands	114
Area increased by cell sizing	-3.00
Area increased by buffer insertion	900.00
Total area increased	897.00

The verbose report can also show information about unfixed timing violations, including the reasons they could not be fixed and the criticality of each violation. To show this information,

specify the maximum number of unfixed violations to report by setting the `eco_report_unfixed_reason_max_endpoints` variable to a positive integer. By default, the variable is set to 0.

Figure 19-2 shows a circuit example and its corresponding report. The fixing priority, ranging from P0 (the lowest fixing priority) to P9 (the highest fixing priority) indicates the effectiveness of fixing the cell. For example, fixing a cell with P5 priority would potentially fix more endpoint violations than fixing a cell with P0 priority.

Figure 19-2 The fix_eco_timing Report of Unfixed Timing Violations



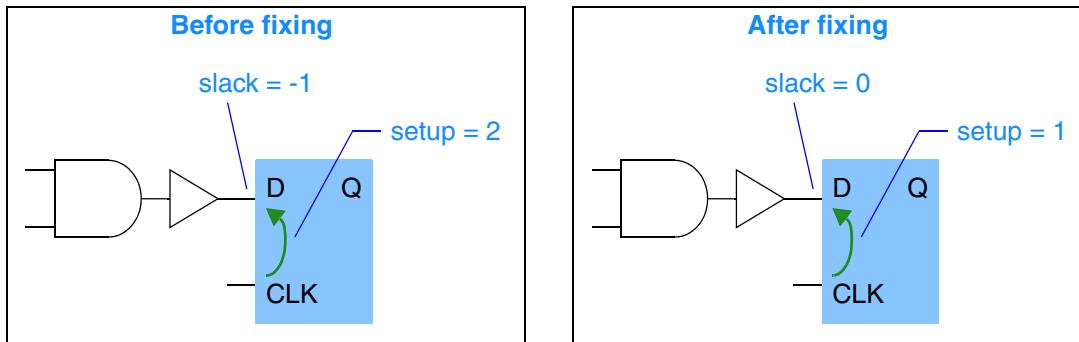
For a description of the reason codes in the report, see the `fix_eco_timing` man page.

Resizing Sequential Cells

With PrimeTime ADV, you can fix timing violations by resizing sequential cells. To do this, use the `fix_eco_timing` command with the `-cell_type sequential` option.

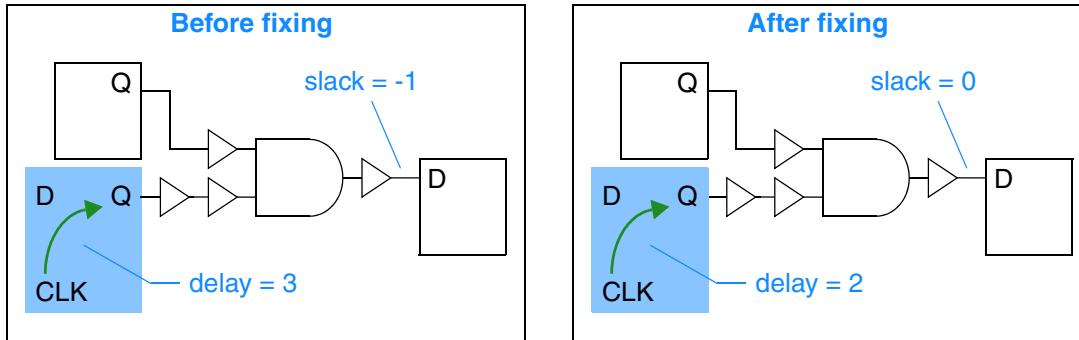
Figure 19-3 shows how the tool fixes violations at the input of sequential cells to improve setup and hold constraint arc delays.

Figure 19-3 Resizing a Sequential Cell to Fix a Violation at the Input



[Figure 19-4](#) shows how the tool fixes violations at the output of sequential cells to improve clock-to-Q arc delays.

Figure 19-4 Resizing a Sequential Cell to Fix a Violation at the Output



Note:

Because changing sequential cells can affect clock tree timing, you should use this capability only when absolutely necessary. Moreover, you must use a place-and-route tool to verify that the resulting changes do not cause any timing degradation.

Prioritizing the Fixing of Timing Violations Over Design Rule Violations

In the late stages of an ECO cycle, a design might still have a few timing violations. Although design rule and setup violations can be selectively waived, hold violations cannot be waived for a tapeout. At this point, you might want to take a calculated risk and fix a hold violation at the expense of possibly degrading a design rule violation.

To prioritize the fixing of timing violations over design rule violations, use the `fix_eco_timing` command with the `-ignore_drc` option. When you specify this option, PrimeTime SI ignores the `max_transition`, `max_capacitance`, and `max_fanout` design rule violations.

Note:

Using the `-ignore_drc` option could degrade design rule violations.

If you do not specify the `-ignore_drc` option, PrimeTime ECO does not fix a timing violation when there are any design rule violations on the path. This is the default behavior.

Multi-Scenario ECO Fixing With Constrained Resources

PrimeTime has the ability to run a distributed multi-scenario analysis (DMSA) ECO even if the number of scenarios exceeds the number of available hosts. You can use this capability when running the `fix_eco_drc` and `fix_eco_timing` commands.

To enable DMSA ECO fixing with more scenarios than hosts, set the `eco_enable_more_scenarios_than_hosts` variable to `true`; by default, it is `false`.

Minimum Host Requirement

To run DMSA ECO with constrained resources, you need at least four available hosts. PrimeTime recommends the number of hosts to be at least eight or one-fourth of the number of scenarios, whichever is larger.

Disk Space Requirement

When running DMSA ECO with constrained resources, you need enough disk space in the DMSA working directory to store the session data of all the scenarios. For example, if you run 32 scenarios, and each scenario requires 1 GB of disk space, then you must have at least 32 GB of disk space allocated for the DMSA working directory.

Runtime Versus Number of Hosts

As the number of available hosts decreases, the turnaround time increases linearly. The increase in the turnaround time depends on the size of the design, number of violations, type of the scenarios, network performance, and available disk space.

Optimizing DMSA Scripts for Faster Turnaround Time

If fewer hosts are available than scenarios, the hosts run in scenario swapping mode, where they rotate and swap scenarios one after the other. This results in significant performance degradation.

To avoid scenario swapping and achieve faster runtimes, minimize the number of merged reporting commands, such as the `report_timing` and `report_analysis_coverage` commands. Also, reduce the number of `remote_execute` commands and merge them into

one `remote_execute` block. For example, the following script invokes four rounds of scenario swapping:

```
remote_execute { set_false_path ... }
remote_execute { set_false_path ... }
remote_execute { set_dont_touch ... }
remote_execute { set_dont_touch ... }
```

If you merge the `remote_execute` commands into one block, the script invokes only one round of scenario swapping, as shown in the following example:

```
remote_execute {
    set_false_path ...
    set_false_path ...
    set_dont_touch ...
    set_dont_touch ...
}
```

Table 19-2 shows an example of script modifications for faster turnaround time.

Table 19-2 Example of Script Modifications for Faster Turnaround Time

<pre>##### # Full-host version works with fewer hosts ##### restore_dmsa_session current_session -all current_scenario -all # # Set configuration remote_execute {set_false_path ...} remote_execute {set_false_path ...} ... remote_execute { set \ eco_alternative_area_ratio_threshold 2.0 ... } # Fix design rule violations fix_eco_drc ... # # Fix timing violations fix_eco_timing -type setup fix_eco_timing -type hold \ -buffer_list {BUF1 BUF2 BUF4} # # OK to write in all scenarios because # they are written simultaneously # remote_execute { write_changes -output my_eco.tcl }</pre>	<pre>##### # Faster version avoids scenario swapping ##### restore_dmsa_session current_session -all current_scenario -all # # Merge configurations into one block remote_execute { set_false_path ... set_false_path set \ eco_alternative_area_ratio_threshold 2.0 ... } # Fix design rule violations fix_eco_drc ... # # Fix timing violations fix_eco_timing -type setup fix_eco_timing -type hold \ -buffer_list {BUF1 BUF2 BUF4} # # Write to only one scenario to avoid # scenario swapping during write_changes current_scenario scen1 remote_execute { write_changes -output my_eco.tcl }</pre>
--	--

For information about the `restore_dmsa_session` Tcl procedure, see [SolvNet article 018039, “How Can I Restore Saved Sessions in Distributed Multiscenario Analysis?”](#)

Performing Power Recovery

PrimeTime ADV can optimize your design by performing power recovery. The tool recovers power on paths with positive slack by swapping cells based on

- Cell area
- Threshold voltage
- User-defined power attribute

To perform power recovery:

1. (Optional) Guide power recovery with the following controls:
 - Exclude specific library cells as candidates for swapping by using the `set_dont_use` command to set the `dont_use` attribute.
 - Restrict alternative library cells to specific attributes by setting the `eco_alternative_cell_attribute_restrictions` variable.
 - Exclude unconstrained cells as candidates for swapping by setting the `eco_power_exclude_unconstrained_cells` variable to `true`; the default is `false`.
2. Perform power recovery by using the `fix_eco_power` command.

3. List the alternative library cells for ECO by using the `report_eco_library_cells` command. For example:

```
pt_shell> report_eco_library_cells

*****
Report : eco_library_cells
...
*****
Alternative library cells:

Attributes:
  u - dont_use or pt_dont_use
  d - dont_touch

Group Library_cell          Area Attributes
-----
[ 0] core_typical/hvt_buf_1    4.50
     core_typical/hvt_buf_2    5.40
     core_typical/hvt_buf_3    6.30
     core_typical/hvt_dly2    8.10 u
[ 1] core_typical/hvt_inv_1    3.60
     core_typical/hvt_inv_2    4.50
     core_typical/hvt_inv_3    5.40
```

4. Report the cells that are used in the design by using the `report_cell_usage` command. For example:

```
pt_shell> report_cell_usage

*****
Report : cell_usage
...
*****
Cell Group      Count          Area
-----
Combinational   3977357 ( 85%)   5496541.50 ( 22%)
Sequential      667986 ( 14%)   9514767.00 ( 38%)
Clock           42891 (  1%)   2975476.75 ( 12%)
Others          973 (  0%)   6873247.50 ( 28%)
-----
Total           4689207 (100%)  24860032.00 (100%)
```

5. Report the power recovery results by using the `report_power` command. For example:

```
pt_shell> set_app_var power_enable_analysis true
pt_shell> report_power -groups {register combinational sequential}

          Internal   Switching   Leakage   Total
Power Group        Power       Power      Power (%)      Attrs
-----
register           2.986e-03 1.733e-03 0.0235  0.0282 (21.23%)
combinational      8.213e-03    0.0374  0.0590  0.1046 (78.77%)
sequential          0.0000     0.0000  0.0000  0.0000 ( 0.00%)
    Net Switching Power = 0.0391 (29.46%)
    Cell Internal Power = 0.0112 ( 8.43%)
    Cell Leakage Power  = 0.0825 (62.12%)
-----
Total Power        = 0.1328 (100.00%)
```

To quantify the power reduction, compare the `report_power` results before and after the using `fix_eco_power` command.

Note:

Performing power recovery requires the following licenses:

- PrimeTime-ADV for the `fix_eco_power`, `report_cell_usage`, and `report_eco_library_cells` commands; for DMSA, one PrimeTime-ADV license is required per scenario
- PrimeTime-PX for the `report_power` command

For more information about PrimeTime ADV power recovery, see

- [Cell Swapping for Leakage Power Reduction](#)
- [Performing Power Recovery With DMSA](#)

Cell Swapping for Leakage Power Reduction

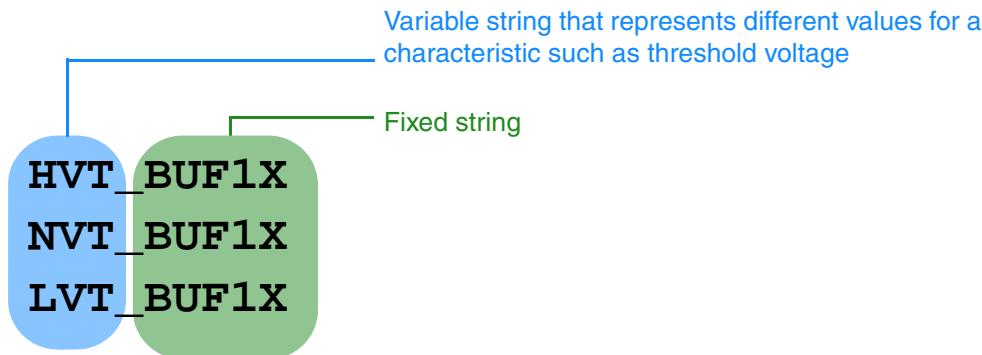
With the `fix_eco_power` command, you can reduce leakage power with the following operations:

- [Cell Swapping Based on the Library Cell Names](#) – When the library cells follow the recommended naming convention.
- [Cell Swapping Based on a User-Defined Attribute](#) – When the library cells do not follow the recommended naming convention.

Cell Swapping Based on the Library Cell Names

For swapping based on the library cell names, the cells must following a naming convention with a variable string and a fixed string, as shown in [Figure 19-5](#).

Figure 19-5 Recommended Library Cell Naming Convention for Cell Swapping



For example, the following buffer cell names indicate high, normal, and low threshold voltages:

Library	Library cells
HVT	HVT_BUF1X, HVT_BUF2X, HVT_BUF4X, HVT_BUF8X, HVT_DLY
NVT	NVT_BUF1X, NVT_BUF2X, NVT_BUF4X, NVT_BUF8X, NVT_DLY
LVT	LVT_BUF1X, LVT_BUF2X, LVT_BUF4X, LVT_BUF8X, LVT_DLY

To perform cell swapping based on these library cell names, use the `fix_eco_power` command with the `-pattern_priority` option. For example:

```
fix_eco_power -pattern_priority { HVT NVT LVT }
```

Cell Swapping Based on a User-Defined Attribute

If the library cell names do not follow the recommended naming convention, you can still perform cell swapping based on a user-defined attribute. For example:

Library cell	Value of user-defined attribute to specify threshold-voltage swap priority
INV1XH	INV1X_best
INV1XN	INV1X_ok
INV1X	INV1X_worst

To perform cell swapping based on a user-defined attribute, such as `vt_swap_priority`, use these commands:

```
define_user_attribute vt_swap_priority -type string -class lib_cell
set_user_attribute -class lib_cell lib/INV1XH \
    vt_swap_priority INV1X_best
set_user_attribute -class lib_cell lib/INV1XN \
    vt_swap_priority INV1X_ok
set_user_attribute -class lib_cell lib/INV1X \
    vt_swap_priority INV1X_worst
...
fix_eco_power -pattern_priority {best ok worst}
    -attribute vt_swap_priority
```

Performing Power Recovery With DMSA

You can run PrimeTime ADV power recovery with distributed multi-scenario analysis (DMSA). For a script example, see [Example 19-1](#).

Example 19-1 Script Example for Power Recovery With DMSA

```
remote_execute {
set timing_save_pin_arrival_and_slack true
update_timing -full
}
# PRE Reporting
report_global_timing
report_constraint -all -max_capacitance -max_transition
remote_execute {
    report_global_timing

    # Cell usage and power reporting done at slaves
    report_constraint -all -max_capacitance -max_transition
    set power_clock_network_include_register_clock_pin_power false
    set power_enable_analysis true
    report_power -groups {combinational sequential register}
    report_cell_usage; report_eco_library_cells
}

fix_eco_power -verbose

# POST Reporting
report_global_timing
report_constraint -all -max_capacitance -max_transition
remote_execute {
    report_global_timing
    report_constraint -all -max_capacitance -max_transition
    report_power -groups {combinational sequential register}
    report_cell_usage; report_eco_library_cells
    write_changes -format icctcl -output pt_eco.tcl
}
```

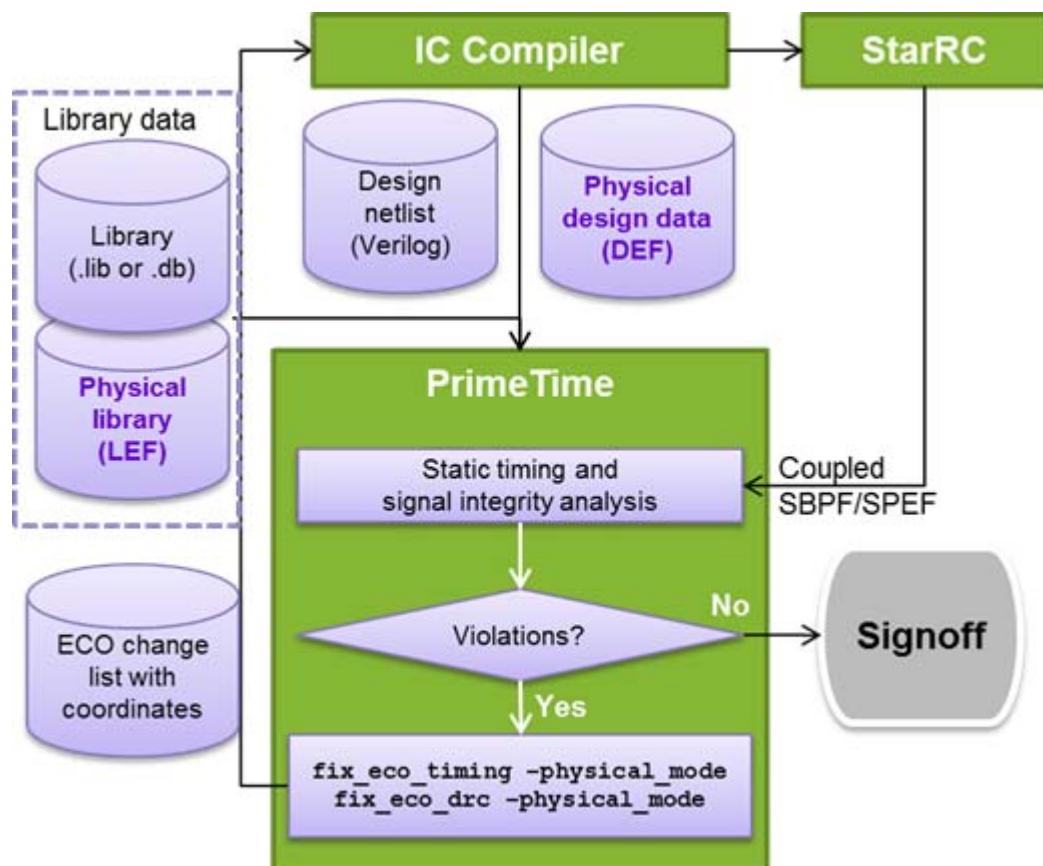
Physically-Aware ECO

The PrimeTime ADV tool can fix design rule and timing violations while considering physical placement and routing information, which provides the following benefits:

- Improves ECO fixing rates and predictability
- Adds placement locations to changed cells
- Avoids large displacements when resizing cells
- Considers available space, placement density, and wire delay when inserting buffers
- Performs on-route buffering when fixing violations
- Results in faster ECO convergence by minimizing the disturbance to the physical layout

PrimeTime ADV works well with IC Compiler and StarRC as part of the Synopsys Galaxy ECO flow ([Figure 19-6](#)).

Figure 19-6 Synopsys Galaxy Physically-Aware ECO Flow



Before running physically-aware ECO, you need the input data listed in [Table 19-3](#).

Table 19-3 Data Requirements for the Physically-Aware ECO Flow

Data required	Description
Verilog netlist, SDC, libraries	Design netlist, constraints, and timing information
DMSA setup	Setup for addressing violations across multiple scenarios
Standard Parasitic Exchange Format (SPEF) or Synopsys Binary Parasitic Format (SBPF) files	Parasitic data with location information; in StarRC, use NETLIST_NODE_SECTION: YES REDUCTION: NO_EXTRA_LOOPS
Block-level Library Exchange Format (LEF) files	Library technology and physical cell information; for more information, see the Library Data Preparation for IC Compiler User Guide
Hierarchical top- and block-level Design Exchange Format (DEF, version 5.7 or higher) files	Physical layout information of the design; for more information, see the “Writing the Floorplan to a DEF File” section in the IC Compiler Design Planning User Guide

To perform physically-aware ECO fixing:

1. Enable parasitics reading with location data by setting this variable:

```
pt_shell> set_app_var read_parasitics_load_locations true
```

2. (Optional) Enable the use of filler cells as open sites:

```
pt_shell> set_app_var eco_allow_filler_cells_as_open_sites true
```

3. Specify the physical library (LEF) and design data (DEF) input files and output log file. If the design has additional physical voltage area constraints or placement blockages, specify the `create_placement_blockage` and `create_voltage_area` commands in the physical constraint file:

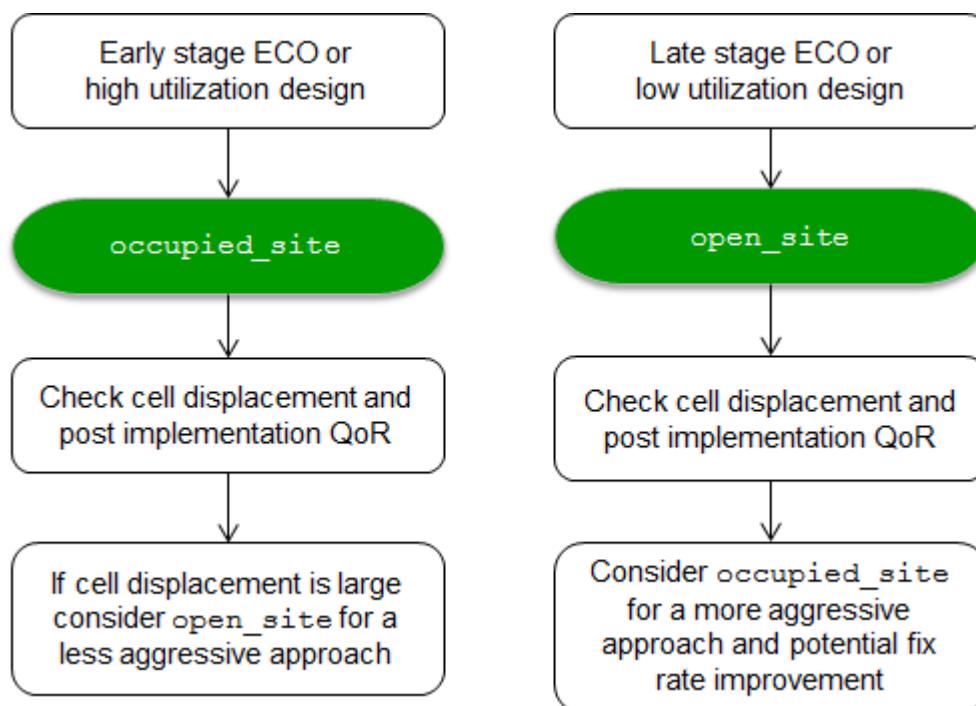
```
pt_shell> set_eco_options \
-physical_lib_path lef_files \
-physical_design_path def_files \
-physical_constraint_file physical_constraint_file \
-filler_cell_names cell_names \
-log_file my_log \
```

4. (Optional) Prevent the replacement or modification of specific cells, nets, designs, and library cells by using the `set_dont_touch` command, which applies the `dont_touch` attribute to the specified objects.
5. Run ECO fixing with a specified physical mode:

```
fix_eco_drc or fix_eco_timing
-physical_mode open_site | occupied_site
```

To choose the physical mode, consider the design characteristics and ECO objectives, as shown in [Figure 19-7](#).

Figure 19-7 Physically-Aware ECO Modes



6. Generate the block- and top-level ECO change list files with the `write_changes` command. (See [Writing Change Lists](#).) The resulting change list file includes location information, as shown in this example:

```
size_cell U1 AND2X
insert_buffer U2/Z BUF1X -location {212.3 753.2}
add_buffer_on_route net1 BUF2X -location {215.0 853.2} -no_legalize
...
```

7. Use the ECO change list files in the IC Compiler tool to implement the changes at the block and top levels.

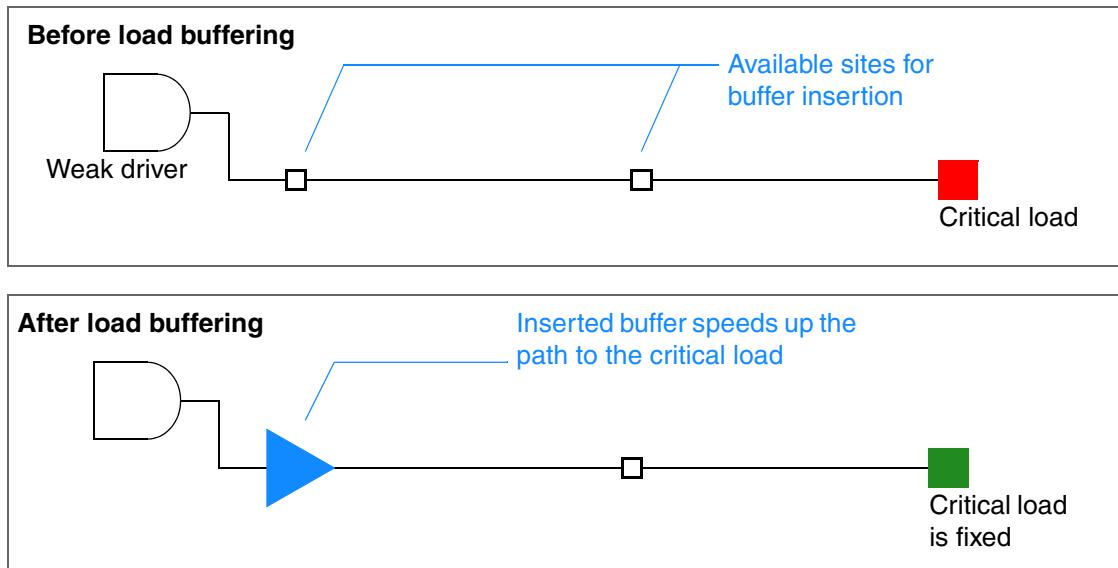
Physically-Aware ECO Setup Fixing Through Buffer Insertion

The PrimeTime ADV tool can fix setup violations by performing physically-aware load buffering and load shielding, resulting in more fixed violations:

- Load buffering – Inserting buffers to strengthen weak drivers.

[Figure 19-8](#) shows how load buffering automatically chooses the best available location for buffer insertion while considering placement blockages.

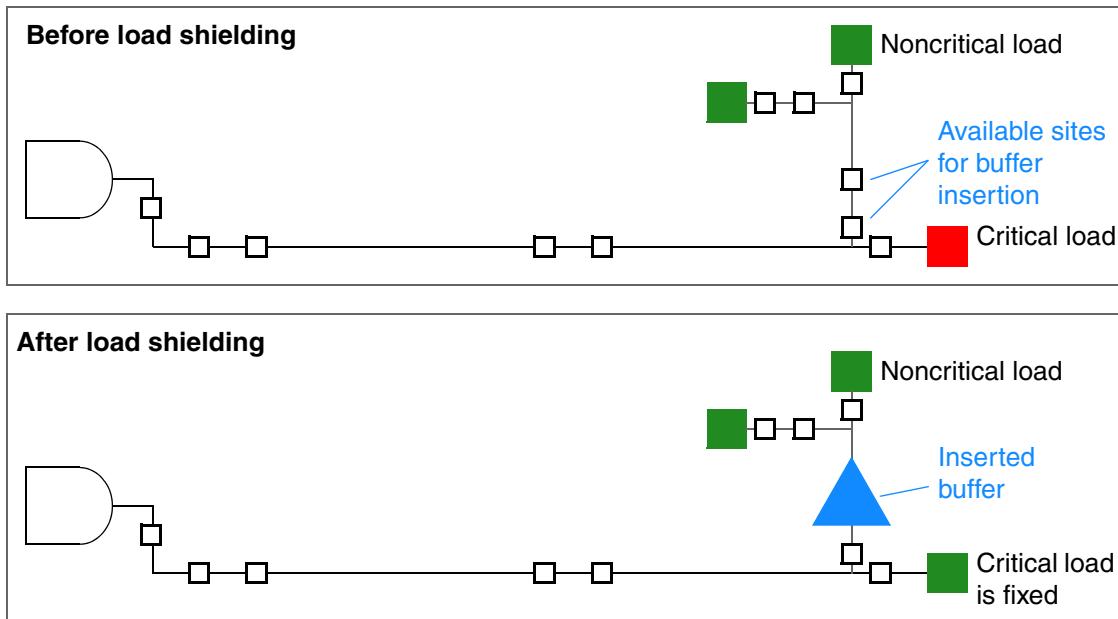
Figure 19-8 Example of Load Buffering



- Load shielding – Inserting buffers to shield critical loads from noncritical loads.

[Figure 19-9](#) shows how load shielding automatically chooses the best available location for buffer insertion while considering placement blockages.

Figure 19-9 Example of Load Shielding



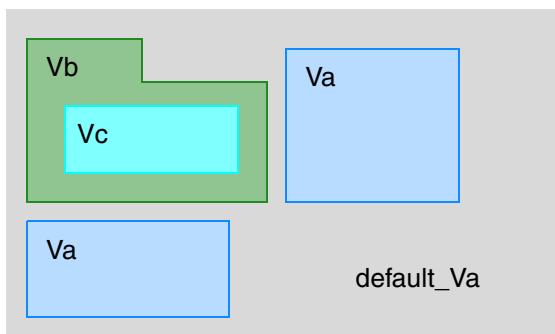
To fix setup violations with load buffering and shielding, use the `fix_eco_timing -type setup` command with the `-methods insert_buffer` option:

```
fix_eco_timing
  -type setup
  -physical_mode open_site | occupied_site
  -methods insert_buffer | size_cell
  -buffer_list list_of_buffers
```

The `insert_buffer` and `size_cell` options are mutually exclusive and cannot be used together in the same command. It is recommended to use `size_cell` first in the flow as the primary method for setup fixing. After using the `size_cell` method, use the `insert_buffer` method to gain incremental improvements in setup fixing.

Physically-Aware ECO for Multivoltage Designs

When providing ECO guidance, PrimeTime ADV can consider physical information in multivoltage designs, including designs with disjointed and nested voltage areas, as shown in [Figure 19-10](#).

Figure 19-10 Multivoltage Design Example

- default_Va is the default voltage area
- Va is a disjointed voltage area
- Vc is nested inside Vb

The tool aims to fix late-stage violations with minimal changes to the existing layout and without generating new electrical rule checking (ERC) violations. To achieve this, provide the tool with physical information by using the commands in [Table 19-4](#).

Table 19-4 Working With Physical Information for ECO

To do this task...	Use this command...
Create a voltage area	<code>create_voltage_area</code>
Create a placement blockage	<code>create_placement_blockage</code>
Load a file with voltage area information containing <code>create_voltage_area</code> and <code>create_placement_blockage</code> commands	<code>set_eco_options</code>

Note:

When you use the `create_voltage_area` command,

- The `-coordinate` and `-polygons` options are mutually exclusive
- The `-name` and `-power_domain` options are mutually exclusive

Manual Netlist Editing

You can resolve violations by manually editing the netlist and analyzing how the changes affect timing. This can be useful late in a design cycle when only small edits can be made.

To manually edit the netlist, use the commands in [Table 19-5](#).

Table 19-5 Netlist Editing Commands

Object	Task	Command
Buffer	Insert a buffer Remove a buffer	insert_buffer remove_buffer
Cell	Change the size (or drive strength) of a cell Create a cell Rename a cell Remove a cell	size_cell create_cell rename_cell remove_cell
Net	Add a new net Connect nets to pins Disconnect nets from pins Rename a net Remove a net	create_net connect_net disconnect_net rename_net remove_net

Note:

Do not perform substantial edits to the design for the purposes of adding new logic, removing out-of-date logic, or fixing widespread functional errors. Perform such significant design changes in the implementation tool.

As you edit the netlist, the tool records the changes in a change list file. See [Writing Change Lists](#).

Finding Alternative Library Cells

Before you replace or resize a cell, you need to know the alternative library cells that you can use. To list the library cell base names, use the `get_alternative_lib_cells -base_names` command. For example:

```
pt_shell> get_alternative_lib_cells -base_names u1
{"AN2", "AN2i"}
```

PrimeTime returns library cells in the order in which they are found according to the same search order that exists for the `size_cell`, `insert_buffer`, and `create_cell` commands. For more information about the order in which libraries are searched, see [Resolving Library Cells](#).

“What-If” Incremental Analysis

For a faster manual ECO fixing flow, you can perform “what-if” analysis of certain design changes entirely within PrimeTime. For analyzing these changes, PrimeTime uses a fast *incremental* analysis, which takes just a fraction of the time needed for a full analysis because it updates only the portion of the design affected by the changes.

To correct violations, you can increase the drive strength of victim nets by increasing the sizes of the driving cells with the `size_cell` command or by inserting buffers with the `insert_buffer` command. To fix crosstalk violations, you can also separate adjacent victim and aggressor nets with the `set_coupling_separation` command. You can also perform other custom netlist operations such as removing the cell and reconnecting a new cell with the `remove_cell`, `create_cell`, and `connect_net` commands.

During incremental analysis, you can use these commands:

- `size_cell`
- `insert_buffer`, `remove_buffer`
- `set_coupling_separation`, `remove_coupling_separation`
- `connect_net`, `disconnect_net`, `remove_net`
- `create_cell`, `remove_cell`

If you use other netlist editing commands, the `update_timing` command performs a full (not incremental) timing update.

For an incremental update, PrimeTime does the following:

1. Identifies the nets that are directly affected by the “what-if” changes, and their aggressors.
2. Performs electrical filtering of the affected nets.
3. Performs the first update iteration on the fanout cone of the affected nets, with the depth of the update cone limited to changes in slew.
4. Performs the second and any subsequent iterations on the nets in the affected cone.

After you decide on a set of changes, you can generate a change list file for IC Compiler.

The results of a “what-if” crosstalk analysis can be slightly different from a full analysis because of the iterative nature of the analysis and the interdependence of timing windows and crosstalk delay results. For final signoff of the design, perform a full analysis using SPEF or Verilog data from IC Compiler or another layout tool.

Automatic Uniquifying of Blocks

By editing a hierarchical block that is one instance of multiple instances in a design, you make that block unique. PrimeTime “uniquifies” that block for you automatically. For example, if the edited block is an instance of BLOCK, PrimeTime renames it BLOCK_0 or some similar name, avoiding any names already used.

Making an instance of a design in PrimeTime unique is somewhat different in comparison to other Synopsys tools because a new design is not created at the time the block becomes unique. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by using the `link_design -remove_sub_designs` command.

To list designs created by automatic uniquifying, use the `list_designs -all` command. These designs become real only if and when you link the design again.

When a block is edited, it is uniquified, along with all blocks above it, up to the first singly-instanced block. Messages are generated when uniquifying occurs. For example, if you use the `size_cell` command to change the size of cell i1/low/n1, it causes the cells to be uniquified:

```
pt_shell> size_cell i1/low/n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

Blocks are also marked as having been edited when netlist editing is performed. As soon as you edit a block, its parent and each level of the hierarchy up to the top-level has also been edited. This information is available from a Boolean attribute, `is_edited`, that is available on a design and on hierarchical cells. After you use the `size_cell` command, the `is_edited` attribute is on i1/low block. For example:

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

Resolving Library Cells

Many of the netlist editing commands pass a library cell as an argument, which can be a library cell object or the name of a library cell. Obtain the former by using the `get_lib_cells` command. The latter can be either the following:

- The full name of the library cell, such as, `class/AND2`, in which case there is no ambiguity about what to link the cell to, or
- The base name of the library cell, such as, `AND2`, in which case the library cell base name must be resolved to a library.

Library cell resolution for netlist editing commands is used primarily for netlist editing in distributed multi-scenario analysis, where you cannot use full names of library cells because each scenario might have a different set of libraries. However, you can also use this feature in single-scenario analysis. For more information, see [Netlist Editing in Multiple Scenarios](#).

Resolve library cell base names from the cell's current library or from a specific library that you define using the `-current_library` or `-library` option, respectively.

Use the `-current_library` option with the `size_cell`, `get_alternative_lib_cells`, and `report_alternative_lib_cells` commands. This option instructs PrimeTime to resolve the specified library cell base name from the existing cell's currently linked library only. This option prevents PrimeTime from searching any other libraries.

To specify a list of libraries to resolve a specified library cell name, use the `-libraries` option of the `size_cell`, `insert_buffer`, and `create_cell` commands. The tool searches the list of libraries in the order that you specify. The `-libraries` option prevents PrimeTime from searching any other libraries. This option is consistent with the `-libraries` option of the `get_alternative_lib_cells` and `report_alternative_lib_cells` commands.

In the absence of these options, PrimeTime resolves library cell base names from libraries in the following order:

- Link library of the current cell for the `size_cell`, `get_alternative_lib_cells`, or `report_alternative_lib_cells` command
- Libraries specified by the `link_path_per_instance` variable
- Libraries specified by the `link_path` variable

Use the `-base_names` option with the `get_alternative_lib_cells` command to return the alternative library cells as a list of library cell base names (a string) rather than a collection.

Estimating Delay Changes

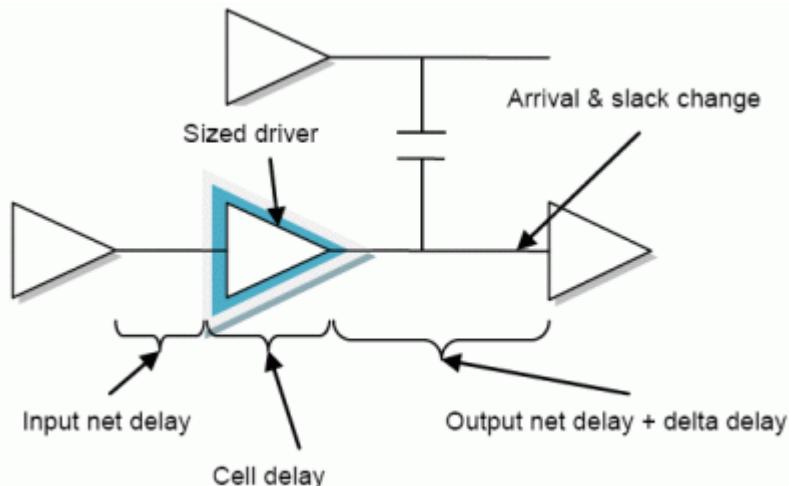
When performing ECO design modifications, fully understanding the consequences of the timing changes is important before you implement the changes in your design. Making the change and evaluating the resulting timing can cause you to incur the cost of an incremental timing update.

To improve the efficiency of the ECO flow, use the `estimate_eco` command. This command quickly computes and displays the ECO alternatives based on current timing values of a particular stage. The estimate takes into account the incoming transition times, output loading, fanout loading, detailed parasitics if present, and driver characteristics of the candidate cells.

The results are expected to reasonably predict, but not necessarily match, the actual timing resulting from subsequently performing that ECO command. Therefore, do not rely on the `estimate_eco` command to get an accurate timing report because the results are not guaranteed. Instead, use the command to help choose the best design change among multiple options by showing the relative timing effect of each option. In particular, this command allows you to write efficient ECO scripts that can quickly choose optimal ECOs without relying on an expensive change, update, check, or modify inner loop.

Two types of ECO fixing methods that are available with the `estimate_eco` command are sizing and buffer insertion. Both of these methods change the netlist and, as a result, change the delay values of the associated stages. For example, sizing a victim driver cell changes its crosstalk delay because of the driver resistance change. Moreover, it changes the previous stage delay and current stage delay due to input capacitance and driver resistance changes as shown in [Figure 19-11](#).

Figure 19-11 Timing Breakdowns



When using the `estimate_eco` command, use the following flow:

1. Show the available options.

In this stage, you evaluate possible ECO options. For example, you might want to size up a driver cell, buf3, in a path that has a setup timing violation. To find out the effects of substituting different drivers, use this command:

```
pt_shell> estimate_eco -type size_cell -max -rise

delay type : max_rise
lib cell      area   stage_delay   arrival    slack     trans
-----
mylib90/CKNXD12 19.05  0.0462      0.0462    0.1474   0.012
mylib90/INVD24  21.17  0.0265      0.0265    0.1671   0.030
mylib90/INVD20  18.35  0.0291      0.0291    0.1645   0.022
mylib90/CKNXD8  14.11  0.0610      0.0610    0.1326   0.105
mylib90/INVD3   3.53   0.1295      0.1295    0.0212   0.199
*mylib90/INVD2  2.82   0.1936      0.1936   -0.0013   0.292
mylib90/CKNXD1   2.82   0.4271      0.4271   -0.3214   0.326
```

Note:

This report shows all possible alternative library cells and associated timings with the asterisk (*) identifying the current cell. The rise or fall timing represents the worst timing through the cell output pins in the corresponding direction. The stage_delay column shows the delay of the inserted buffer or resized cell and the driven net. The arrival and slack columns represent the arrival time and slack at the load pins of the newly inserted buffer or resized cell. For more information about customizing the report output, see [Customizing Estimation Columns](#).

2. Estimate the stage delay improvements.

After deciding on the approaches for fixing the timing violation, you analyze the delay changes at the stage in which the netlist change is to occur. For example, to insert a buffer to the input pin, buf6/I, and estimate the stage delay change, use this command:

```
pt_shell> estimate_eco -type insert_buffer -inverter_pair \
           -max -rise -verbose \
           -lib_cells INVD

cell name : buf3
current lib cell: mylib90/INVD2
buffer lib cell : mylib90/INVD2

max_rise          current      estimate
-----
input net delay   0.102954   0.052354
stage delay       0.000240   0.000120
cell delay        0.371027   0.175534
output net delay  0.206952   0.110121
buffer delay      0.023532   0.013895
delta delay       0.140543   0.051518
arrival time      0.371027   0.175534
slack             0.304100   0.499593
transition
  cell input      0.202001   0.101001
```

cell output	0.252001	0.151001
load input	0.261001	0.161001

3. Commit changes and verify.

If you are sure about the fix, run the `size_cell` or `insert_buffer` command or both commands, and then run the `update_timing` command to see the actual changes. Since the `estimate_eco` command shows only estimated delays, based on current-stage information, the real delay values after the change can be different due to the interaction between previous, next, and coupled stages.

Note:

The timing update in this process cannot always guarantee real silicon delays because PrimeTime does not have place and route information.

For more information about customizing the report output, see [Customizing Estimation Columns](#).

Customizing Estimation Columns

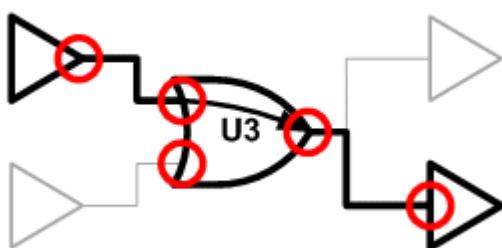
You can customize the format of the default compact output of the `estimate_eco` command. To specify the desired output columns for the `estimate_eco` command, use the `eco_estimation_output_columns` variable.

If you specify the `size_cell` estimation type in the `estimate_eco` command, the design rule checking estimations include

- All pins of the cell being resized
- Previous driver pin
- Slack-critical load pin

For example, [Figure 19-12](#) shows the input and output pins with the `estimate_eco -type size_cell U3` command.

Figure 19-12 estimate_eco -type size_cell Example

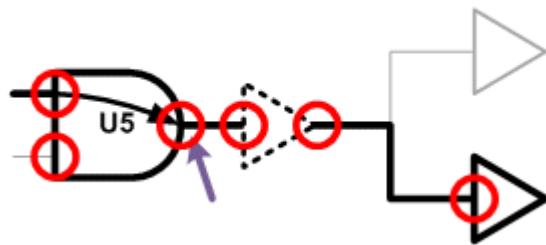


If you specify the `insert_buffer` estimation type in the `estimate_eco` command, the design rule checking estimations include

- All pins of the buffer being inserted
- All pins of the driver cell
- Slack-critical load pin

For example, [Figure 19-13](#) shows the input and output pins with the `estimate_eco -type insert_buffer U5/Z` command.

Figure 19-13 estimate_eco -type insert_buffer Example



To specify the information shown by the `estimate_eco` command, set the `eco_estimation_output_columns` variable.

The following example shows the output of the `estimate_eco -verbose` command.

Example 19-2 estimate_eco -verbose Report

```
pt_shell> estimate_eco U2319 -lib_cells OR2X1 -verbose \
           -significant_digits 4

cell name      : U2319
current lib cell: OR2X2

new lib cell: OR2X1
max_rise          current          estimate
-----
transition
  cell input        0.0941        0.0877
  cell output       0.0800        0.1154
  load input        0.0800        0.1154
max transition slack
  input net driver   4.4059        4.4123
  cell input         4.4059        4.4123
  cell output        4.4200        4.3846
  load input         4.4200        4.3846
```

```

new lib cell: OR2X1
max_fall           current      estimate
-----
transition
  cell input          0.1217      0.1170
  cell output         0.0821      0.1081
  load input          0.0821      0.1081
max transition slack
  input net driver    4.3783      4.3830
  cell input          4.3783      4.3830
  cell output         4.4179      4.3919
  load input          4.4179      4.3919

```

Sizing Cells

To replace the library cell referenced by a leaf cell in the netlist with a new cell, use the `size_cell` command. The result is a cell with new characteristics with respect to delay calculation, which results in different timing.

A library might contain several alternative library cells. For example:

```
pt_shell> get_alternative_lib_cells o_reg1
{ "class/FD2P1", "class/FD2P2", "class/FD2P3" }
```

If you do not know which cell to use for replacement, you can obtain the slack at the outputs of the leaf cell by using this command:

```
pt_shell> report_alternative_lib_cells o_reg1

*****
Report : alternative_lib_cells o_reg1 -delay_type max
...
*****
Alternative          Slack
Library Cells
-----
class/FD2P3          1.88(r)
class/FD2 *           -1.02(r)
class/FD2P1           -2.88(r)
class/FD2P2           -2.98(r)
```

The report indicates that the class/FD2P3 cell would produce a positive slack at the output of the leaf cell. Therefore, you would choose class/FD2P3 to size the leaf cell:

```
pt_shell> size_cell o_reg1 class/FD2P3
Sized 'o_reg1' with 'class/FD2P3'
1
```

Inserting and Removing Buffers

To add a buffer at one or more pins in the netlist, use the `insert_buffer` command. To remove a buffer from the netlist, use the `remove_buffer` command.

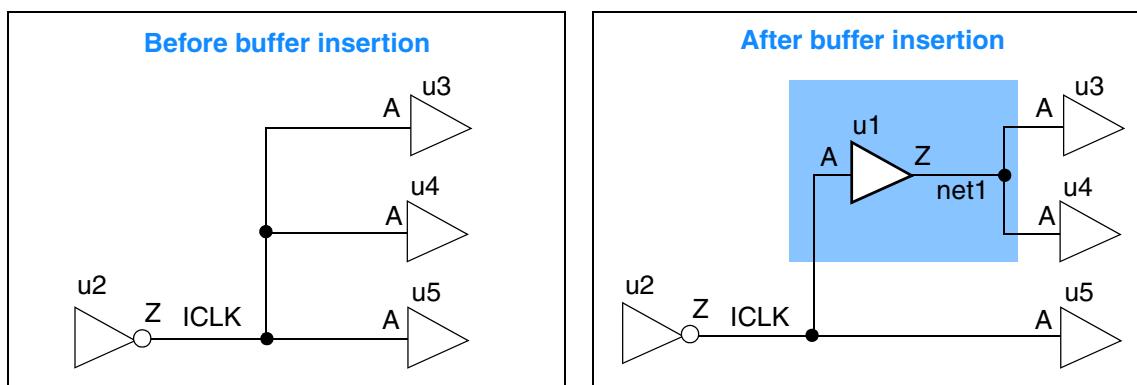
Note:

If you insert a buffer when parasitics are present, the parasitics might not be preserved.

[Figure 19-14](#) shows the effect of the following command:

```
pt_shell> insert_buffer {u3/A u4/A} class/B1I
```

Figure 19-14 Inserting Buffer



The `insert_buffer` command creates a new cell, `u1`, and a new net, `net1`. The input of the new buffer is the existing net, in this case, `ICLK`. The output of the new buffer is the new net, in this case, `net1`.

To change the prefix used to name the newly inserted buffers, you can use the `eco_instance_name_prefix` variable. By default, this variable is set to `U`.

Swapping Cells

To replace the cells in the specified cell list with a new design or library cell, use the `swap_cell` command.

Before performing the swap, the tool verifies that the pins of the replacement cell are equivalent to the pins of the original cell. For every pin of the cell you are swapping out, there must be a pin with the same name in the cell you are swapping in.

The `swap_cell` command always relinks the part of the design that has been replaced. Do not use the `link_design` command after you have used a `swap_cell` command; doing so often undoes the work of the `swap_cell` command. The PrimeTime data model is instance based. When you use the `swap_cell` command, an instance is modified. For example, `U2/U0` is an instance of design `x`, and you swap `U2/U0` for a different design. PrimeTime did not

modify the design `x`; it modified the instance `U2/U0`. Therefore, an initial link reverts to the old design.

PrimeTime does not save information about cells that were swapped; that information is maintained only until the next link. However, the change list for the design records the fact that the swap occurred, and you can export this information using the `write_changes` command.

By default, the `swap_cell` command restores the constraints that were on the design before the swap occurred. If you are doing multiple swaps, saving the constraints using the `write_script` command is far more efficient, and then use the `swap_cell` command with the `-dont_preserve_constraints` option.

When you have completed the swaps, restore the constraints by sourcing your script that you had written with the `write_script` command. If you are swapping one library cell for another, consider using the `size_cell` command, which is much more efficient. For example, to replace cells `u1`, `u2`, and `u3` in `TOP` with the `A` design (in `A.db`), enter

```
pt_shell> read_db A.db
pt_shell> current_design TOP
pt_shell> swap_cell {u1 u2 u3} A.db:A
```

To replace cells `u1`, `u2`, and `u3` in `TOP` with an LC3 lib_cell in the `misc_cmos` library, enter

```
pt_shell> swap_cell {u1 u2 u3} [get_lib_cells \
    misc_cmos/LC3]
```

Note:

The `swap_cell` command is intended to swap complex cells or hierarchical cells. When cells are swapped, a full timing update is incurred. Do not use the `swap_cell` command for simple cell resizing.

Renaming Cells or Nets

To change the name of a cell or net, use the `rename_cell` or `rename_net` command. When renaming a cell or a net using hierarchical names, make certain that the old and new names are at the same level of hierarchy. You can reference a cell or net by its full hierarchical name if it is below the current instance.

To successfully rename a cell or net, you must meet the following conditions:

- PrimeTime must be able to find the cell or net.
- No more than one cell or net matches the cell or net pattern that you specify.
- No leaf cell or net is found to match the new name that you specify.
- A new name must be at the same hierarchical level as the cell or net that you are renaming.

The following example successfully renames cellA to cellB in the current instance, middle.

```
pt_shell> rename_cell cellA cellB
Information: Renamed cell 'cellA' to 'cellB' in design
'middle'. (NED-008)
1
```

The following example renames a net at a level below the current instance. In this example, u1 and u1/low are instances of designs that have multiple instances; therefore, they are uniquified as part of the editing process.

```
pt_shell> rename_net [get_net u1/low/w1] u1/low/netA
Uniquifying 'u1/low' (low) as 'low_0'.
Uniquifying 'u1' (inter) as 'inter_0'.
Information: Renamed net 'w1' to 'netA' in 'middle/u1/low'.
(NED-009)
1
```

PrimeTime saves the name changes using the `rename_cell` and `rename_net` commands for output with the `write_changes` command.

User Function Class Support

For some complex library cells, the `user_function_class` attribute is required to describe the functional behavior of the cell instead of the `function_id` attribute. With support for the `user_function_class` library cell attribute, what-if analysis cell resizing is possible for these complex cells.

Although PrimeTime scales cells with the same `function_id` attribute, you can define their own function class. For example:

```
data buffers
clock buffers
delay buffers (used to fix hold times)
```

The foundry could provide these function classes to ensure that the proper cells are chosen for upsizing. You can set the `user_function_class` attribute using the `set_user_attribute` command, in Design Compiler, or in the source .lib file. For more information about setting this attribute, see the Design Compiler documentation.

Netlist Editing in Multiple Scenarios

When editing the netlist to fix violations, you want to verify that the changes do not negatively affect the timing results in all scenarios. To do this, use distributed multi-scenario analysis (DMSA) to evaluate the effects of different netlist edits and obtain merged reporting on the tested scenarios

With DMSA, you can run any number of netlist editing commands using a varying command focus. You can also execute netlist editing commands directly at the master; however, complex nested command structures are not supported in this mode of operation.

In DMSA, you can use all netlist editing commands listed in [Table 19-5](#). You cannot use the `swap_cell` command in the DMSA master, but you can execute it through the `remote_execute` command.

By using the `remote_execute` command, you can execute the following commands in a slave process and apply them to all scenarios in the command focus:

- `set_coupling_separation` – Creates a separation constraint on nets in all the scenarios in command focus.
- `remove_coupling_separation` – Removes the coupling separation from all scenarios in command focus.
- `read_parasitics -eco` – Reads StarRC ECO parasitic files to all scenarios in command focus.

Writing Change Lists

A change list describes all the changes made to the design during ECO and tells the place-and-route tool how to implement the changes. To generate a change list file, use the `write_changes` command.

To specify the format of the change list, use the `-format` option with one of the following arguments:

- `icctcl` – Tcl script for IC Compiler or Design Compiler.
- `icc2tcl` – Tcl script for IC Compiler II.
- `ptsh` – Tcl script for PrimeTime.
- `text` – ASCII text that can be parsed by external scripts for conversion into third-party formats.

Implementing the Changes With the Galaxy Design Platform

To implement the ECO changes with the tools in the Galaxy Design Platform, follow these steps:

1. In PrimeTime, generate the change list file for IC Compiler by executing the following command:

```
write_changes -format icctcl
```

2. In IC Compiler,
 - a. Read the change list file by using the `eco_netlist` command.
 - b. Execute the `place_eco_cells`, `legalize_placement`, and `route_zrt_eco` IC Compiler commands.
3. Run StarRC parasitic extraction.
4. After implementation, run PrimeTime to validate the QoR.
5. Repeat additional ECO fixing iterations through PrimeTime and layout, as necessary.

ECO Flow With StarRC Incremental Parasitic Extraction

After you make changes to the netlist, you generate the change list file and implement the changes with the IC Compiler place-and-route tool. Then you perform incremental parasitic extraction on the updated design with the StarRC extraction tool. To complete signoff, you need to rerun PrimeTime static timing analysis with the incremental extraction data.

After you make changes to the netlist, you generate the change list file and implement the changes with the IC Compiler place-and-route tool. Then you need to run parasitic extraction on the updated design with the StarRC extraction tool with one of the following flows:

- Full-design parasitic extraction – StarRC generates a new full-design parasitics file that matches the new netlist. You rerun PrimeTime analysis with the new netlist and parasitic files.
- Incremental parasitic extraction – StarRC generates an incremental parasitics file with changes due to buffer insertions, cell insertions, cell legalization, or net rerouting. This results in a reduced Standard Parasitic Exchange Format (SPEF) or Synopsys Binary Parasitic Format (SBPF) file. This flow improves productivity in large designs by retaining the netlist edits already performed in PrimeTime. You rerun timing analysis with the retained netlist edits and the incremental parasitic data.

To run the ECO flow with incremental parasitic extraction, follow these steps:

1. In PrimeTime, run timing analysis:

```
read_verilog ...
read_sdc ...
read_parasitics my_spef.spef -keep_capacitive_coupling
update_timing
```

2. Fix timing violations with automated ECO guidance commands (such as `fix_eco_timing` and `fix_eco_drc`) or manual netlist editing commands (such as `size_cell` or `insert_buffer`).

3. Generate a change list file for IC Compiler:

```
write_changes -format icctcl -output eco.tcl
```

4. Save the PrimeTime session:

```
save_session my_session
```

5. In the IC Compiler place-and-route tool, read the change list and implement the changes.

6. In the StarRC parasitic extraction tool, read the updated design and run incremental parasitic extraction.

7. In PrimeTime, restore the previous session and read the incremental parasitics:

```
restore_session my_session
read_parasitics -keep_capacitive_coupling original.SPEF \
-eco incremental.SPEF
```

8. Clear the netlist changes from the previous session and rerun timing analysis:

```
write_changes -reset
update_timing
report_timing
```

20

Timing Models for Hierarchical Analysis

PrimeTime provides hierarchical analysis capabilities by creating and using timing models to represent the timing characteristics of complex blocks in a design. A timing model of the block models the full input and output timing characteristics without the complete netlist of the block.

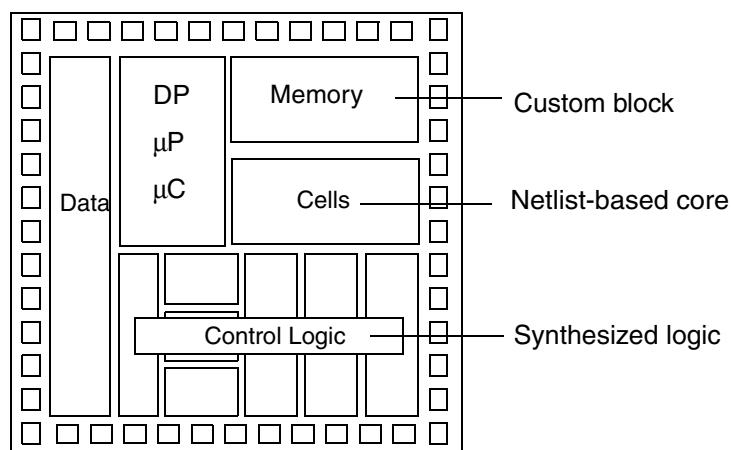
To learn about creating and using timing models for hierarchical timing analysis, see

- [Overview of Timing Models](#)
- [Synopsys Logic Libraries](#)
- [Quick Timing Models](#)
- [Interface Logic Models](#)
- [Extracted Timing Models](#)
- [Model Validation](#)
- [Hierarchical Scope Checking](#)
- [Context Characterization](#)

Overview of Timing Models

A typical chip can contain synthesized logic, netlist-based cores, and predesigned custom blocks, as shown in [Figure 20-1](#).

Figure 20-1 Components of a Typical Chip



Before you can perform static timing analysis on a chip using PrimeTime, every leaf cell must have a timing model. For static timing purposes, a leaf cell can be a simple macro cell (such as a NAND, NOR, or flip-flop) or a complex block (such as a RAM or microprocessor).

Synopsys provides timing model capabilities that are appropriate for different types of leaf cells:

- Synopsys logic libraries compiled by Library Compiler
- Interface timing specification models compiled by Library Compiler
- Generated interface logic models
- Extracted models created by PrimeTime
- Quick timing models created by PrimeTime

Synthesized logic is modeled as a netlist containing gates such as NANDs, NORs, and flip-flops. The gates are modeled using the standard Synopsys modeling language. They are then compiled into a logic library database (.db) file using Library Compiler. This logic library is the same library used by Design Compiler.

A netlist-based core is a predefined functional block designed at the gate level. For chip-level timing analysis, you can model a netlist-based core using the gate-level netlist. From the gate-level netlist you can generate a flattened Verilog netlist that contains the interface logic of a block, or you can extract a timing model.

A predesigned custom block is defined at the transistor level and imported into the chip as a fixed unit, such as a RAM or microprocessor block. Because a gate-level netlist does not exist for this type of block, the Liberty modeling language can be used to describe the timing behavior of the block.

Interface timing specification models can also be used to model custom blocks. However, the Liberty modeling language offers additional features, such as mode-dependent timing arcs, internally generated clocks, and internal pins.

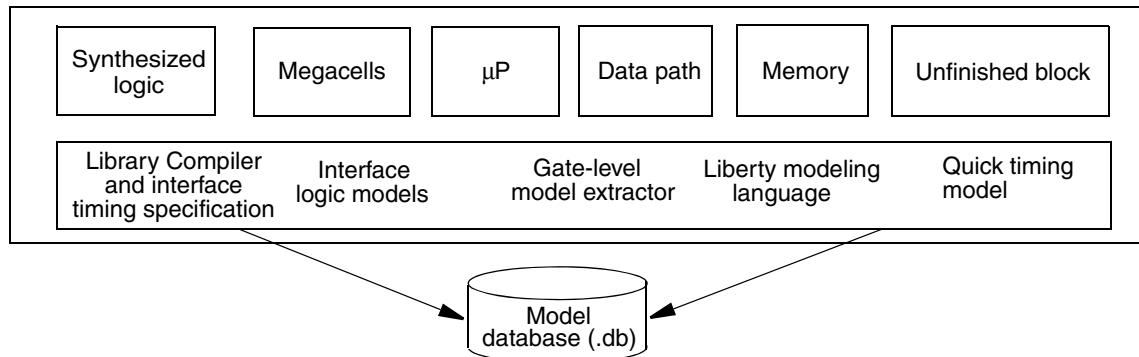
Table 20-1 lists the four methods of model generation that are used for the blocks in a typical chip.

Table 20-1 Model Generation for Blocks in Typical Chips

Block	Method of model generation
Synthesized logic	Gate-level netlist
Interface logic	Flattened netlist
Netlist-based core	Model extraction
Custom block	Liberty or interface timing specification

Some methods of model generation store model information in the database (.db) format, as shown in **Figure 20-2**.

Figure 20-2 Modeling Capabilities



Synopsys Logic Libraries

Synopsys logic libraries (or synthesis libraries) specify the timing and function of macro cells in an ASIC technology using Library Compiler. Design Compiler uses these libraries during the synthesis and timing steps of the design flow.

A logic library contains cell descriptions, which provide specific information about each component within an ASIC technology. Cell descriptions include

- Structures – Cell, bus, and pin structure that describes each cell's connection to the outside world.
- Function – The logical function of every output pin contained in the cell used by synthesis.
- Timing – Timing analysis and design optimization information, such as the parameters for pin-to-pin timing relationships and timing constraints for sequential cells.
- Other synthesis parameters – Parameters that describe area, power, and design rules (such as the maximum capacitance allowed for an output pin).

For more information about logic library cells, see the Library Compiler documentation.

Quick Timing Models

In the early stages of the design cycle, if a block does not yet have a netlist, you can use a quick timing model to describe its initial timing. Later in the cycle, you can replace each quick timing model with a netlist block to obtain more accurate timing.

To learn about creating and using quick timing models, see

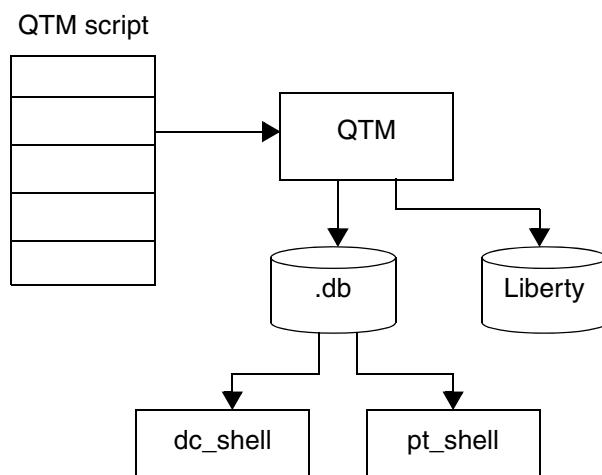
- [Quick Timing Model Design Flow](#)
- [Quick Timing Model Parameters](#)
- [Creating a Quick Timing Model](#)
- [Using a Quick Timing Model in a Design](#)
- [Quick Timing Model Command Summary](#)

Quick Timing Model Design Flow

To create a quick timing model, you use a series of PrimeTime commands to specify the model ports, the setup and hold constraints on the inputs, the clock-to-output path delays, and the input-to-output path delays. You can also specify the loads on input ports and the drive strength of output ports.

PrimeTime uses the information in the quick timing model to create a Synopsys library cell in .db format with the appropriate arcs. [Figure 20-3](#) shows the design flow for using quick timing models.

Figure 20-3 Design Flow for Using Quick Timing Models



You can save a quick timing model in the Synopsys .db format, then instantiate the quick timing model in a design just as you would instantiate library cells or interface timing specification models. Both Design Compiler and PrimeTime accept designs with instantiated quick timing models.

Quick Timing Model Parameters

To create quick timing models, use

- [Commands to Set Global Parameters](#)
- [Commands to Specify the Model Information](#)

Commands to Set Global Parameters

Before you specify model information, specify the quick timing model global parameters in [Table 20-2](#).

Table 20-2 Quick Timing Model Global Parameters

To set this QTM global parameter	Use this command	Command description
Logic library	<code>set_qtm_technology</code>	The <code>set_qtm_technology</code> command specifies the logic library to use for creating the quick timing model. Information includes the name of the logic library, the maximum transition time, the maximum capacitance, and the wire load model information.
Path type	<code>create_qtm_path_type</code>	The <code>create_qtm_path_type</code> command defines a path type. A library cell with an average fanout count for each level constitutes a path type. You can specify any number of path types. Use these types when you specify the timing arc in the quick timing model (for example, four levels of the NAND2 path type). For each level, the quick timing model computes the delay for each of those path types when the path is defined. You can also specify the timing arcs in terms of the actual numbers.
Path type	<code>create_qtm_path_type</code>	The <code>create_qtm_path_type</code> command defines a path type. A library cell with an average fanout count for each level constitutes a path type. You can specify any number of path types. Use these types when you specify the timing arc in the quick timing model (for example, four levels of the NAND2 path type). For each level, the quick timing model computes the delay for each of those path types when the path is defined. You can also specify the timing arcs in terms of the actual numbers.

Table 20-2 Quick Timing Model Global Parameters (Continued)

To set this QTM global parameter	Use this command	Command description
Flip-flop setup time, hold time, and clock-to-output delay	<code>set_qtm_global_parameter</code>	The <code>set_qtm_global_parameter</code> command specifies the global setup time, hold time, and clock-to-output delay for a flip-flop in the model. You specify these parameters globally because it is unlikely that you need to specify different times for different arcs. If you want to specify a different setup time, hold time, or clock-to-output delay, adjust the timing arcs you specify for the model. Specify timing arcs by providing a library element (such as a flip-flop) or by specifying a particular time value (such as 2 ns).
Load type	<code>create_qtm_load_type</code>	The <code>create_qtm_load_type</code> command defines load types that are used to specify the net capacitance of input ports. A library cell in the library constitutes a load type. You can label each of the library cell types and use this label when specifying the capacitance of the input ports (such as two NAND2 loads).
Drive type	<code>create_qtm_drive_type</code>	The <code>create_qtm_drive_type</code> command creates drive types. You can set the drive type for each output port. The drive type can be any library cell in the library (similar to the load type). This command enables you to specify an output drive of a port that might be different from the library cell you use in the path types.

Commands to Specify the Model Information

After you specify the global model parameters, specify the following model information:

Model port

The `create_qtm_port` command specifies model ports.

Input port capacitance

The `set_qtm_port_load` command specifies input port capacitance in terms of the load type defined in the global parameter section.

Output port drive

The `set_qtm_port_drive` command specifies output port drive in terms of the drive type global parameter.

Timing arcs

Use the `create_qtm_delay_arc` command to specify delay arcs between ports of the model (clock to output and input to output). Use the `create_qtm_constraint_arc` command to specify constraint arcs (setup and hold). To create a generated clock, use the `create_qtm_generated_clock` command.

You can specify constraint and delay arcs in terms of levels of path types specified in the global parameters. For an explanation of the calculation of the delay for each arc, see [Computing Arc Delays](#).

Computing Arc Delays

You can specify path delays and constraint values in terms of the delay arcs of any gate in the current logic library. For example, you can specify that the path delay of an input port to an output is equivalent to a path containing five NAND gates with an average fanout of three for each gate. PrimeTime computes the delay of this path and generates the corresponding arc.

Similarly, you can specify that the typical setup value is equal to the setup time of the D flip-flop in the library or that the output drive of the out1 port is equivalent to that of a particular buffer in the library.

PrimeTime computes the constraint and delay values for each timing arc from the arc specification. The path type and the number of levels of logic are stored for each timing arc. The global parameters contain setup, hold, and clock-to-output delays. The delay and level are computed from the path type.

Using the information that you provide, PrimeTime computes the delay for each arc as follows.

- For a constraint arc:

```
delay = (#levels) * (delay/level) + setup time
```

- For a delay arc (launch type from-clock-to-output-port):

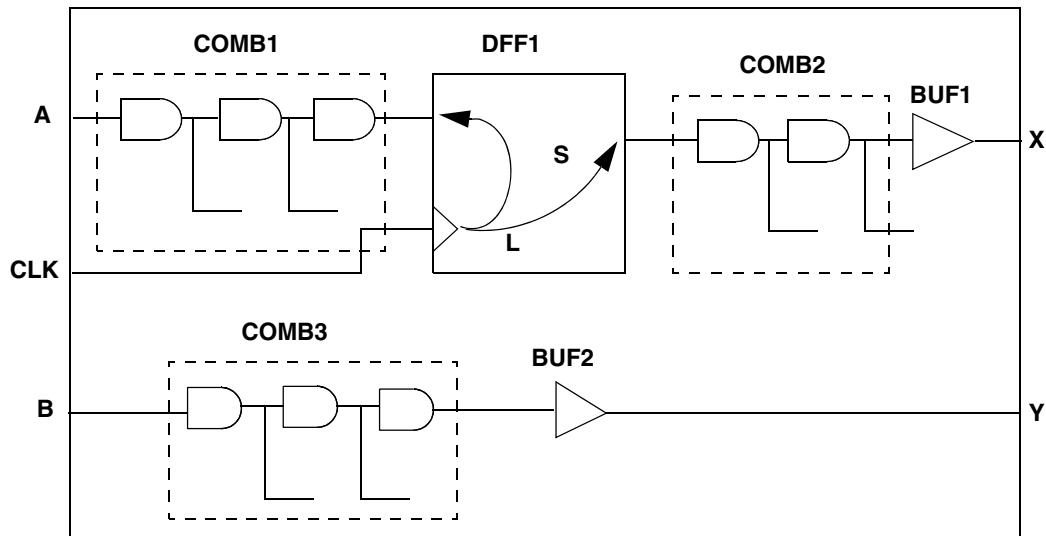
```
delay = (#levels) * (delay/level) + CLK-Q delay + output  
load-dependent delay
```

- For a delay arc (combinational arc):

```
delay = (#levels) * (delay/level) + output load-dependent  
delay
```

[Figure 20-4](#) shows the quick timing model for an undesigned block.

Figure 20-4 Quick Timing Model for an Undesigned Block



In [Figure 20-4](#),

- Port A is constrained by CLK using the setup requirement. This is modeled as a combinational path, COMB1, plus the setup to a flip-flop. The flip-flop setup time is the global setup time.
- The path from CLK to port X is modeled as the clock-to-output delay of a flip-flop plus the delays of combinational path COMB2 and driver BUF1.
- The path from input port B to output port Y is modeled as combinational path COMB3 plus the delay of the output driver BUF2.

To define this quick timing model:

1. Define a path type (path1) that is a 2-input AND with a fanout of two.
2. Define a drive type, named BUF1.
3. Define another drive type, named BUF2.
4. Define the global setup time, which is equivalent to the setup time of DFF1.
5. Define the global clock-to-output delay, which is equivalent to the launch time of DFF1.

The setup time of port A relative to CLK is

$$\text{delay of COMB1} + \text{setup time of DFF1 (denoted by arc 'S')} = \\ 3 * (\text{delay of path1}) + \text{global setup time}$$

The delay from DFF1 to output port X is

```
launch time of DFF1 (denoted by arc 'L') + delay of COMB2 +
load-dependent delay of BUF1 =
launch time of DFF1 + 2 * (delay of path1) + delay of BUF1
```

The delay from input port B to output port Y is

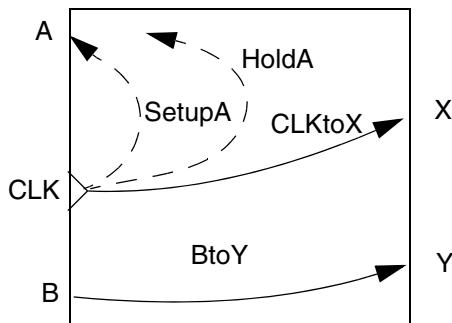
```
delay of COMB3 + load-dependent delay of BUF2 =
3 * delay of path1 + delay of BUF2
```

For input arcs, if the fanout of the input port is high, you must take into account the delay caused by buffering. If there is an input port with a high fanout, insert a chain of buffers before driving the library cells, or split the fanout among several buffers. You need to account for such delays by adding to the number of levels when specifying the input arcs.

Creating a Quick Timing Model

[Figure 20-5](#) shows a quick timing model representation of a block. Constraint arcs appear as dashed lines and delay arcs appear as solid lines. Port CLK is a clock port, ports A and B are input ports, and ports X and Y are output ports.

Figure 20-5 Quick Timing Model Representation of a Block



The arcs are

- SetupA – Constrains port A; the constraining port is clock CLK.
- HoldA – Constrains port A; the constraining port is clock CLK.
- CLKtoX – Delay arc from CLK to X.
- BtoY – Delay arc from B to Y.

To create, specify, and save the quick timing model in [Figure 20-5](#):

1. [Create the Model and Set Global Parameters](#).
2. [Specify Model Information](#).

3. [View the New Model.](#)
4. [Save the Model.](#)

Create the Model and Set Global Parameters

To create a model and set the global parameters:

1. Specify the name of the new model. For example:

```
pt_shell> create_qtm_model qtm_example
```

2. Specify the logic library to use with your design:

```
pt_shell> set_qtm_technology -library lib_new
```

3. Define a path type:

```
pt_shell> create_qtm_path_type path1 -lib_cell nand21 \
           -fanout 2
```

In this command, path1 is a 2-input NAND with a fanout of two.

4. Define a load type:

```
pt_shell> create_qtm_load_type load1 -lib_cell and2
```

PrimeTime selects the input port of a library cell to use at the input load.

5. Define the drive type:

```
pt_shell> create_qtm_drive_type drive1 -lib_cell buf1
pt_shell> create_qtm_drive_type drive2 -lib_cell buf2
```

6. Define a global setup time:

```
pt_shell> set_qtm_global_parameter -param setup \
           -lib_cell DFF1 -clock CLK -pin D
```

In this command, the setup time is equivalent to the setup time of the DFF1 library cell.

7. Define a hold time:

```
pt_shell> set_qtm_global_parameter -param hold \
           -value 0.0
```

8. Define a clock to the output delay time:

```
pt_shell> set_qtm_global_parameter -param clk_to_output \
           -lib_cell DFF1 -clock CLK -pin Q
```

In this command, the output delay time is equivalent to the launch time of the DFF1 library cell.

Specify Model Information

After you create a quick timing model and define its parameters, specify the various ports, attributes, and arcs for the model:

1. Create a clock port:

```
pt_shell> create_qtm_port {CLK} -type clock
```

2. Create the input ports:

```
pt_shell> create_qtm_port {A B} -type input
```

3. Create the output ports:

```
pt_shell> create_qtm_port {X Y} -type output
```

4. Set the load1 load type on the A and B ports:

```
pt_shell> set_qtm_port_load {A B} -type load1 -factor 2
```

5. Set a load of three capacitance units on CLK:

```
pt_shell> set_qtm_port_load {CLK} -value 3
```

6. Set a drive on the output ports:

```
pt_shell> set_qtm_port_drive X -type drive1  
pt_shell> set_qtm_port_drive Y -type drive2
```

7. Define the setup and hold arcs:

```
pt_shell> create_qtm_constraint_arc -setup -edge rise \  
-name SetupA -from CLK -to A -path_type path1 \  
-path_factor 2
```

```
pt_shell> create_qtm_constraint_arc -hold -edge rise \  
-name HoldA -from CLK -to A -path_type path 1 \  
-path_factor 2
```

8. Create the delay arcs:

```
pt_shell> create_qtm_delay_arc -name BtoY -from B \  
-to Y -path_type path1 -path_factor 3
```

```
pt_shell> create_qtm_delay_arc -name CLKtoX \  
-from CLK -to X -path_type path1 -path_factor 2
```

View the New Model

To report the defined global parameters, ports, and arcs in the quick timing model, use the `report_qtm_model` command:

```
pt_shell> report_qtm_model
```

Save the Model

To save the model in .db format or .lib format, use this command:

```
pt_shell> save_qtm_model -output file_name -format {db}
```

If you do not use the `-format` option, PrimeTime creates a model in .db format only.

You can choose to create either a library cell or wrapper and core by using or not using the `-library_cell` option of the `save_qtm_model` command. A library cell is easier to use but is not compatible with certain analysis flows. The `-format` option only controls the output format for a library cell; PrimeTime always writes a wrapper in .db format.

To save a quick timing model as a .db wrapper and core, use this command:

```
pt_shell> save_qtm_model -format db file_name
```

A command in this form writes two files: `file_name.lib.db` containing a library cell `model_name_core`, and `file_name.db` containing wrapper design `model_name` (where `model_name` is the name specified in the `create_qtm_model` command).

To save a quick timing model as a .db library cell, use this command:

```
pt_shell> save_qtm_model -format db file_name -library_cell
```

A command in this form writes one file: `file_name.lib.db` containing a library cell `model_name`.

Using a Quick Timing Model in a Design

To use a quick timing model in a design:

1. Instantiate the model in your design the same way you instantiate a leaf library cell.
2. Add the library file name to the `link_path` variable. Ensure that the path name of the directory containing this file appears in the `search_path` variable.
3. If your quick timing model has a wrapper, use the `read_db` command to read the wrapper file.
4. Link the design that uses the model.
5. Define the clocks and other timing assertions for the design containing the model.
6. Use the PrimeTime `report_timing` command to get reports on the timing of the design.

Quick Timing Model Command Summary

Table 20-3 summarizes the quick timing model commands that define, report, and save quick timing models.

Table 20-3 Quick Timing Model Commands Summary

Command	Task
create_qtm_constraint_arc	Creates a constraint arc
create_qtm_delay_arc	Creates a delay arc for a quick timing model
create_qtm_drive_type	Creates a drive type in a quick timing model description
create_qtm_generated_clock	Creates a generated clock for the model
create_qtm_load_type	Creates a load type for a quick timing model description
create_qtm_model	Begins defining a quick timing model
create_qtm_path_type	Creates a path type in a quick timing model
create_qtm_port	Creates a quick timing model port
report_qtm_model	Reports model data
save_qtm_model	Saves the quick timing model
select_qtm_port	Selects a quick timing model port
set_qtm_global_parameter	Sets a global parameter
set_qtm_port_drive	Sets drive on a port
set_qtm_port_load	Sets load on ports
set_qtm_technology	Sets various technology parameters

Interface Logic Models

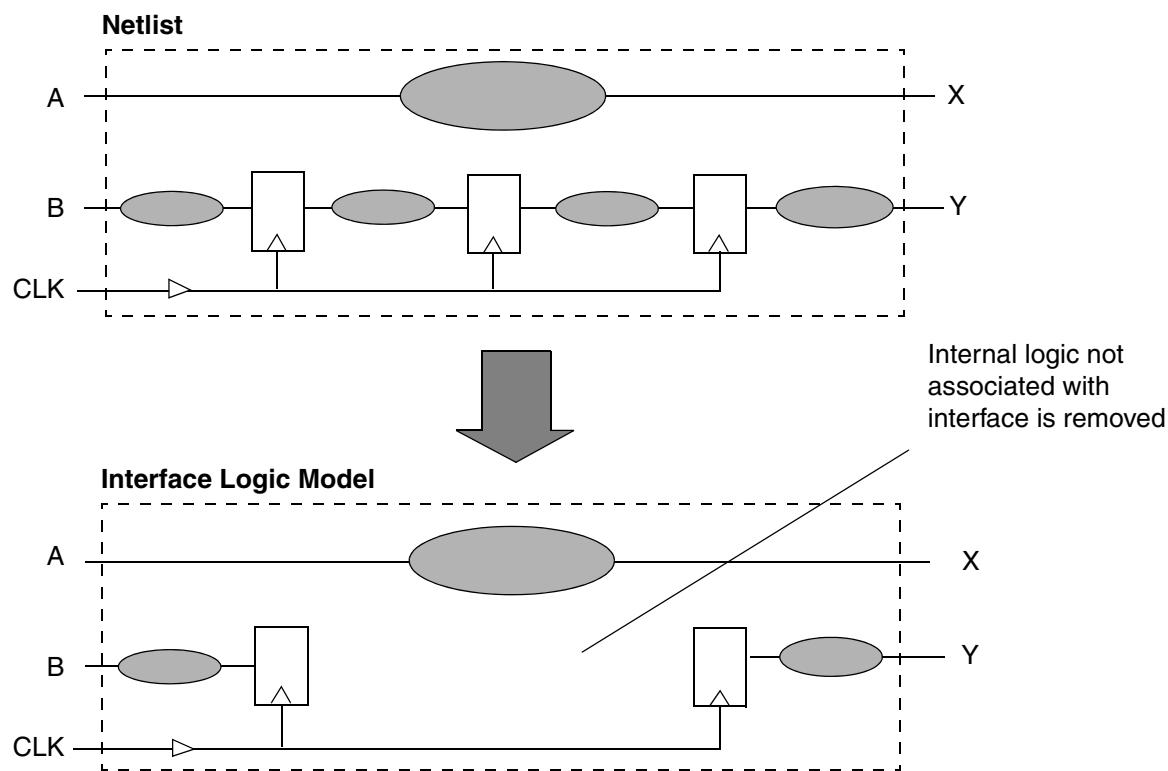
An *interface logic model* (ILM) is a partial netlist that contains only the interface logic of a block. The ILM contains

- The combinational logic from each input port to the first stage of sequential elements of the block

- The combinational logic from the last stage of sequential elements to each output port of the block
- The clock paths to these sequential elements
- Combinational paths from the input ports that do not encounter a sequential element and pass directly to an output port

[Figure 20-6](#) shows an ILM.

Figure 20-6 Generated Interface Logic Model



A generated ILM is context-independent, which means that the model is accurate for a range of operating environments. When the model is used in a design, the timing behavior of the model is different for different operating environments.

An ILM is a partial netlist that retains the combinational logic from each input or output port to the first or last stage of sequential elements of the block. The intent of an ILM is to produce a model that closely resembles the timing of the interface to that of the block-level netlist.

To learn about generating, validating, and using ILMs in a hierarchical static timing analysis flow, see

- [Benefits, Flow, and Limitations of ILMs](#)
 - [Generating, Validating, and Using an ILM](#)
 - [Shielded PrimeTime SI ILM Flow](#)
 - [Non-Shielded PrimeTime SI ILM Flow](#)
-

Benefits, Flow, and Limitations of ILMs

An ILM partial netlist can provide a smaller memory footprint, faster runtimes, and an extremely accurate timing of the interface logic of a block.

You can use ILMs in Standard Delay Format (SDF) or Standard Parasitic Exchange Format (SPEF) based flows. In addition, you can use ILMs in PrimeTime SI using both crosstalk and noise analysis.

To use the model in a hierarchical static timing analysis methodology, you must validate an ILM. This validation requires that the timing and scope match the block-level netlist usage within the design. A certain amount of time is needed to validate an ILM.

Generating, Validating, and Using an ILM

To generate, validate, and use an ILM in a hierarchical static timing analysis flow:

1. Properly constrain the block.

To constrain a block, you must define all the clocks and a conservative range of minimum and maximum values for bounding the input and output ports of the block used during the top-level static timing analysis.

These values should include bounding both the arrival times and transition times of each port of the block. This bounding is accomplished by using the following commands:

```
set_input_delay -min | -max  
set_output_delay -min | -max  
set_input_transition -min | -max  
set_clock_latency -dynamic
```

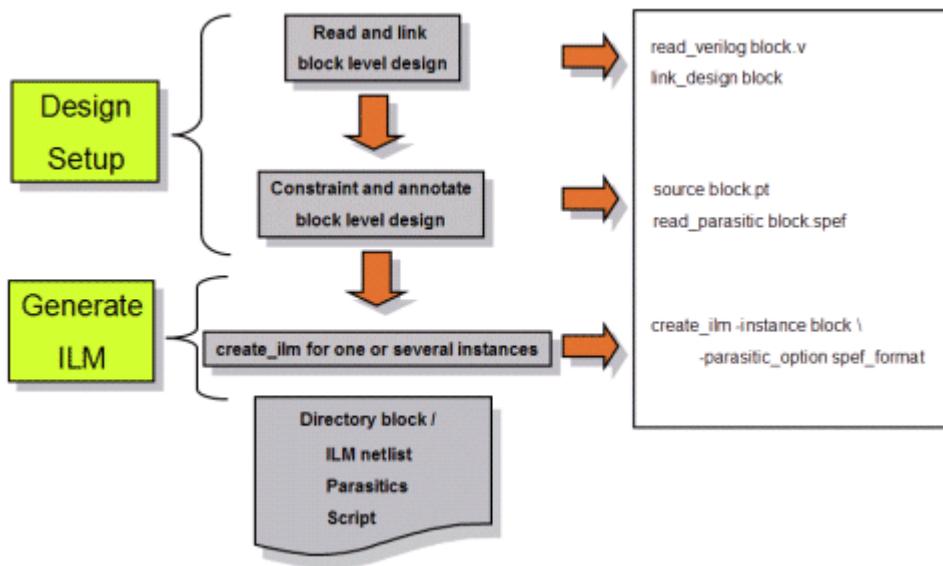
During scope checking of the block within the top-level static timing analysis, these values are verified to ensure that the block is operating within the context range used during extraction.

2. Generate an ILM. (See [Generating an ILM](#).)

3. Validate an ILM. (See [Validating an ILM](#).)
4. Use the ILM in a top-level netlist. (See [Using the ILM in a Top-Level Netlist](#).)

[Figure 20-7](#) outlines the steps used when generating an ILM.

Figure 20-7 ILM Usage Flow: Block Level



Generating an ILM

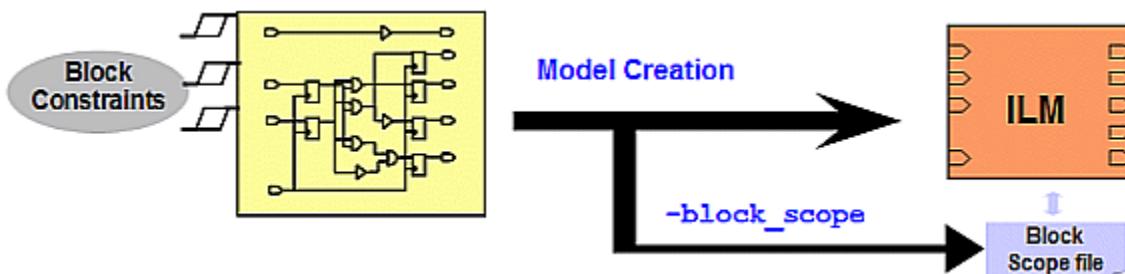
To generate an ILM, create an ILM for the current block-level netlist with the `create_ilm` command. By default, the command creates a model that includes all of the block interface logic and excludes all the internal register-to-register logic of the block.

You can use the `create_ilm` command with the following options:

- To produce a file that reflects the range of signal arrivals and slews entering and leaving the block, use the `-block_scope` option of the `create_ilm` command. Run the `check_block_scope` command during the top-level analysis to check to see if the range is honored.

[Figure 20-8](#) shows the process for checking the block level scope.

Figure 20-8 Checking the Block Level Scope



- To produce the files needed to verify the timing of the ILM, use the `-verification_script` option of the `create_ilm` command.
- To produce a SPEF file for the nets within the ILM, use the `-parasitics_options` option of the `create_ilm` command.

Example of Generating an ILM

In the following example, PrimeTime reads in a design called A.v and creates an ILM for that block:

```
pt_shell> set_app_var search_path ". ./db ./verilog ./spef ./libraries"
pt_shell> set_app_var link_path "* my_lib.db"
pt_shell> read_verilog A.v
pt_shell> link_design A
pt_shell> read_parasitics A.spef
pt_shell> source A.sdc; # script applies constraints
pt_shell> create_ilm -block_scope \
           -verification_script \
           -parasitics_options {spef input_port_nets constant_nets}
pt_shell> write_interface_timing net_tim.rep
```

This example produces the following files in the A subdirectory:

- A/ilm.v – Verilog netlist.
- A/ilm_inst.pt.gz – ILM constraint file.
- A/ilm.spef.gz – ILM SPEF file.
- A/ilm.scope – ILM block scope file.
- A/ilm_verif.pt.gz – ILM verification file.

Specific Setups with ILM

You can generate an ILM for a block-level netlist that has the following:

- High fanout ports (scan_enable, reset, and set)

When generating an ILM, each input port that is not defined as a clock is considered as a data port. When the tool generates an ILM, it traces each input port to the first level of the sequential element. For input ports, such as scan_enable, this means that each flip-flop in the block netlist is kept.

To address this, you can use either the `-auto_ignore` or `-ignore_ports` option. Both allow for special handling of these input ports. The `-auto_ignore` option allows the `create_ilm` command to check to see if an input port goes to more than a specific percent of the sequential elements in the block. The percentage is governed by the `ilm_ignore_percentage` variable with the default being 25%. You can also use the `-ignore_ports` options. This option allows for special handling of the `scan_enable` in this specific instance. For example,

```
pt_shell> create_ilm -ignore_ports [get_ports scan_enable]
```

When using the `-ignore` or `-auto_ignore` option, it can be useful to add the `-keep_ignored_fanout` option. In this case, the fanout from ignored input ports to interface logic is maintained in the model, making the ILM validation easier.

- Latches

For designs with latches, define the number of levels of borrowing for the latches at the interface. Set the value of the `-latch_level` option of the `create_ilm` command to match the number of levels of latch borrowing. For example:

```
pt_shell> create_ilm -latch_level transparent_latch_levels
```

Recommended Options for the `create_ilm` Command

In the SPEF-based flows, use the `-parasitics_options` option with the `create_ilm` command to generate parasitics for ILMs. For example:

```
pt_shell> create_ilm \  
      -parasitics_options {spef input_port_nets constant_nets}
```

To aid in validating the ILM, produce both the verification and block scope files by using the `-block_scope` and `-verification_script` options of the `create_ilm` command.

To add pins to the interface logic that otherwise would not be included, use the `-include_pins` option of the `create_ilm` command. For example, if you wanted to keep an internal clock-gating logic, you can add these pins for this internal clock-gating logic:

```
pt_shell> set pins [get_pins -of_objects [get_net w12]]  
pt_shell> create_ilm -include_pins $pins
```

In summary, in a SPEF-based flow, use the following command to create an ILM:

```
pt_shell> create_ilm \
      -parasitics_options {spef input_port_nets constant_nets} \
      -block_scope -verification_script
```

Validating an ILM

There are two checks that help ensure that the ILM block works within the static timing analysis hierarchical flow:

- [ILM Timing Matches Block-Level Netlist Timing](#)
- [Scope Checking for ILM Within Chip-Level Design](#)

ILM Timing Matches Block-Level Netlist Timing

The first check validates that the interface timing in the ILM matches the block-level netlist. In the flow, both the ILM and block-level netlist produce a file describing the timing at its interface. The `compare_interface_timing` command checks the two files and reports the results.

The scope check verifies that the block at the top-level falls within the ranges of I/O constraints that were set during block-level analysis.

The `create_ilm` command with the `-verification_script` option produces the `ilm_verif.pt.gz` file with the constraints for the block. You produce the interface timing for both the ILM and the full block netlist and then compare this timing.

To produce the interface timing for the ILM block, use the following syntax:

```
pt_shell> set_app_var search_path " . ./db ./verilog ./spef ./libraries"
pt_shell> set_app_var link_path " * my_lib.db"
pt_shell> read_verilog A/ilm.v
pt_shell> link_design block
pt_shell> source A/ilm_verif.pt.gz
pt_shell> read_parasitics A/ilm.spef.gz
pt_shell> update_timing -full
pt_shell> write_interface_timing ilm_tim.rep
```

The output of the timing report file is similar to the following:

```
*****
Command: write_interface_timing
          -significant_digits 6
Design : A
...
*****
Section: slack
Info   : Worst-case slack for each port and path group
Design : A
*****
```

Generated Clock and Source Info:

Attributes:

L<n> - latch level where <n> is 0 for first level

From	To	Arc Type	Worst-case Slack
EN(r)	clock1(f)	setup	3.826993
EN(f)	clock1(f)	setup	3.805498
EN(r)	clock1(f)	hold	5.903111
EN(f)	clock1(f)	hold	5.923214
ain[0](r)	clock1(r)	setup	13.671270
ain[0](f)	clock1(r)	setup	13.650400
ain[0](r)	clock1(r)	hold	-3.937757
ain[0](f)	clock1(r)	hold	-3.929484

For the full netlist, you now have the `net_tim.rep` timing file and for the ILM, you have the `ilm_tim.rep` timing file.

```
pt_shell> compare_interface_timing net_tim.rep ilm_tim.rep
```

Slight differences in the slack value can be due to slew propagation differences for unconnected pins within the ILM. Therefore, set a reasonable range for these types of differences, such as within 2 percent of the clock period.

The `compare_interface_timing` command compares the values in the two timing report files to each other. For example:

```
*****
Command: compare_interface_timing
          net_tim.rep A/ilm_tim.rep
          -session full
...
*****

```

From	To	Arc		Slack		Diff	Status
		Type	Ref	Cmp			
EN(r)	clock1(f)	setup	3.83	3.83	0.00	PASS	
EN(f)	clock1(f)	setup	3.81	3.81	0.00	PASS	
EN(r)	clock1(f)	hold	5.90	5.90	0.00	PASS	
EN(f)	clock1(f)	hold	5.92	5.92	0.00	PASS	
ain[0](r)	clock1(r)	setup	13.67	13.67	0.00	PASS	
ain[0](f)	clock1(r)	setup	13.65	13.65	0.00	PASS	
ain[0](r)	clock1(r)	hold	-3.94	-3.94	0.00	PASS	
ain[0](f)	clock1(r)	hold	-3.93	-3.93	0.00	PASS	
...		Totals	Slack	Transition Time	Capacitance	Rules	
Passed	1098	258		280	560	-	
Failed	0	0		0	0	0	
Total	1098	258		280	560	0	

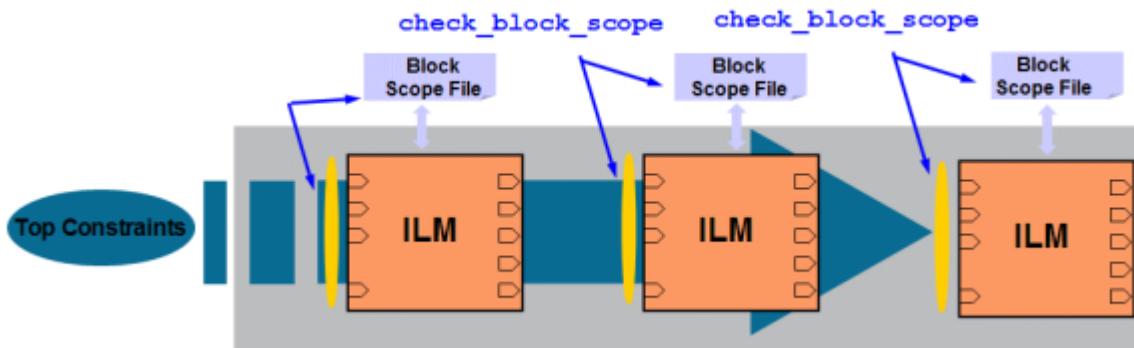
By validating that the slack, capacitance, and transition times match between the full netlist and the ILM, you are assured that the ILM matches the full netlist.

Scope Checking for ILM Within Chip-Level Design

Scope checking verifies that external timing conditions are within the ranges specified for block-level analysis. To ensure that the block context is valid for the original block-level timing analysis, you can check the block scope at the higher level.

While generating the ILM, the `-block_scope` option produces the block scope file. The `check_block_scope` command compares the actual arrivals and slews in the top-level static timing analysis to the values in the block scope file. [Figure 20-9](#) shows the process for checking the top level scope.

Figure 20-9 Checking the Top Level Scope



By default, the `check_block_scope` command checks the following:

- Clock consistency between block-level and top-level clock definitions, including existence, period, and mode, such as ideal or propagated
- Clock arrival times at block boundary pins
 - Scope contains absolute numbers
 - Command transforms these absolute numbers to relative numbers to check clock and edge relationships
 - Command determines a shift computed to minimize the number of scope violations and applies it to all the arrivals
- Clock transition at block boundary pins
- Interclock relationships at block boundary

From the `create_ilm -block_scope` command, the `A/ilm.scope` file is created.

To validate that the chip-level context is within the range of values from which the block was extracted, use these commands:

```
pt_shell> set_app_var search_path ". ./.db ./verilog ./spif ./libraries"
pt_shell> set_app_var link_path "* my_lib.db"
pt_shell> read_verilog chip.v A/ilm.v
pt_shell> link_design chip
pt_shell> read_parasitics -path inst1 A/ilm.spif

pt_shell> current_instance inst1
pt_shell> source A/ilm_inst.pt.gz
pt_shell> current_instance
pt_shell> source chip.pt
pt_shell> update_timing

pt_shell> check_block_scope -instances inst1 A/ilm.scope
```

To report the captured block scope data, use the `report_scope_data` command. The resulting block scope report is similar to the following:

```
*****
Report : scope_check
  -instance elvis
  -scope_scenario default
Design : top
...
*****
1. Checking analysis environment settings.
  No global derating violations.

2. Matching top-level and block-level clocks.
  (Information) Found top-level clock 'clock_bob' matching block-level
  clock 'clock1'.
  (Information) Found top-level clock 'clock2' matching block-level
  clock 'clock2'.

Clock mappings
  Top-level      Block-level      Ref_pin
  -----
  clock2          clock2          elvis/clk_buf2/A   (MET)
  clock_bob       clock1          elvis/clk_buf1/A   (MET)

3. Checking interclock relationships at block boundary.
  No violation to report or not enabled for checking.

4. Checking arrival and transition times at block boundary pins.
  No violation to report or not enabled for checking.
```

Verify that both the timing and scope checking is correct before proceeding to use the ILM block in the top-level chip level static timing analysis.

Model Validating Issues and Resolutions

For information about model validating issues, see

- [Modeling Issues](#)
- [Scope Checking Issues](#)

Modeling Issues

For ILMs, the `compare_interface_timing` command identifies interface paths that are not matching the block-level netlist.

One possible source of a slack issue can be due to slew propagation of unconnected input pins of the interface cell. To validate this difference, use the `report_timing` `-transition_time` command on both the block-level netlist compared with the ILM. The report shows the differences in the slew at the output pins of the cells. You can also see the

difference by using path-based analysis by using the `report_timing -pba_mode` command for both the block-level netlist and the ILM.

To add these additional unconnected pins to the ILM, use the `create_ilm -include_pins` command. This makes the new ILM larger.

Scope Checking Issues

Scope checking issues with clock definitions for the ILM are identified when you execute the `check_block_scope` command. Ensure that the clock definitions match from the block-level to the top-level netlist.

Another potential context issue has to do with signal latencies and transition times entering the block. The latencies or transition times entering the block from the top-level does not fall within the range defined while extracting the block.

There are two solutions to this issue. The first solution is to correct the top-level design to match the range set during the extraction. The second solution is to review the ranges set during the block-level analysis. You can then revise these signal ranges during block-level analysis before producing a new block scope file for the block.

Using the ILM in a Top-Level Netlist

After you have generated and verified the ILM, you can use it at the chip level in place of the block-level netlist for timing analysis.

To use the generated model,

1. Read in the netlist for each interface model. For example:

```
pt_shell> read_verilog A.v
```

2. Read and link the chip-level design:

```
pt_shell> read_verilog chip.v
pt_shell> link_design chip
```

3. For each ILM, read in the SDF or parasitics for the block. To specify the hierarchical path name leading to the instance, use the `-path` option with the `read_sdf` and `read_parasitics` commands. Enter one of the following:

```
pt_shell> read_parasitics -path inst1 A/ilm.spef.gz
```

or

```
pt_shell> read_sdf -path inst1 A/ilm.sdf
```

4. For each ILM, set the current instance to the block and source the script containing assertions and exceptions defined on the block:

```
pt_shell> current_instance inst1
```

```
pt_shell> source ilm_inst.pt.gz
```

5. Return to the top level netlist:

```
pt_shell> current_instance
```

6. Perform chip-level timing analysis:

```
pt_shell> check_timing -verbose
pt_shell> report_timing
```

7. Perform scope checking:

```
pt_shell> check_block_scope -instances inst1 A/ilm.scope
```

Shielded PrimeTime SI ILM Flow

This section describes the shielded PrimeTime SI ILM flow. A shielded block is where the designer has ensured that there are no wires routed either on top of or in close proximity to the block; therefore, crosstalk effects from the rest of the design into the block are not produced.

The methodology and flow for the PrimeTime flow is comparable to that of the PrimeTime SI ILM flow. The steps are as follows:

1. [Generating PrimeTime SI ILM](#)
2. [Model Validating for PrimeTime SI ILM](#)
3. [Using a PrimeTime SI ILM in a Top-Level Netlist](#)

Generating PrimeTime SI ILM

For generating a PrimeTime SI ILM, provide the transition times (minimum and maximum) for each input port with the `set_driving_cell` command. This reflects the cell driving the input port at the chip-level. For example:

```
pt_shell> set_driving_cell -min -max
```

Add the following to your block-level constraints to enable PrimeTime SI analysis:

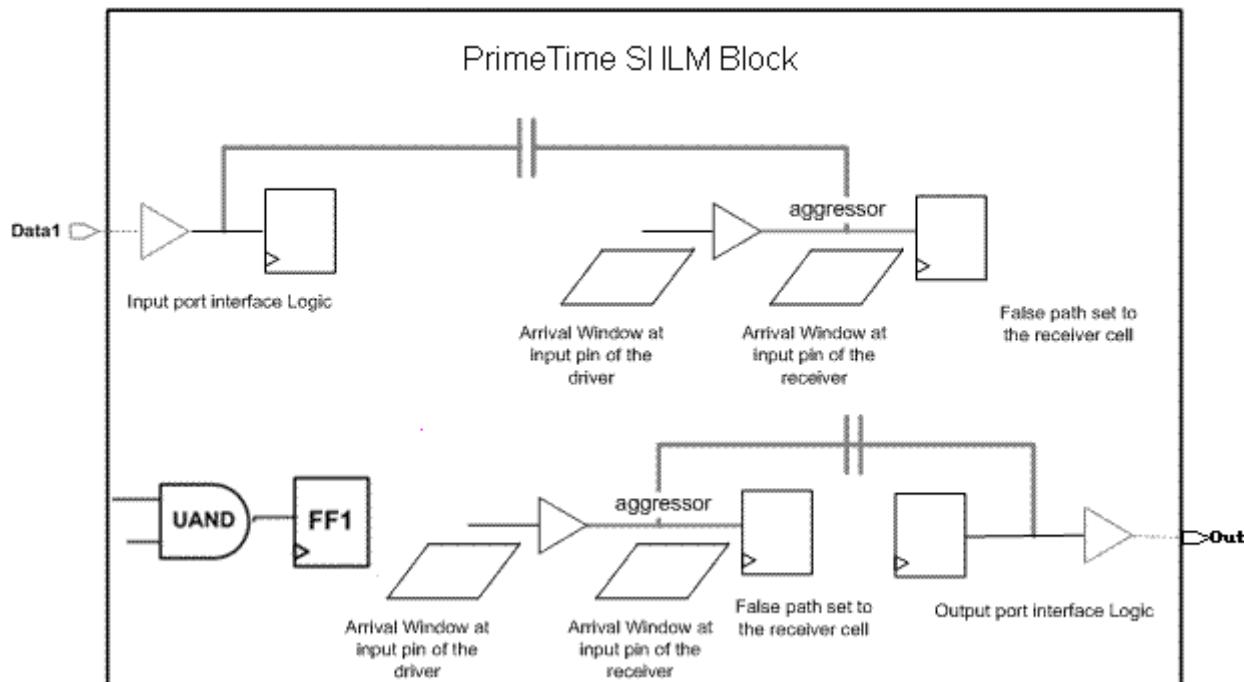
```
pt_shell> set_app_var si_enable_analysis true
pt_shell> read_parasitics -keep_capacitive_coupling
```

To include the PrimeTime SI delay analysis, use the `create_ilm -include si_delay_pins` command. To include the PrimeTime SI noise analysis, use the `create_ilm -include si_noise_pins` command. To include both, use this command:

```
pt_shell> create_ilm -include {si_delay_pins si_noise_pins}
```

These internal nets that are cross-coupled to the interface nets are retained in the PrimeTime SI ILM Verilog netlist. [Figure 20-10](#) shows the PrimeTime SI Block flow.

Figure 20-10 PrimeTime SI ILM Block



To generate an additional file with the arrival and transition times for the cross coupled internal nets, use this command:

```
pt_shell> write_arrival_annotations -design
```

Therefore, to produce a PrimeTime SI ILM, use these commands:

```
pt_shell> create_ilm -block_scope \
           -verification_script \
           -ignore_ports [get_port reset] \
           -include {si_delay_pins si_noise_pins} \
           -parasitics_options {spef input_port_nets constant_nets}
```

```
pt_shell> write_arrival_annotations -design
```

The subdirectory has the following additional files:

- A/ilm.v – Verilog Netlist
- A/ilm.pt.gz – PrimeTime SI ILM arrival annotations for internal nets
- A/ilm.txt – Contains list of added pins
- A/ilm_inst.pt.gz – PrimeTime SI ILM constraint file

- A/ilm.spef.gz – PrimeTime SI ILM specification file
- A/ilm.scope – PrimeTime SI ILM block scope file
- A/ilm_verif.pt.gz – PrimeTime SI ILM verification file

The `ilm.pt.gz` file contains commands similar to the following:

```
alias sat_r "set_annotation_transition -rise"
alias sat_f "set_annotation_transition -fall"
alias sad_r "set_input_delay -rise -add_delay"
alias sad_f "set_input_delay -fall -add_delay"

set pin [get_pin inst1/A1]
sad_r -max -reference_pin clkbuf1/A 2.18602 $pin
sad_f -max -reference_pin clkbuf1/A 2.21633 $pin
sad_r -min -reference_pin clkbuf1/A 1.62264 $pin
sad_f -min -reference_pin clkbuf1/A 1.64355 $pin
sat_r 0.0625615 -max $pin
sat_f 0.0503023 -max $pin
sat_r 0.055667 -min $pin
sat_f 0.0474351 -min $pin
```

The `ilm.pt.gz` file contains the following information:

- Arrival times and transition times for the nets that have been added to the model because they have a coupling effect on I/O ports.
- Transition times for the non-ILM side input pins for cells in the interface logic.

Model Validating for PrimeTime SI ILM

The model validation for a PrimeTime SI ILM uses the same steps as those that are explained in [Model Validating Issues and Resolutions](#).

The model validating described in [Modeling Issues](#) is also used for PrimeTime SI. However, the `ilm.pt.gz` command file must be applied together with the `ilm_verif.pt.gz` file. For comparison, the internal aggressors need to be applied in the same arrival window on input transition. Notice, that the parasitic files contain coupling capacitors; therefore, you must use the `-keep_capacitive_coupling` option with the `read_parasitics` command. You generate timing reports and use the `compare_interface_timing` command in the same way. For example:

```
pt_shell> set_app_var search_path ". ./db ./verilog ./spef ./libraries"
pt_shell> set_app_var link_path "* my_lib.db"
pt_shell> read_verilog A/ilm.v
pt_shell> link_design block
pt_shell> source A/ilm_verif.pt.gz
pt_shell> source A/ilm.pt.gz
pt_shell> read_parasitics -keep_capacitive_coupling A/ilm.spef.gz
pt_shell> update_timing -full
pt_shell> write_interface_timing ilm_tim.rep
```

Using a PrimeTime SI ILM in a Top-Level Netlist

The steps in the previous section for using an ILM at the top-level netlist are identical. You need to add only the `ilm.pt.gz` file for each instance in their top-level script.

For each ILM, set the current instance to the block and source the script containing assertions and exceptions defined on the block. If any exceptions (those defined relative to clocks on ports) need to be rewritten, do so before sourcing the script. Use the following syntax:

```
pt_shell> current_instance inst1
pt_shell> source A/ilm_inst.pt.gz
pt_shell> source A/ilm.pt.gz
pt_shell> current_instance
```

The scope checking issues described in [Scope Checking Issues](#) is also used for PrimeTime SI. Include the `data_input_arrival` and `data_input_transition` options of the `hier_scope_check_defaults` variable when scope checking. For example:

```
pt_shell> set_app_var hier_scope_check_defaults \
           clock_arrival clock_transition clock_skew_with_uncertainty \
           data_input_arrival data_input_transition
```

The `check_block_scope` command ensures that the data arrival and transitions of the inputs ports are within the conservative range set when the block was extracted. This ensures that a data arrival and transition change at the top level could not affect timing of the nets that are no longer in the ILM.

Non-Shielded PrimeTime SI ILM Flow

A *non-shielded block* is a block in which there is coupling from the top level down into the block level. By using the non-shielded PrimeTime SI ILM flow, you can perform hierarchical crosstalk analysis on non-shielded blocks.

To learn about the non-shielded PrimeTime SI ILM flow, see

- [Hierarchical Crosstalk Analysis](#)
- [Generating the Block Context](#)
- [Context Files](#)
- [Top-Level Design Files](#)
- [Blocks Below the Top Level](#)
- [Block-Level Analysis and Timing Model Generation](#)
- [Top-Level Analysis](#)

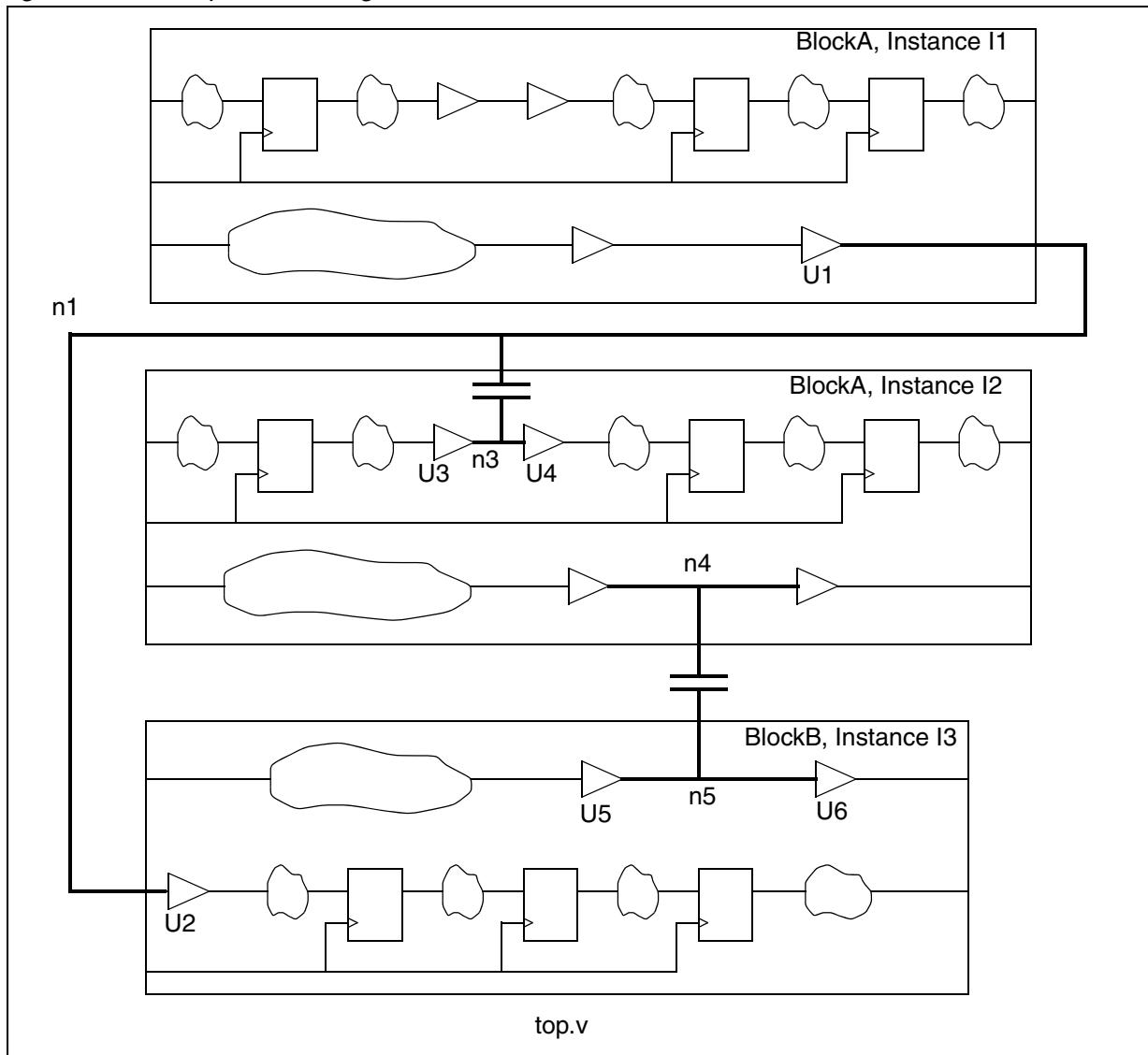
- [Analysis Iterations Using Annotated Arrival and Slew](#)
- [Using an ILM](#)

Hierarchical Crosstalk Analysis

You can do hierarchical crosstalk analysis with PrimeTime SI using timing models, taking into account any crosstalk between different hierarchical blocks or between a block and the top level. The analysis flow involves the creating the context or wrapper surrounding each block. The context includes aggressor nets and drivers that are outside the usual scope of the interface logic for the block. This information allows in-context crosstalk analysis at the block level between nets inside and outside of each block.

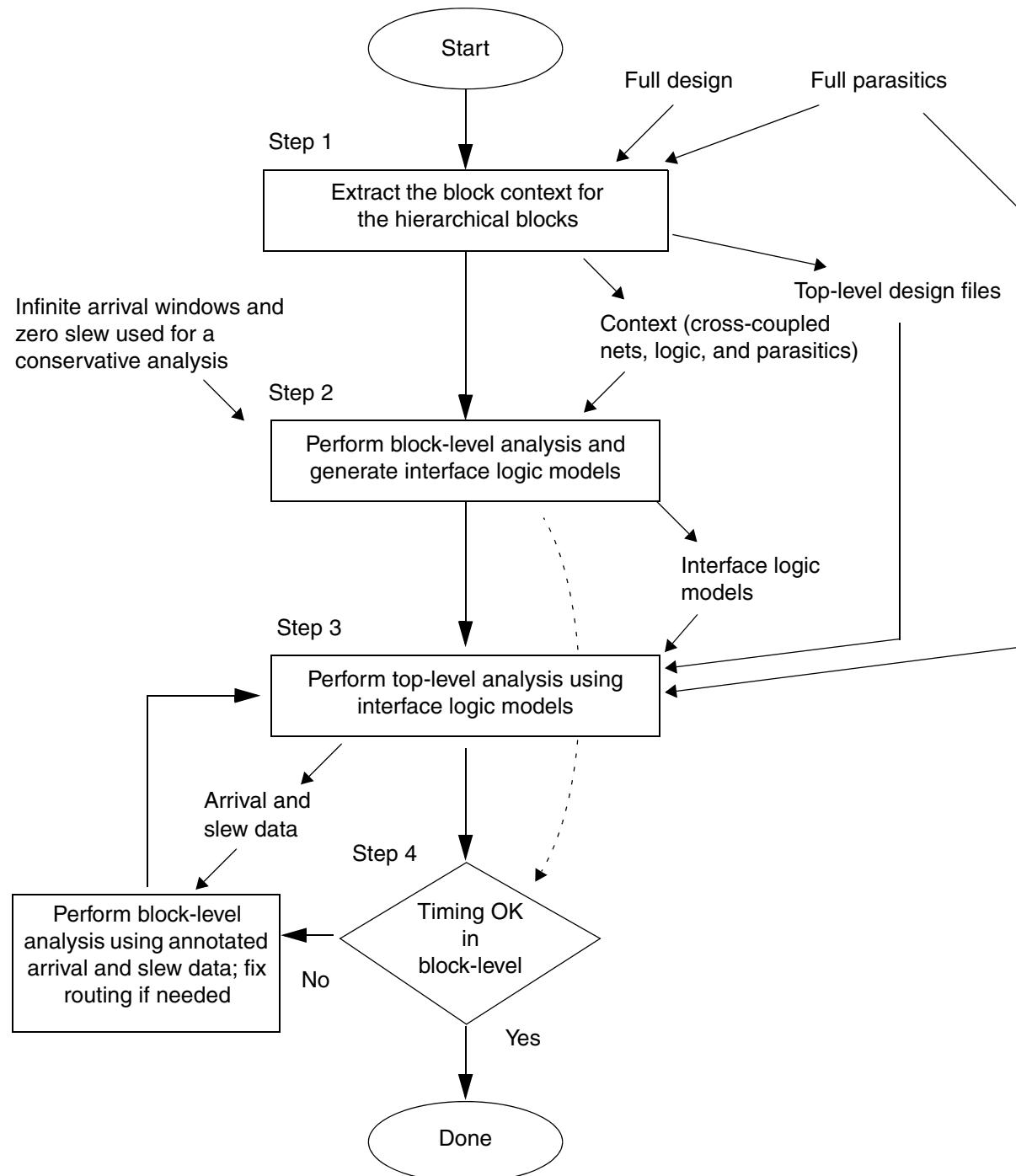
In [Figure 20-11](#), the I1, I2, and I3 blocks are modeled as interface logic models. There is crosstalk between an internal net of block I2 and the top level, and between the internal nets of blocks I2 and I3. Using ordinary interface logic models, these coupling effects would be lost. However, by generating the surrounding context for each block instance, the crosstalk effects can be maintained in the hierarchical analysis flow.

Figure 20-11 Top-Level Design With Crosstalk Between Blocks



[Figure 20-12](#) shows the flow to analyze crosstalk effects between different blocks and different levels of hierarchy.

Figure 20-12 Hierarchical Crosstalk Analysis Flow



The analysis flow consists of the following steps:

1. For each lower-level block, generate the block context and the parasitics for the block and its context. Generate a top-level design that uses interface logic models for the lower-level blocks. For more information, see [Generating the Block Context](#).
2. For each lower-level block, perform in-context timing analysis using the block, context, and block/context parasitics. Generate the interface logic models and related data files. For more information, see [Generating the Block Context](#).
3. Perform top-level analysis using the interface logic models and parasitic data annotated on the nets still remaining in the design. For more information, see [Top-Level Analysis](#).
4. Ensure that the design met all timing constraints in the previous block-level analysis. If not, it might be due to the conservative analysis used initially for block-level analysis. To find out the issue, generate new arrival/slew information for the block contexts, repeat in-context block-level analysis for each block, and go back to Step 3 to get a more accurate analysis. If necessary, change the routing or design parameters to meet the timing requirements.

This crosstalk analysis flow uses several different files to transfer data from one step to the next. By default, PrimeTime SI writes the files to the current working directory. You can specify a different directory by setting a variable called `pt_ilm_dir`. For example:

```
pt_shell> set_app_var pt_ilm_dir "/u/john/ilms"
pt_shell> set_app_var search_path "$pt_ilm_dir $search_path"
pt_shell> set_app_var sh_source_uses_search_path "true"
```

After executing these commands, the generated files go into the specified directory path and into automatically created subdirectories.

Generating the Block Context

The context of a block is the set of the nets and associated cells outside of a block that are involved in cross-coupling with nets inside the block. To perform in-context timing analysis of a block, you need to generate a file containing the block context. This information lets you analyze the cross-coupling effects of higher-level nets and nets from other blocks acting as aggressors to the internal nets of the block.

To generate the context parasitics, you can use either an external parasitic extraction tool, such as StarRC or PrimeTime. To use an external tool, follow the instructions provided with the tool for generating cross-coupling parasitics for each block. To use PrimeTime, you load in the full design, back-annotate the full parasitics, and then generate the context parasitics with the `create_si_context` command.

The top-level design in [Figure 20-11](#) is used as an example to describe the analysis flow. The design consists of three design files: `blockA.v`, `blockB.v`, and `top.v`. The parasitics for the full design are available in a file called `full_chip.spf`.

The three blocks at the top level are I1, I2, and I3. They are all to be modeled as interface logic models. Blocks I1 and I2 are instances of the same block, blockA . A top-level net starts at an output of block I1 and goes to an input of block I3, and has cross-coupling capacitance to net n3 in block I2. In addition, there is cross-coupling capacitance between net n4 in block I2 and net n5 in block I3.

The following script generates the context files for the three blocks:

```
read_verilog blockA.v
read_verilog blockB.v
read_verilog top.v
link_design
read_parasitics -keep_capacitive_coupling full_chip.spf
create_si_context -parasitics_options {sbpf_format}
```

The three `read_verilog` commands read in the full design. The `read_parasitics` command annotates the full-chip parasitics. The `create_si_context` command generates the context files and parasitics for the blocks listed in the command. Unless you specify a list of instances, the command generates context files and parasitics for all blocks at the top level.

To generate contexts for the blocks, it is not necessary to specify constraints (clocks, input delays, and so on) or to run a timing analysis. If you are unable to read in the full design parasitics due to computer memory constraints, you need to generate the context parasitics from StarRC or similar tool.

The `create_si_context` command generates several different files used in the hierarchical crosstalk analysis flow. It generates parasitic data files only if you use the `-parasitics_options` option of the command. You can specify `sbpf_format` or `spf_format` as the parasitic data format for the generated files. If you already have parasitics for the blocks instances from StarRC or other external tool, you can omit `-parasitics_options` from the command.

In addition to generating the parasitics for the three block contexts, the command also writes out the full-chip parasitics in Synopsys Binary Parasitic Format (SBPF), as this parasitic data is needed for the top-level analysis. If you already have the parasitic data in SBPF format, you can suppress generation of the data file by using the `-no_design_parasitics` option. The files generated by the `create_si_context` command can be divided into the two categories: context files and top-level design files.

Context Files

The `create_si_context` command creates the context for each block listed in the command. For the I1 block, it generates the following context files:

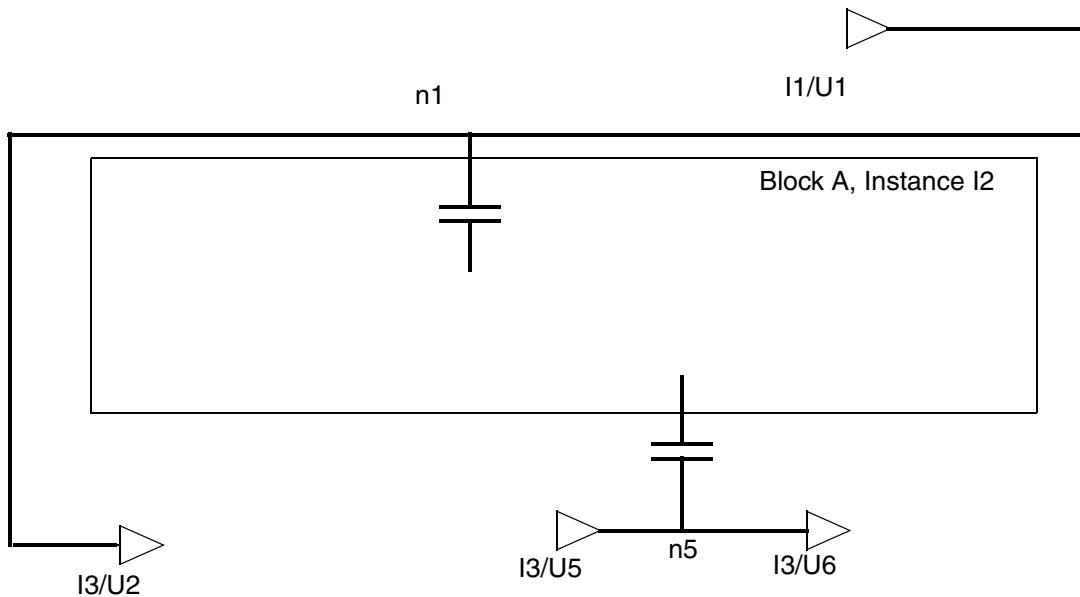
- I1/wrapper.v – Verilog design file for context
- I1/wrapper.sbpf – Block and context parasitics
- I1/wrapper.tcl – Tcl script for in-context, block-level analysis

- I1/wrapper.txt – List of aggressor annotation pins
- I1/wrapper.pt.gz – Arrival window annotation script

For the I2 and I3 blocks, it generates similar files and places them into directories named I2 and I3. Note that the context files for the I1 and I2 blocks are different, even though the two blocks are the same internally, because the cross-coupling capacitors to nets outside the blocks are not the same. The I1, I2, and I3 directories are created in the current working directory by default, or in the path specified by the `pt_ilm_dir` variable.

The wrapper.v file is a Verilog design file containing the block instance, the cross-coupled aggressor nets outside of the block, and the drivers and receivers of the aggressor nets outside of the blocks. [Figure 20-13](#) shows the design contained in the I2/wrapper.v file. The cross-coupling capacitors shown in light gray are later annotated on the design from the wrapper.sbpf file, along with the parasitics for the inside of the block.

Figure 20-13 Contents of Context File I2/wrapper.v



The wrapper.sbpf file is a parasitic data file for the block and its surrounding context. The generated file is in Synopsys Binary Parasitic Format (SBPF). This file is used for in-context, block-level analysis.

The wrapper.tcl file is a Tcl script used to help set up the in-context, block-level analysis. For the I2 block, the I2/wrapper.tcl file contains the following lines:

```
##### You must have read the block design by now
read_verilog I2/wrapper.v
current_design I2_wrapper
link_design
set_operating_conditions -analysis_type on_chip_variation
```

```
read_parasitics -keep_capacitive_coupling -format SBPF I2 wrapper.sbpf
##### Apply block level constraints now
```

The wrapper.txt file is a text file containing a list of the input pins of the drivers that are driving the aggressor nets in the context. This list can be used later in the flow to annotate arrival times and slew values on the context pins for a more accurate analysis. The I2 wrapper.txt file contains the following text:

```
I1/U1/A
I3/U5/A
```

The wrapper.pt.gz file is a gzip-compressed script file that sets infinite arrival windows on the crosstalk aggressor nets. Later in the flow, if you need to perform a more accurate (less pessimistic) analysis of a block using annotated arrival and slew times, you can overwrite this script file using the `write_arrival_annotations` command.

Top-Level Design Files

The `create_si_context` command generates the following files for the top-level design:

- `top.v` – Verilog design file with renamed block instances
- `top.sbpf` – Full-chip parasitics in SBPF binary format
- `top.tcl` – Tcl script for top-level analysis

The `top.v` file is the top-level design with the block instances renamed to avoid conflict between blocks, such as the I1 and I2 blocks (multiple instances of the same block with different cross-coupling to outside nets). The blocks are renamed by combining the block name and instance name, giving `blockA_I1`, `blockA_I2`, and `blockB_I3`. You can use this design later to generate different timing models for these blocks.

Note:

If you do not specify the destination directory with the `pt_ilm_dir` variable, the file is written to the current working directory, possibly overwriting an existing file with the same name.

The `top.sbpf` file contains the detailed parasitics for the whole chip in SBPF format. This file is needed for the top-level analysis. If you already have the full-chip parasitics in SBPF format, you can suppress generation of the file with the `-no_design_parasitics` option.

The `top.tcl` file is a Tcl script that helps set up the top-level analysis. For the design example, the file contains the following text:

```
read_verilog I1/iln.v
read_verilog I2/iln.v
read_verilog I3/iln.v
read_verilog top.v
current_design top
link_design
```

```

set_operating_conditions -analysis_type on_chip_variation
source I1/ilm_inst.pt.gz
source I1/ilm.pt.gz
source I2/ilm_inst.pt.gz
source I2/ilm.pt.gz
source I3/ilm_inst.pt.gz
source I3/ilm.pt.gz
read_parasitics -format SBPF top.sbpf -ilm_context \
    -keep_capacitive_coupling
##### INSERT CHIP LEVEL CONSTRAINTS NOW

```

For information about how to use this script, see [Top-Level Analysis](#).

Blocks Below the Top Level

The I1, I2, and I3 blocks are at the top level of the design. If the blocks to be modeled as timing models are not at the top level, you must use the `-top_inst` option of the `create_si_context` command to specify the name of the instance containing the blocks. The blocks must be at the same level of hierarchy and must be within the same higher-level block. For example, suppose that the design example was organized as shown in [Figure 20-14](#), with the timing-model blocks one level below the top level, inside a higher-level block called `core`. The command to extract the block context is as follows:

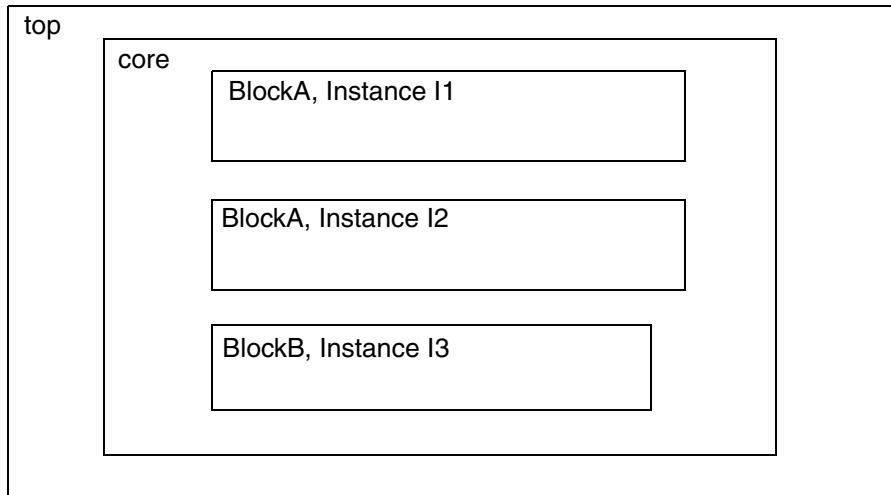
```

pt_shell> create_si_context \
    -instances {core/I1 core/I2 core/I3} \
    -top_inst core -parasitics_options {sbpf_format}

```

This generates a `core.v` file with the names of I1, I2, and I3 blocks renamed to `blockA_I1`, `blockA_I2`, and `blockB_I3` within the `core` block.

Figure 20-14 Blocks Below the Top Level



Block-Level Analysis and Timing Model Generation

After extracting the block context for each block, the next step is to perform block-level, in-context analysis of each block instance and to generate a context-accurate timing model for each block.

Block-Level Analysis

At this point of the flow, no arrival times or slew values are available for the context, so the analysis uses infinite arrival windows and zero transition times, resulting in a conservative analysis.

For each individual block instance, you can perform timing analysis with commands similar to the following:

```
pt_shell> read_verilog blockA.v
pt_shell> source I2/wrapper.tcl
pt_shell> source your_block_constraints.pt
pt_shell> update_timing
pt_shell> report_timing
pt_shell> ...
```

The `read_verilog` command reads in the original block design, not including the context. Sourcing the Tcl script surrounds the block with the context and applies detailed parasitics to the design, including the block and its context. Then you need to apply the timing constraints to the design such as clocks, input delays, and output delays. After you apply the constraints, you can run the analysis.

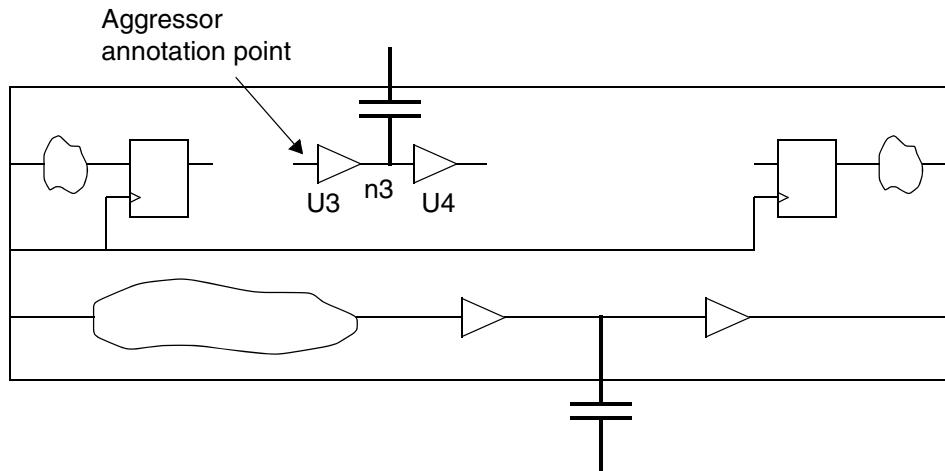
Generating ILMs for a Block Instance

To generate an ILM for a block instance, use the `create_ilm` command. For example:

```
pt_shell> create_ilm -instances {I2} -include {si_delay_pins}
```

This command extracts the timing model from the I2 instance and puts the model files in the current directory by default or in the path specified by the `pt_ilm_dir` variable. Because the `-include {si_delay_pins}` option is used, the command also identifies the interface logic of the block and adds any pins needed for crosstalk analysis. The command generates the Verilog description of the timing model design as shown in [Figure 20-15](#).

Figure 20-15 Interface Logic Model Created for Block I2



The U3 and U4 cells and n3 net are included in the interface logic because they are involved in cross-coupling with an aggressor net outside of the block. The generated timing models can be different for different instances of the same block, such as the I1 and I2 blocks because of different cross-coupling or different options used in the `create_ilm` command.

The `create_ilm` command also generates a PrimeTime script to help set up usage of the model and a list of aggressor annotation points. These are the files generated by the command:

- I2/ilm.v – Verilog design file for the interface logic model
- I2/ilm_inst.pt.gz – Script to apply the instance constraints
- I2/ilm.txt – List of aggressor annotation points

Optionally, you can generate a verification script, parasitic data, or SDF data by using the appropriate options in the `create_ilm` command. You can generate the following additional files:

- I2/ilm_verif_pt.gz – Model verification script
- I2/ilm_sbpf or I2/ilm_specf – Parasitic data
- I2/ilm.sdf – Delay data in SDF format

The `create_ilm` command can perform all of the functions of the following ILM commands:

```
identify_interface_logic
write_ilm_netlist
write_ilm_script
write_ilm_parasitics
write_ilm_sdf
```

It can also perform functions that are not available in the ILM commands, such as generating a timing model for just one instance or including specific parts of the netlist that are not part of the interface logic.

Annotating Arrival and Slew Information

The list of aggressor annotation points in the I2/ilm.txt file (generated by the `create_ilm` command) can be used to annotate arrival times and slew values at the timing model pins for a more accurate top-level analysis. To annotate this information, you can use the `write_arrival_annotations` command. For example:

```
pt_shell> write_arrival_annotations -instances {I2}
```

For the I2 block, the I2/ilm.txt file contains the name of one pin, the U3/A pin. The `write_arrival_annotations` command writes out the arrival and transition time information as a script file named I2/ilm.pt.gz, which can be used for top-level analysis. The annotations are written with respect to a leaf-level pin in the clock network of the arrival window clock. For example, if leaf_clk/A is the first leaf-level pin found that is connected to the block-level clock source named clk, the `write_arrival_annotations` command creates a script similar to the following:

```
set pin [get_pin u3/A]
set_input_delay -rise -max -reference_pin leaf_clk/A 23.1968 $pin
set_input_delay -fall -max -reference_pin leaf_clk/A 23.1464 $pin
set_input_delay -rise -min -reference_pin leaf_clk/A 14.1356 $pin
set_input_delay -fall -min -reference_pin leaf_clk/A 14.0872 $pin
set_annotated_transition -rise 0.194342 -max $pin
set_annotated_transition -fall 0.137126 -max $pin
set_annotated_transition -rise 0.194429 -min $pin
set_annotated_transition -fall 0.137327 -min $pin
```

At the top level, the actual clock source latency that reaches the pin leaf_clk/A is automatically applied to the arrival window of the u3/A pin.

If multiple clocks are defined at same source or multiple clock sources share the same net, the first level leaf pin has multiple clocks reaching it. In that case, the annotations are written with respect to a virtual clock created by the script. The virtual clock has the same waveform as the block-level clock. You might want to modify the source latency of this virtual clock to fit the conditions of the top-level analysis.

Validating an Interface Logic Model

You can validate a generated ILM against the original netlist for the block with the `write_interface_timing` and `compare_interface_timing` commands. To verify an ILM, use a simple wire load model rather than detailed parasitics because parasitics are not generated in the hierarchical crosstalk analysis flow. When you use the `create_ilm` command, use the `-verification_script` option to generate a script for model validation.

The following flow is recommended for validating ILMs:

1. Read in and link the block-level design.
2. Source the script that applies the block constraints.
3. Write the interface timing report:

```
pt_shell> write_interface_timing net.rpt
```

4. Remove the design or open another pt_shell.
5. Read in and link the interface logic model.
6. Source the script that applies the model verification constraints.
7. Write the interface timing report:

```
pt_shell> write_interface_timing ilm.rpt
```

8. Generate the comparison report:

```
pt_shell> compare_interface_timing net.rpt ilm.rpt tolerances
```

For more information about timing model validation, see [Model Validation](#).

Top-Level Analysis

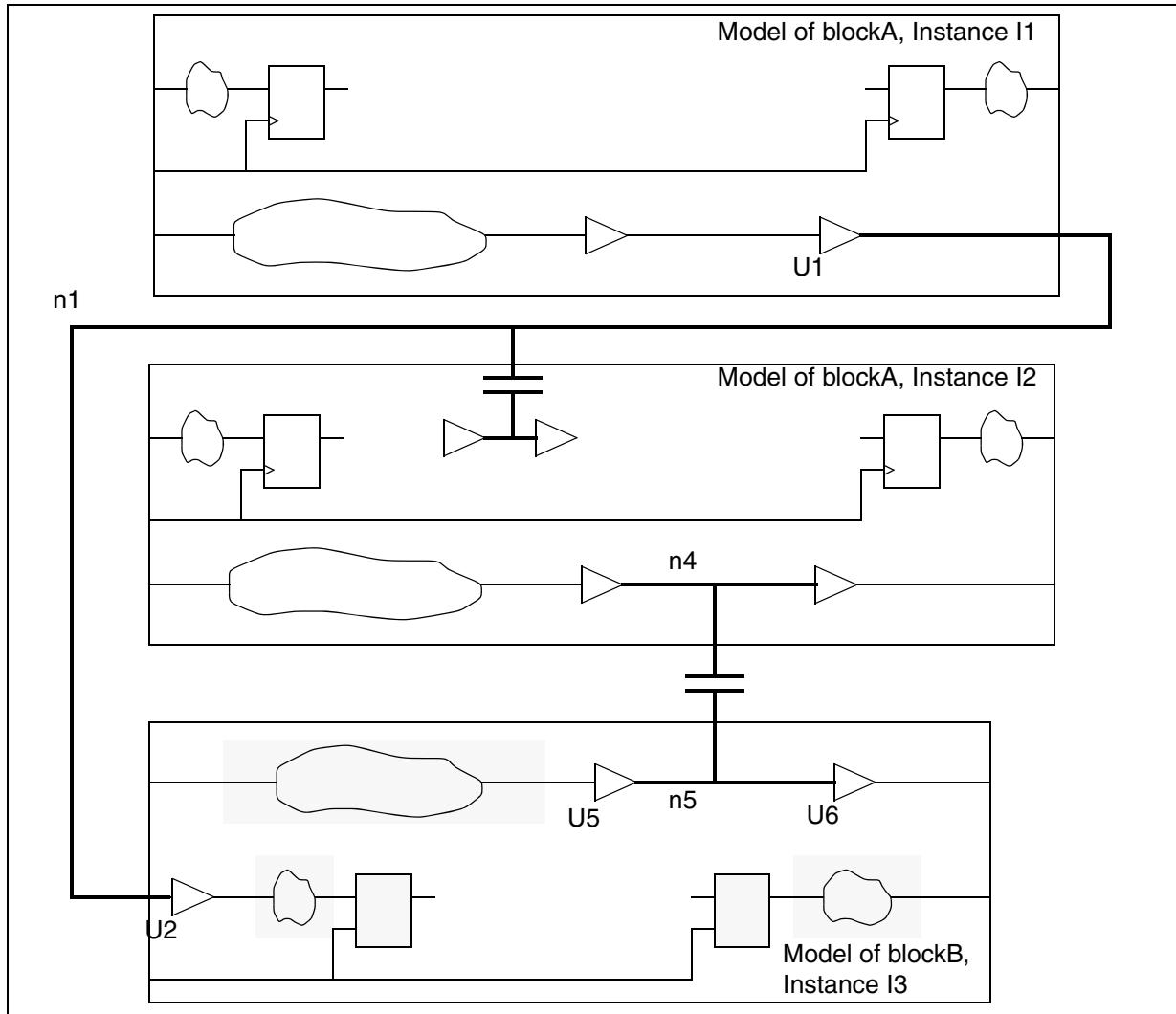
To perform top-level analysis, source the top-level analysis script created by the `create_si_context` command, apply the top-level constraints, and perform the analysis. For example:

```
pt_shell> source top.tcl
pt_shell> source your_top_constraints.tcl
pt_shell> update_timing
pt_shell> report_timing
pt_shell> ...
```

The `top.tcl` script reads in the top-level model and interface logic models, applies the instance-level constraints, applies the arrival window annotation script, and annotates the design with detailed parasitics.

[Figure 20-16](#) shows what the top-level design looks like. The register-to-register paths in the timing models have been eliminated, except in places where they are needed for crosstalk analysis. The `read_parasitics` command in the `top.tcl` script annotates the nets in the top-level design and in the interface logic models of each block. The `-ilm_context` option of the `read_parasitics` command suppresses messages about missing objects, such as nets, cells, and pins, that are present in the parasitic data file, but not used in the design.

Figure 20-16 Top-Level Design With Timing Models



Analysis Iterations Using Annotated Arrival and Slew

If there are timing violations reported in a block-level analysis, such as in step 2 in [Figure 20-12](#), it might be due to the conservative analysis used initially, with infinite arrival windows for aggressor transitions and zero transition times.

For a more accurate analysis, you can back-annotate the newly calculated arrival times and transition times for the context cells, and then run the block-level analysis again for each lower-level block. It is not necessary to generate ILMs again for this process. For example,

this command annotates the timing model pins for the I2 block with new arrival and transition times:

```
pt_shell> write_arrival_annotations -instances {I2} -context
```

The command uses information in the I2/wrapper.txt file and produces a new script file called I2/wrapper.pt.gz, which contains annotations like the ilm.pt.gz file. This is the same process described in [Annotating Arrival and Slew Information](#).

At this point of the flow, however, you might want to annotate all of the contexts of the top-level instances. To do so, use the `write_arrival_annotations -context` command. You can then repeat the block-level and top-level analysis with greater accuracy.

Using an ILM

After you have generated and verified the ILM, you can use it at the chip level in place of the full gate-level implementation for timing analysis. To use the generated model,

1. Read in the netlist for each interface model. For example:

```
pt_shell> read_verilog block_model.v
```

2. Read and link the chip-level design:

```
pt_shell> read_verilog top.v
pt_shell> link_design top
```

3. For each ILM, read in the SDF or parasitics for the block. To specify the hierarchical path name leading to the instance, use the `-path` option with the `read_sdf` and `read_parasitics` commands. Enter one of the following commands:

```
pt_shell> read_parasitics -path block block_model.spf
```

or

```
pt_shell> read_sdf -path block block_model.sdf
```

4. For each ILM, set the current instance to the block and source the script containing assertions and exceptions defined on the block. Use the script that was generated using the `write_ilm_script` command with the `-instance` option. If you need to write any exceptions (those defined relative to clocks on ports), do so before sourcing the script. Enter the following command:

```
pt_shell> current_instance top/block
pt_shell> source block_instance.pt
```

5. To go back to the top-level netlist, enter

```
pt_shell> current_instance
```

6. Perform chip-level timing analysis:

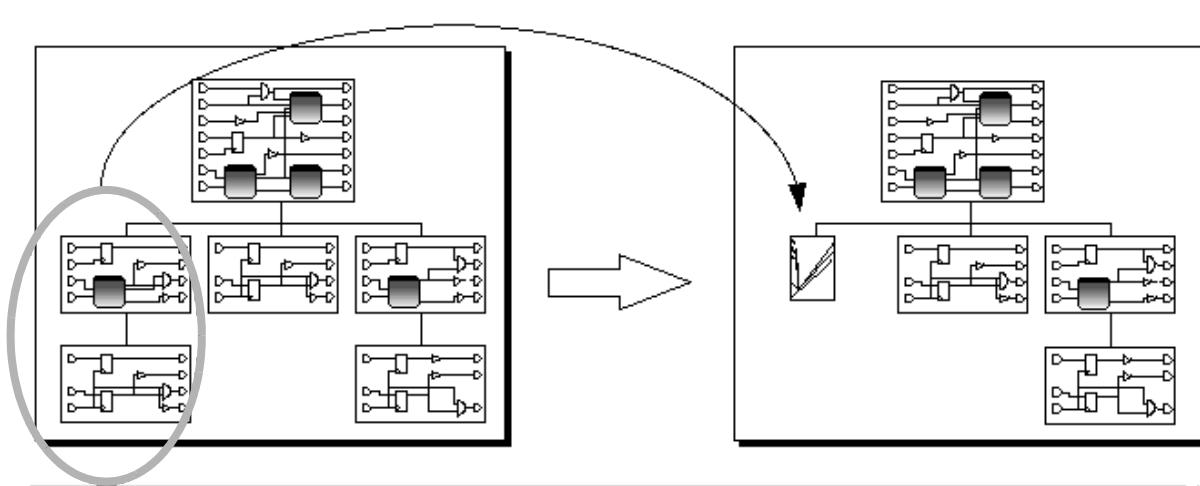
```
pt_shell> report_timing
```

Extracted Timing Models

You can generate an extracted timing model (ETM) for a block with a corresponding technology-mapped gate-level netlist. Using extracted timing models provides the following benefits:

- Reduces the runtime and memory for full-chip analysis
You can run chip-level analysis with extracted models in place of the gate-level netlist for some modules, as shown in [Figure 20-17](#).
- Protects the intellectual property of a netlist-based core

Figure 20-17 Extracted Model of a Bottom-Up Design



Extraction uses a technology-mapped netlist as input and generates a context-independent timing model in the Synopsys .db format or Liberty format.

The generated model contains the same timing behavior as the original netlist. The delay values of arcs in the model are within the user-defined tolerance of the original path delays.

To learn about creating and using extracted timing models, see

- [Model Extraction Overview](#)
- [Timing Model Extraction Requirements](#)
- [Timing Model Extraction Process](#)

- [Limitations of Model Extraction](#)
- [Extracted Model Types](#)
- [Extraction Variables](#)
- [Preparing for Extraction](#)
- [Model Extraction Options](#)
- [Extracted Model Examples](#)
- [Extracting Clock-Gating Checks](#)
- [Extracting Constant Values](#)
- [Extracting Timing Exceptions](#)
- [Restricting the Types of Arcs Extracted](#)
- [Back-Annotated Delay and Layout Information](#)
- [Performing Model Extraction](#)
- [PrimeTime User-Defined Attributes in .lib](#)
- [Merging Extracted Models](#)
- [Extracting Internal Pins](#)

Model Extraction Overview

The `extract_model` command generates a static timing model for the current design from its gate-level netlist. The generated model has the same timing behavior as the original netlist, and can be used in place of the original netlist in a hierarchical timing analysis.

Using an extracted timing model has these advantages:

- The generated model is usually much smaller than the original netlist. When you use extracted models in place of netlists in PrimeTime, you can significantly reduce the time needed to analyze a large design.
- Using a model in place of a netlist prevents a user from seeing the contents of the block, allowing the block to be shared while protecting the intellectual property of the block creator.

Before you can extract a timing model for a design, you must set up the context for extraction. Much of the required information depends on the options you use with the `extract_model` command.

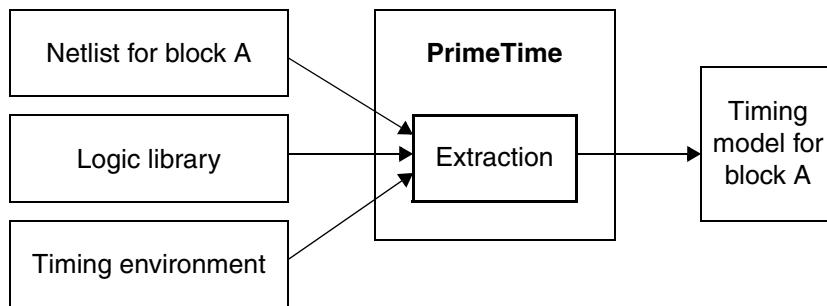
You can control accuracy in model extraction by using the `extract_model` command options and setting the model extraction variables, as explained in [Extraction Variables](#).

Timing Model Extraction Requirements

To generate an extracted timing model, you need the following elements (shown in [Figure 20-18](#)):

- Block netlist that contains more than one cell and net connections between cells
- Logic library
- Timing environment of the block, such as clocks and operating conditions

Figure 20-18 Timing Model Extraction Process



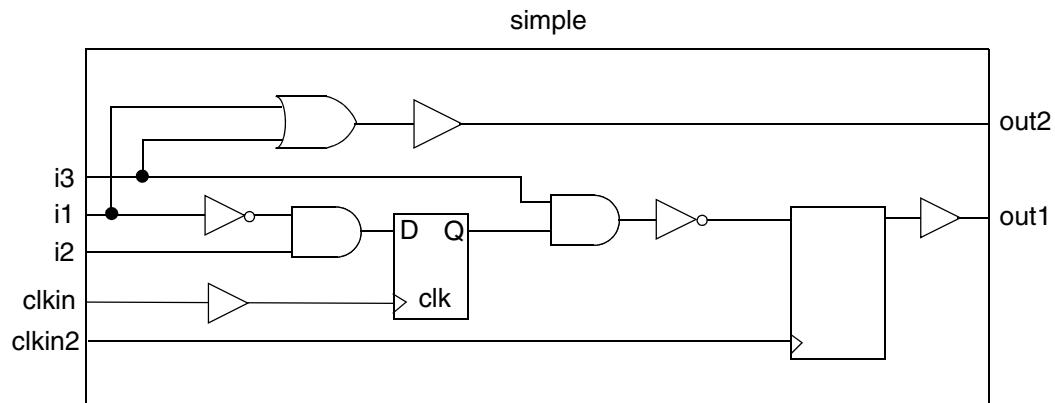
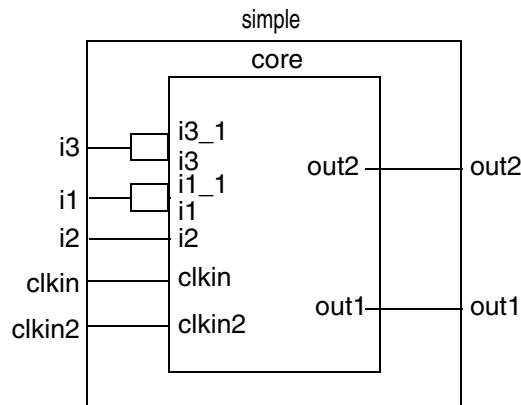
Note:

Before you generate a model, ensure that the netlist contains no timing violations by using the `check_timing` or `report_constraint` command.

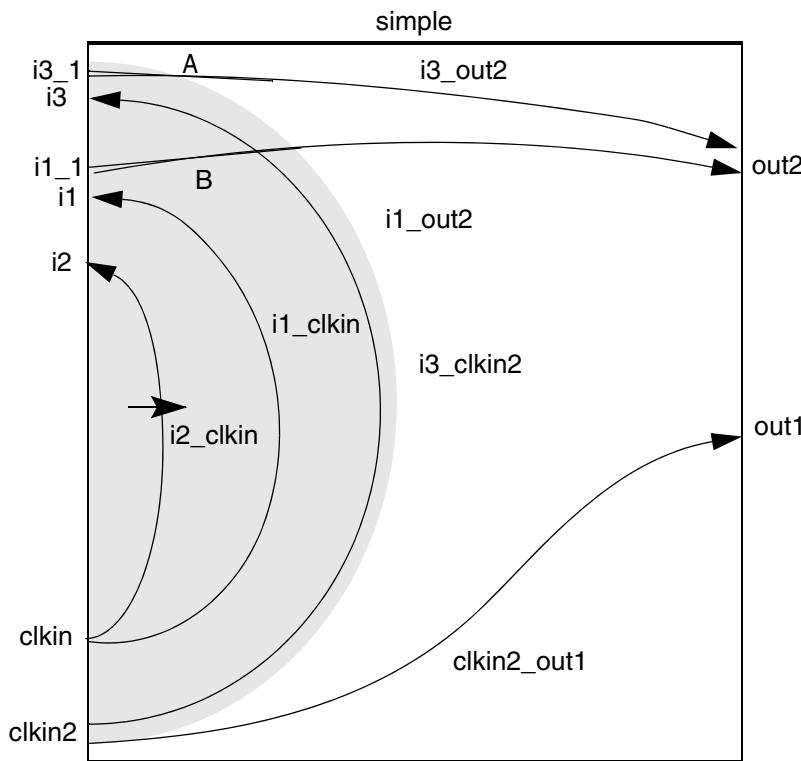
Timing Model Extraction Process

Timing model extraction creates a timing arc for each path from an input port to a register, an input port to an output port, and from a register to an output port.

[Figure 20-19](#) and [Figure 20-20](#) show an example of a gate-level design named “simple” and the model extracted from the netlist.

Figure 20-19 Gate-Level Netlist*Figure 20-20 Netlist of Extracted Model*

The generated timing model is another design containing a single leaf cell, as shown in [Figure 20-20](#). The core cell is connected directly to input and output ports of the model design. This cell contains the pin-to-pin timing arcs of the extracted model. [Figure 20-21](#) shows the timing arcs of the core cell. These arcs are extracted from the timing paths of the original design.

Figure 20-21 Timing Arcs of Core Cell

The delay data in the timing arcs is accurate for a range of operating environments. The extracted delay data varies with the victim slew or load, but the aggressor context remains the same. When the model is used in a design, the arc delays vary with the input transition times and output capacitive loads. This is called a “context-independent” model because it works correctly in a variety of contexts.

The characteristics of the extracted model depend on the operating conditions and wire load model in effect at the time of extraction. However, clocking conditions and external constraints do not affect the model extraction process. Commands, such as `create_clock`, `set_clock_latency`, `set_clock_uncertainty`, `set_input_delay`, and `set_output_delay`, do not affect the model extraction process, but the extracted model, when used for timing analysis, is sensitive to those commands.

Extracting Timing Paths

To learn about extracting various types of information in the original design, see

- [Boundary Nets](#)
- [Internal Nets](#)

- [Paths From Inputs to Registers](#)
- [Paths From Inputs to Outputs](#)
- [Paths From Registers to Outputs](#)
- [Paths From Registers to Registers](#)
- [Clock Paths](#)
- [Interface Latch Structures](#)
- [Minimum Pulse Width and Minimum Period](#)
- [False Paths](#)
- [Clock-Gating Checks](#)
- [Back-Annotated Delays](#)
- [Block Scope](#)
- [Noise Characteristics](#)

Boundary Nets

The extracted model preserves the boundary nets of the original design, thereby maintaining accuracy for computing net delays after the model design is instantiated in another design. However, if you use the `-library_cell` option with the `extract_model` command, PrimeTime does not preserve the boundary nets, but factors the boundary net delays into the model.

If you do not use the `-library_cell` option and boundary nets in the original design have been annotated with detailed parasitic, the information is written in a separate parasitic data file in the standard SPEF format as part of the files generated by the `extract_model` command. For best results, reapply the lumped boundary annotations on the model. When you use the model, you need to read in the saved parasitic information with the `read_parasitics -path path_name` command. For information about using a subdesign with annotated parasitics in a larger design, see [Back-Annotation](#).

Internal Nets

When performing extraction, if the internal nets are not annotated with detailed parasitics, PrimeTime computes the capacitance and resistance of internal nets. The delay values of the arcs in the model are accurate for the wire load model you set on the design before extraction. To get a model that is accurate for a different set of wire load models, set these models on the design and perform a new extraction. If internal nets are annotated with

detailed parasitics, such as Reduced Standard Parasitic Format (RSPF) or SPEF, PrimeTime does not use a wire load model to calculate the net capacitance and resistance. Instead, it takes the values from the detailed parasitics.

If the internal nets are back-annotated with delay or capacitance information, the back-annotated values are used instead of the computed values. For more information, see [Back-Annotated Delays](#).

Paths From Inputs to Registers

A path from an input to a register is extracted into an equivalent setup arc and a hold arc between the input pin and the register's clock. The setup arc captures the delay of the longest path from the particular input to all registers clocked by the same clock, plus the setup time of the register library cell. The hold arc represents the delay of the shortest path from the particular input to all register clocks, including the hold times of the register library cell.

The register's setup and hold values are incorporated into the arc values. The setup and hold value of each arc is a function of the transition time of the input signal and the transition time of the clock signal. If an input pin fans out to registers clocked by different clocks, separate setup and hold arcs are extracted between the input pin and each clock.

Paths From Inputs to Outputs

A path from an input to an output is extracted into two delay arcs. One of the arcs represents the delay of the longest path between the input pin and the related output pin. The other arc represents the delay of the shortest path between the two pins.

The delay values for these arcs are functions of the input signal transition time and output capacitive load. The extracted arc is context-independent, resulting in different delays when used in different environments.

Paths From Registers to Outputs

A path from a register to an output is extracted into two delay arcs. One arc represents the longest path delay between the register's clock pin and the output pin, and the other arc represents the shortest path delay between those pins. The clock-to-data output delay is included in the arc value. Different clock edges result in different extracted arcs.

Similar to input-to-output arcs, the delays of register-to-output arcs are functions of input transition times and output load capacitance. The arc delays differ depending on the environment in which they are used.

Paths From Registers to Registers

Paths from registers to registers are not extracted. For timing arc extraction, PrimeTime only traverses the interface logic.

Clock Paths

Delays and transition times of clock networks are reflected in the extracted model. However, clock latency is not included in the model. Therefore, the source latency and network latency must be specified when the timing model is used.

Interface Latch Structures

The `extract_model` command can extract simple latch structures on the interface of the design (latches with fanin from a primary input or fanout to a primary output). The extracted model preserves only the borrowing behavior specified at the time of model extraction, not all possible borrowing behaviors.

Specifying Latch Borrowing

The following options of the `extract_model` command affect the extraction of latch behavior: `-context_borrow` and `-latch_level`. The `-context_borrow` option is the default behavior, which you can override by using the `-latch_level` option.

When you use the `-context_borrow` option, the tool identifies latches on the interface that borrow, traces through these latches, and stops each trace at a latch that does not borrow. On the other hand, the `-latch_level` option limits the tracing of all latch chains at the interface to the specified length. Use the `-latch_level` option only if you are certain of the borrowing behavior of latches at the interface of the design, and you are able to specify a single latch chain length for the entire design.

The total amount of borrowing should not exceed one clock cycle. If it does, you need to make the model timing match the netlist timing by manually setting multicycle paths in the extracted model with the `set_multicycle_path` command.

How the Model Extraction Handles Latches

The `extract_model` command handles latches as follows:

- It traces through borrowing latches and stops when it encounters a flip-flop, port, or nonborrowing latch.
- It extracts a setup arc when an input port goes through borrowing latches to a flip-flop or nonborrowing latch.
- It extracts a delay arc when an input port, or a clock connected to a flip-flop or nonborrowing latch, goes through borrowing latches to an output port.

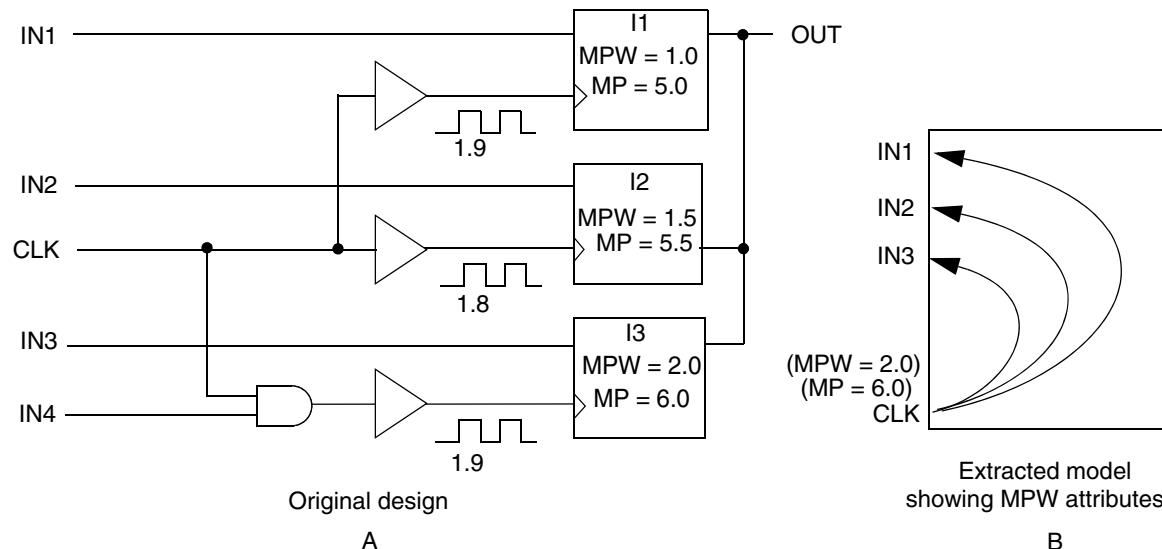
- Neither the `-context_borrow` nor `-latch_level` option to `extract_model` causes the actual time borrowed by a latch to be factored into the generated setup or delay values.
- The timing of the extracted model matches that of the original netlist if the specified borrowing behavior at the time of model generation remains valid for the arrival times defined on input ports and clocks when the model is used. The actual time borrowed on a latch does not need to remain the same. Only the borrowing status (borrowing or not borrowing) must remain the same.

The `write_interface_timing` and `compare_interface_timing` model validation commands are useful for checking the model timing against the netlist timing, especially when there are latches on the interface. The `write_interface_timing` command traverses any borrowing latch in the netlist, which is necessary for matching the numbers reported for the model and for the netlist.

Minimum Pulse Width and Minimum Period

Minimum pulse width and minimum period constraints on cell pins are propagated as minimum pulse width and minimum period attributes on the clock pins that are present in the extracted model, as shown in [Figure 20-22](#).

Figure 20-22 Minimum Pulse Width and Minimum Period Extraction



In the original design A, all latches have minimum pulse width and minimum period attributes on their clock pins. In the extracted model B, the minimum pulse width attributes are translated into attributes on the clock port. Only the maximum of the minimum pulse width and minimum period values are used. CLK has the `MPW_HIGH = 2.0` attribute set on it. In the original design A, the input pin IN4 does not have minimum pulse width or minimum period attributes, even though it is in the fanin of the clock pin of latch I3.

The extracted timing model only looks at the minimum pulse width attributes of the library pin, not the user minimum pulse width constraints applied with the `set_min_pulse_width` command. When the extracted timing model is instantiated, you can still include the user-defined minimum pulse width and minimum period constraints in the scripts at a higher level in the hierarchy.

The delay effects (asymmetrical rise or fall) and slew propagation along the path are not taken into consideration. Violations can occur when the clock pulse width decreases to less than the required minimum pulse width during the course of its propagation toward the latch clock pin. These violations are not reported for the model. For example, in [Figure 20-22](#) the clock waveform at the input latch I3 has a width of 1.9, which is less than the required 2.0. Using 2.0 for the clock port attribute without considering the slew effects causes this minimum pulse width violation to be missed.

False Paths

The model extraction algorithm recognizes and correctly handles false paths declared with the `set_false_path` command.

Clock-Gating Checks

If your design has clock-gating logic, the model contains clock-gating checks. Depending on your environment variable settings, these checks are extracted as a pair of setup and hold arcs between the gating signal and the clock, or a pair of no-change arcs between the gating signal and the clock. For more information, see [Extracting Clock-Gating Checks](#).

Back-Annotated Delays

If the design to be extracted is back-annotated with delay information, the model reflects these values in the extracted timing arcs. Transition time information for each arc is extracted in the same way as if the design were not back-annotated.

The delays of boundary nets, including any back-annotated delay values, are extracted if the `-library_cell` option is used. If the `-library_cell` option is not used, delays of boundary nets are not extracted, whether or not they are back-annotated. These delays are computed (or can be back-annotated) after the model is instantiated in a design and placed in context.

If the delays of boundary cells (cells connected directly to input or output ports) are back-annotated, the delays of the generated model are no longer context-independent. For example, the delays of paths from inputs are the same for different transition times of input signals. In addition, the delays of paths to outputs are not affected by the input transition time and output load.

Note:

Do not back-annotate the delays of output boundary cells if you want the output delays to change as a function of output load. Similarly, do not back-annotate the delays of input boundary cells if you expect the arc delays to depend on input transition times. To remove already-annotated information, use the `remove_annotated_delay` command.

Block Scope

An extracted timing model is context-independent (valid in different contexts) if the external timing conditions are within the ranges specified for block-level analysis. When the timing model is used at a higher level of hierarchy, it is no longer possible to check the internal clock-to-clock timing of the block. To ensure that the block context is valid for the original block-level timing analysis, you can check the block “scope” at the higher level.

When you generate the extracted model, you can also generate a “block scope” file containing information about the ranges of timing conditions used to verify the timing of the block. When you use the timing model in a chip-level analysis, you can have PrimeTime check the actual chip-level conditions for the block instance and verify that these conditions are within the ranges recorded in the scope file. This process ensures the validity of the original block-level analysis.

To generate the block scope file, use the `-block_scope` option of the `extract_model` command. To check the scope of the timing model during chip-level analysis, use the `check_block_scope` command. For more information, see [Hierarchical Scope Checking](#).

Noise Characteristics

PrimeTime SI users can analyze designs for crosstalk noise effects. An extracted timing model supports noise analysis by maintaining the noise immunity characteristics at the inputs of the extracted model, and by maintaining the steady-state resistance or I-V characteristics at the outputs of the extracted model. However, an extracted model does not maintain the noise propagation characteristics of the module. Also, any cross-coupling capacitors between the module and the external circuit are split to ground. Therefore, PrimeTime SI does not analyze any crosstalk effects between the extracted model and the external circuit.

Note:

To create timing models that can handle crosstalk analysis between modules and between different levels of hierarchy, use interface timing models instead of extracted timing models. See [Hierarchical Crosstalk Analysis](#).

To include noise characteristics in the extracted model, use the `-noise` option of the `extract_model` command. Otherwise, by default, no noise information is included in the extracted model. If you use the `-noise` option, but no noise information is available in the module netlist, then the only noise characteristics included in the extracted model are the

steady-state I-V characteristics of the outputs. In that case, the model extractor estimates the output resistance from the slew and timing characteristics.

An extracted timing model can be created in two forms: a wrapper plus core or a library cell. A wrapper-plus-core model preserves the input nets, the noise immunity curves of all first-stage cells at the inputs, the output nets, and all last-stage driver cells at the outputs (including their I-V characteristics). For a library cell, the model extractor must combine parallel first-stage cells at each input to make a single noise immunity curve, and must combine parallel last-stage cells at each output to make a single driver. For this reason, a wrapper-plus-core model provides better accuracy for noise analysis than a library cell model.

An extracted timing model with noise can be created in two forms: wrapper-plus-core or library cell. A wrapper-plus-core model preserves the input nets, the noise immunity curves of all first-stage cells at the inputs, the output nets, the noise I-V curve or steady-state resistance at the outputs, and the accumulated noise at the outputs. For a library cell, the model extractor combines the inputs nets (including coupling) and all leaf-cell noise immunity curves or DC margins into a single, worst-case noise immunity curve. Also, the library cell combines the worst-case resistance of last-stage cells and net resistance at each output to make a single I-V curve or resistance for each noise region. For this reason, a wrapper-plus-core model provides better accuracy for noise analysis than a library cell model.

To create a library cell model, the model extractor considers the noise immunity curves of individual cells connected in parallel within the module netlist. It considers the worst-case (lowest) data points of multiple curves and combines them into a single worst-case curve for the library cell input. For general information about static noise analysis, see [Static Noise Analysis](#).

Other Extracted Information

In addition to the timing paths, the model extraction algorithm extracts other types of information:

Mode information

Timing modes define specific modes of operation for a block, such as the read and write mode. PrimeTime extracts modes of the original design as modes for the timing arcs in the extracted model.

For a description of how to define mode information for a design, see [Mode Analysis](#).

Generated clocks

Generated clocks you specify in the original design become generated clocks in the extracted model. For more information about generated clocks, see [Clocks](#).

Capacitance

PrimeTime transfers the capacitance of input and output ports of the original design to the model's extracted ports.

Design rules

PrimeTime extracts the maximum transition at input and output ports. PrimeTime reflects maximum capacitance at output ports within the design in the extracted model.

Operating conditions

The values of timing arcs in the extracted model depend on the operating conditions you set on the design when you perform the extraction.

Design name and date of the extraction

PrimeTime preserves the design name and the date of extraction in the extracted model.

Multicycle paths

PrimeTime extracts multicycle paths and writes that information to a script containing a set of `set_multicycle_path` commands that can be applied to the extracted model.

Three-state arcs

Three-state arcs ending at output ports are included in the extracted model.

Preset and clear delay arcs

Preset and clear arcs are extracted if enabled. For more information, see [Extraction Variables](#).

Clock latency arcs

Clock latency arcs are extracted if enabled (see [Extraction Variables](#)). Clock latency arcs are used by tools, such as Physical Compiler to compensate and balance clock tree skews at chip-level. They are also reported by the `report_clock_timing` command in PrimeTime.

Limitations of Model Extraction

This section provides general guidelines and lists the general limitations of model extraction in PrimeTime, the limitations of using extracted models in Design Compiler, and the limitations of extracted models in .lib format.

Observe the following model extraction guidelines:

- Complex latch structures on the design interface are not supported. For more information, see [Interface Latch Structures](#).

- Avoid using the `-remove_internal_arcs` and `-latch_level` options.

The following environment variables affect model extraction:

```
extract_model_capacitance_limit
extract_model_clock_transition_limit
extract_model_data_transition_limit
extract_model_enable_report_delay_calculation
extract_model_gating_as_nochange
extract_model_num_capacitance_points
extract_model_num_clock_transition_points
extract_model_num_data_transition_points
extract_model_status_level
extract_model_with_clock_latency_arcs
timing_clock_gating_propagate_enable
timing_disable_clock_gating_checks
timing_enable_preset_clear_arcs
```

The following limitations apply to model extraction:

- Minimum and maximum delay constraints

Paths in the original design that have a minimum or a maximum delay constraint set on them are treated by extraction as follows:

- If you set the constraint on a timing path, PrimeTime ignores the constraint in the model.
 - If you set the constraint on a pin of a combinational cell along a path, PrimeTime ignores the timing paths that pass through this pin. These paths do not exist in the extracted model.
- Data checks that are specified with the `set_data_check` command are not supported.

For these reasons, it is recommended that you remove all minimum and maximum delay constraints from the design before you perform extraction.

Input and output delays

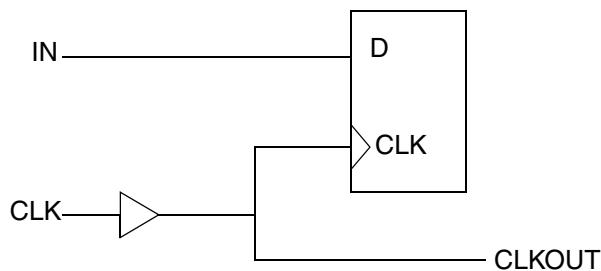
PrimeTime ignores input and output delays set on design ports when it extracts a model, unless your design contains transparent latches and you specify the `-context_borrow` option of the `extract_model` command.

Input or output delay values set on pins of combinational cells along a timing path cause the paths that pass through these pins to be ignored in the extracted model. It is recommended that you remove input and output delay values set on internal paths (or set on objects other than ports) from the design before performing an extraction.

Extraction of load-dependent clock networks

If the delay of a clock to register path in the original design depends on the capacitive load on an output port, the delay of the extracted setup arc to the register in the model is independent of the output load. For example, consider the design shown in [Figure 20-23](#). In this design, the setup time from the IN input relative to the CLK clock depends on the load on the CLKOUT output. This occurs because the delay of the clock network depends on the load on the CLKOUT output and because the transition time at the register's clock pin depends on the output load.

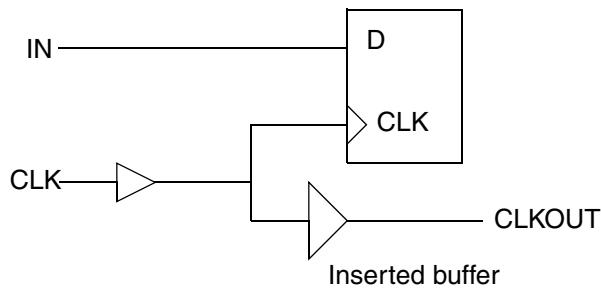
Figure 20-23 Load-Dependent Clock Network



In the extracted model, the setup time from input IN to input CLK is independent of the load on output CLKOUT. However, the extracted CLK delay from input to CLKOUT remains load dependent.

To avoid inaccuracies in the extracted model, isolate the delay of the clock network inside the design to be modeled from the changes in the environment by inserting an output buffer, as shown in [Figure 20-24](#).

Figure 20-24 Isolating the Delay of the Clock Network



Limitations of Extracted Models in Design Compiler

Design Compiler does not support all the features of extracted timing models. Specifically, Design Compiler ignores mode information. All modes are considered enabled.

Limitations of Extracted Models in .lib Format

You can optionally specify .lib format for the extracted model. This format is useful for exporting the timing model to an external tool. The following limitations apply to a model extracted in this format:

- Like extracted models in other formats, the .lib model captures only the timing behavior, not the logic, of the original netlist.

Extracted Model Types

The `extract_model` command can create two different types of timing models: a library cell or a wrapper with a core cell inside. A library cell serves as a port-to-port replacement for the design being modeled, with the boundary net capacitances approximated inside the model. A wrapper-and-core model consists of a central core cell surrounded by a wrapper design that preserves the original boundary nets.

The type of model created by `extract_model` depends on the `-library_cell` option. If the `-library_cell` option is present, PrimeTime creates a library cell. The name assigned to the new cell is the same as the design name.

If the `-library_cell` option is absent, PrimeTime creates a wrapper design with a core cell. The `-format` option controls the output format of the core cell. However, PrimeTime always writes out the wrapper design in .db format, irrespective of the `-format` setting. The name of the core cell is *design_name_core*. The name of the wrapper design is *output.db*, where *output* is the value specified by the `-output` option.

You can optionally create a test design containing an instance of the extracted library cell. To do this, use the `-test_design` option along with `-library_cell` in the `extract_model` command. The test design is named *design_name_test* and the output file is named *output_test.db*. PrimeTime does not write parasitics to the test design.

Extraction Variables

Several environment variables control the model accuracy and other aspects of timing model extraction. To attain the level of accuracy and complexity you want, consider each variable setting.

Table 20-4 lists the environment variables and summarizes how they affect the `extract_model` command.

Table 20-4 Extraction Environment Variables

Extraction environment variable	Effect on <code>extract_model</code> command
<code>extract_model_capacitance_limit</code>	Specifies the maximum load capacitance on outputs. Model extraction characterizes the timing within this specified limit.
<code>extract_model_clock_transition_limit</code>	For clock input ports, specifies the maximum input port transition time. Model extraction characterizes the timing within this specified limit.
<code>extract_model_data_transition_limit</code>	For data input ports, specifies the maximum input port transition time. Model extraction characterizes the timing within this specified limit.
<code>extract_model_enable_report_delay_calculation</code>	Specifies whether to allow the final user of the extracted model to get timing information using the <code>report_delay_calculation</code> command. Set this variable to <code>false</code> if the timing calculations for the model are proprietary.
<code>extract_model_gating_as_nochange</code>	Extracts clock-gating setup and hold checks as corresponding no-change arcs.
<code>extract_model_num_capacitance_points</code>	Specifies the number of capacitance data points used in the delay table for each arc.
<code>extract_model_num_clock_transition_points</code>	Specifies the number of transition time data points used to make the delay table for each clock arc.
<code>extract_model_num_data_transition_points</code>	Specifies the number of transition data points used to make the delay table for each data arc.
<code>extract_model_status_level</code>	Specifies the amount of detail provided in model extraction progress messages.
<code>extract_model_with_clock_latency_arcs</code>	Specifies whether to enable modeling of clock tree insertion delay timing arcs in the extracted model.

Table 20-4 Extraction Environment Variables (Continued)

Extraction environment variable	Effect on <code>extract_model</code> command
<code>timing_clock_gating_propagate_enable</code>	Determines whether data signals that gate a clock are propagated past the gating logic.
<code>timing_disable_clock_gating_checks</code>	When set to <code>true</code> , disables all clock-gating checks and does not extract them to the model.
<code>timing_enable_preset_clear_arcs</code>	Specifies whether to enable or disable preset and clear timing arcs.

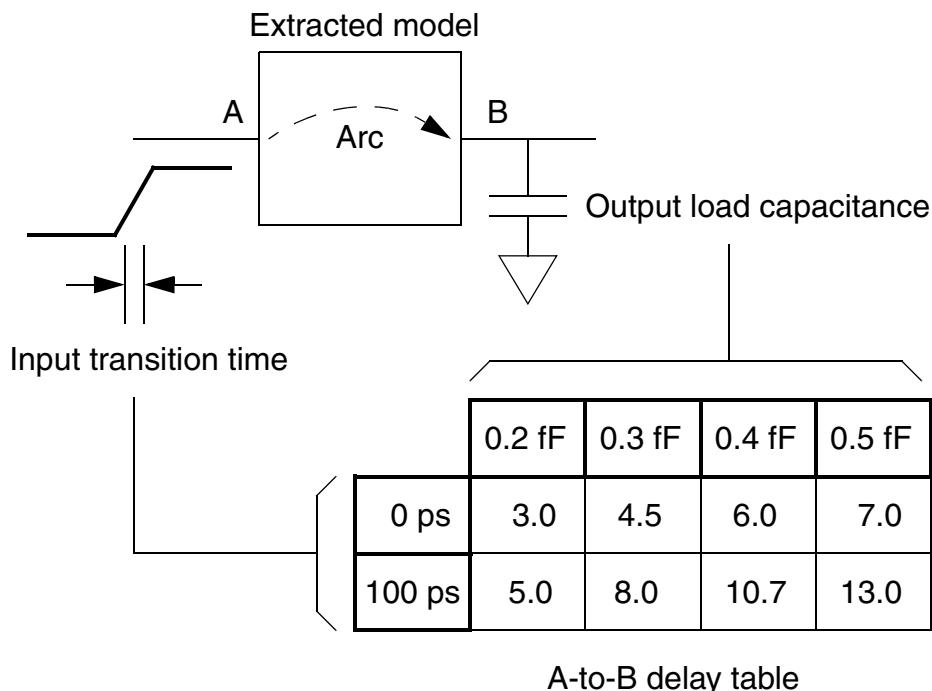
Variable Setting Guidelines

To generate accurate timing models, observe these guidelines:

- If the design contains latches on the interface, run `extract_model` with the `-context_borrow` option.
- Avoid the `-remove_internal_arcs` and `-latch_level` options of the `extract_model` command because they compromise accuracy.

Delay Table Generation

When you use the `extract_model` command, PrimeTime extracts a set of timing arcs and creates a delay table for each arc. A delay table is a matrix that specifies the arc delay value as a function of the input transition time and the output load capacitance. For example, the delay arc shown in [Figure 20-25](#) uses a delay table similarly to the 2-by-4 table shown. The amount of delay for the arc depends on the input transition time and output load capacitance.

Figure 20-25 Delay Table Example

When you use the extracted model in a design, PrimeTime calculates the arc delay according to the context where the model is being used. The delay table makes the model accurate for the range of input transition times and output loads defined in the table.

For transition times or output loads at intermediate values within the table, PrimeTime uses interpolation between the table points to estimate the delay. A table with a larger number of entries, such as a 10-by-10 matrix, provides better accuracy at the cost of more memory and CPU time for model generation.

For transition times and output loads outside of the ranges defined in the table, PrimeTime uses extrapolation to estimate the delay. A table that spans a larger range of transition time or load capacitance provides better accuracy for a larger range of conditions, but wastes memory and CPU resources if the range is larger than necessary to accommodate actual usage conditions.

You can control the number of data points and the range of conditions specified in delay tables of models generated by model extraction. To do so, set the following variables:

```

extract_model_num_capacitance_points
extract_model_num_clock_transition_points
extract_model_num_data_transition_points

extract_model_capacitance_limit
extract_model_clock_transition_limit
extract_model_data_transition_limit

```

The number or *_points variables control the number of data points in the generated delay tables. By default, these variables are set to 5, resulting in 5-by-5 delay tables. The *_limit variables control the range of capacitance and transition times covered in the generated delay tables. The default settings are 64 pf for the capacitance limit and 5 ns for the transition time limits.

Preparing for Extraction

To prepare your design for extraction, consider the following tasks:

- [Setting the Extraction Variables](#)
- [Setting the Operating Conditions](#)
- [Defining the Wire Load Models](#)
- [Defining Clocks](#)
- [Setting Up Latch Extraction](#)
- [Identifying False Paths](#)
- [Removing Internal Input and Output Delays](#)
- [Controlling Mode Information in the Extracted Model](#)
- [Performing Timing Checks](#)
- [Preparing for Crosstalk Analysis](#)

Setting the Extraction Variables

Several environment variables affect the extraction process, as explained in [Extraction Variables](#). Set them as needed by using one of these methods:

- Specify variables and their corresponding values individually. For example, enter

```
pt_shell> set_app_var extract_model_capacitance_limit 5.0
```
- Specify a variable and its value in a script file, then source the batch file. For example, enter

```
pt_shell> source script_file
```
- Specify the variable and its value in the .synopsys_pt.setup (PrimeTime setup) file.

Setting the Operating Conditions

The operating conditions for vendor libraries vary. For example, some conditions might be nominal, worst-case commercial (WCCOM), and best-case commercial (BCCOM).

Model extraction uses the current single operating condition or current minimum and maximum operating conditions. For example, for a single operating condition, enter:

```
pt_shell> set_operating_conditions WCCOM
pt_shell> extract_model -output model
```

To create a single conservative model using on-chip variation to calculate the timing arcs, use commands like the following:

```
pt_shell> set_operating_conditions -min BEST_COND -max WORST_COND \
           -library your_lib -analysis_type on_chip_variation
pt_shell> extract_model -output file_name -format db
```

You can also extract two different models at two different operating conditions, and then invoke the two models for min-max analysis. For example, you can use a script similar to the following:

```
set link_path "* your_library.db"
read_db BLOCK.db
link BLOCK
source constraint.pt
read_parasitics ...
read_sdf ...
set_operating_conditions WORST_COND -library your_library.db
extract_model -format db -output CORE_max
set_operating_conditions BEST_COND -library your_library.db
extract_model -format db -output CORE_min
remove_design -all
```

Then you can use instances of the core wrapper CORE_max.db in your design, and invoke the two models for min-max analysis using a script similar to the following:

```
set link_path "* CORE_max_lib.db your_library.db"
read_db CHIP.v
set_min_library CORE_max_lib.db -min_version CORE_min_lib.db
link CHIP
...
```

Defining the Wire Load Models

Defining the wire load models is a prerequisite for extraction unless the delays of all internal nets in the design have been back-annotated. The `extract_model` command uses wire load models to calculate the net capacitance and net delays in the extracted paths.

Use the `set_wire_load_model` and `set_wire_load_mode` commands to define one or more wire load models.

For example, enter the following commands to specify a 20-by-20 wire load model for the top design and a 05-by-05 wire load model for block U4.

```
pt_shell> set_wire_load_mode enclosed
pt_shell> set_wire_load_model -name 20x20 -library class
pt_shell> set_wire_load_model -name 05x05 -library class U4
```

The delay values of the arcs in the model are accurate for the wire load model you set on the design before extraction. To get an accurate model for a different set of wire load models, set wire load models on the design and perform a new extraction.

Defining Clocks

Defining all clocks in the current design is a prerequisite for extraction. You identify pins as clocks by using the `create_clock` or `create_generated_clock` command.

The `extract_model` command uses the clock period and input delay information under the following conditions:

- If you use the `-context_borrow` option, the model extractor uses the clock information to determine which transparent latches are borrowing.
- If you set a multicycle path definition at a point that the model extractor cannot easily move to the design boundary, the clock period determines which path is considered the critical path.

Typically, you place the created clocks on input ports of your design. Clocks created with the `create_clock` command on pins inside the design produce internal clocks in the generated model that require special attention. That is, you must specify the clocks on the pins in the model again; however, generated clocks referenced to boundary ports are extracted as expected and require no extra setup when using the extracted model.

Setting Up Latch Extraction

If the design you want to extract contains transparent latches and you want the generated model to exhibit time-borrowing behavior, you can use the `-context_borrow` and `-latch_level` options of the `extract_model` command to specify the method of latch extraction.

When you use the `-context_borrow` and `-latch_level` options, observe these guidelines:

- If your design has no transparent latches or has transparent latches with no time borrowing, these option settings have no effect on the results.
- With a latch-based design, do not use the `-latch_level` option together with the `-context_borrow` option.

- Setting `-latch_level` option to 0 or 1 is preferred. Using `-context_borrow` or `-latch_level 2` or above can cause some mismatches in model validation. Most of these mismatches are because the total amount of borrowing has exceeded one clock cycle, and it is required to manually set multicycle paths in the model. When performing model validation, use the corresponding `-latch_level` option of the `write_interface_timing` command.
- If the extracted model is to be used in an environment where the clock waveforms or input arrival times (or both) are expected to change drastically, do not use the `extract_model` command. Instead, generate an interface logic model as described in [Interface Logic Models](#).

For more information about extraction of latch behavior, see [Interface Latch Structures](#).

Identifying False Paths

Before you perform model extraction, define false paths in your design by using the `set_false_path` command. Paths defined as false are not extracted into timing arcs by the `extract_model` command. For example, enter the following command to define the path between input IN4 and output OUT1 as a false path.

```
pt_shell> set_false_path -from IN4 -to OUT1
```

Removing Internal Input and Output Delays

Remove input or output delays set on design objects that are not input or output ports. These objects are usually input or output pins of cells. To remove delays, remove the commands from the original script that set the delays.

Controlling Mode Information in the Extracted Model

A complex design might have several modes of operation with different timing characteristics for each mode. For example, a microcontroller might be in setup mode or in monitor mode. A complex design might also have components that themselves have different modes of operation. A common example is an on-chip RAM, which has a read mode and a write mode.

The `extract_model` command creates a model that reflects the current mode settings of the design. The generated model itself does not have modes, but contains timing arcs for all the paths that `report_timing` detects with the modes at their current settings.

If you extract multiple timing models from a design operating under different modes, different conditions set with case analysis, or different exception settings, you can merge those models into a single model that has operating modes. For more information, see [Merging Extracted Models](#).

Performing Timing Checks

To check your design for potential timing problems before you extract, use the `check_timing` command. Because problems reported by the `check_timing` command affect the generated model, fix these problems before you generate the timing model.

Preparing for Crosstalk Analysis

If you enable PrimeTime SI crosstalk analysis, the `extract_model` command can consider crosstalk effects when calculating the timing arcs for the extracted model.

The model extractor cannot calculate the crosstalk-induced delays for all combinations of input transition times for victim and aggressor nets. Instead, it accounts for crosstalk effects in a conservative manner. You specify a range of input delays and slews for the model before extraction. When you use the `update_timing` command, PrimeTime SI uses this information to calculate the worst-case changes in delay and slew that can result from crosstalk. The model extractor then adds the calculated changes to the extracted timing arcs.

To include crosstalk effects during model extraction:

1. Use the `set_input_delay` command with both the `-min` and `-max` options to specify the worst-case timing window for each input, given the current clock configuration.
2. Use the `set_input_transition` command with both the `-min` and `-max` options to specify the worst-case slew change for each input.
3. With PrimeTime SI crosstalk analysis enabled (by setting the `si_enable_analysis` variable to `true`) and with the design back-annotated with cross-coupling capacitors, perform a crosstalk analysis with the `update_timing` command. PrimeTime SI calculates the worst-case delay changes and slew changes under the specified conditions using on-chip variation analysis.
4. Run the `extract_model` command in the usual manner. PrimeTime performs model extraction without crosstalk analysis, then adds the fixed delta delay and delta slew values to the resulting timing arc values.

In step 4, PrimeTime adds only crosstalk values that make the model more conservative. For example, it adds a delta delay value to a maximum-delay arc only if the delta delay is positive, or to a minimum-delay arc only if the delta delay is negative.

The transition times you define with the `set_input_transition` command are used to calculate the delta delay values. These transition times are preserved in the extracted model as a set of design rules. When you use the extracted model, if a transition time at an input port of the model is outside of the defined range, you receive a warning message.

Model Extraction Options

The `extract_model` command has a `-format` option, which can be set to any combination of `db` or `lib` to generate models in `.db` format or `.lib` format. The `.db` format can be used directly by most Synopsys tools. Use the `.lib` format when you want to be able to read and understand the timing arcs contained in the model, or for compatibility with a third-party tool that can read `.lib` files. The `.lib` model can be compiled by Library Compiler to get a `.db` file. To control the level of detail and accuracy of the extracted model, use the `extract_model` command with these options:

- `-library_cell` – Generates the model as a library cell rather than a wrapper and core.
- `-remove_internal_arcs` – Generates the model with internal arcs and timing points removed.

By default, the generated model is a design containing a single core cell. The model preserves all the boundary nets in the original design and is the more accurate type of model.

The `-library_cell` option causes the `extract_model` command to generate the model as a library cell instead of a wrapper design and core. In this case, all boundary net delays are added to the arcs in the model.

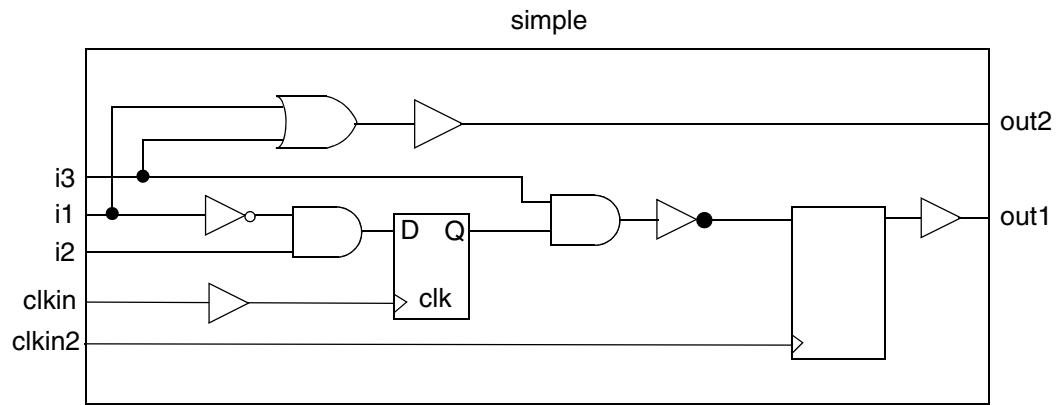
The library cell model is simpler than the detailed model. However, the boundary net delays are not as accurate. In the case where outputs are shorted, the dependence of output delay on the capacitance of other outputs is lost.

You can use the `-test_design` and `-library_cell` options together to have the model extractor generate a test design that instantiates the model. You can link the model to this design and obtain timing reports using the `report_timing` command. This design is not part of the model. It is provided as a convenience for testing the model after extraction.

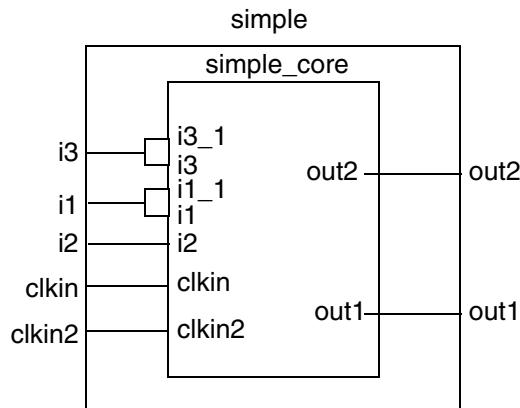
If you have a PrimeTime PX license, you can use the `-power` option to generate a power model. By using this option, you can store power data for IPs and large blocks in a model that you can instantiate. This option performs an implicit `update_power`, if necessary; therefore, you must set the `power_enable_analysis` variable to `true` before using the `extract_model -power` command. For more information about this option, see the *PrimeTime PX User Guide*.

Extracted Model Examples

Figure 20-26 shows a gate-level netlist for a design called simple.

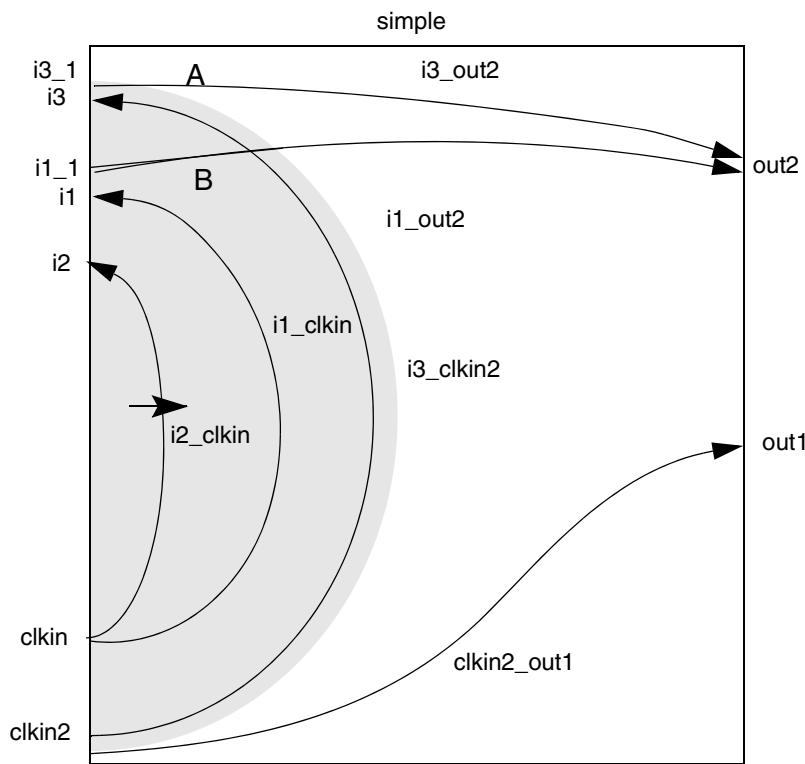
Figure 20-26 Gate-Level Netlist

[Figure 20-27](#) shows the extracted model when you do not use the `-library_cell` option. The generated model is a design called simple. This design contains an instance of a single cell called simple_core, which contains all the timing arcs in the model.

Figure 20-27 Extracted Model Without -library_cell

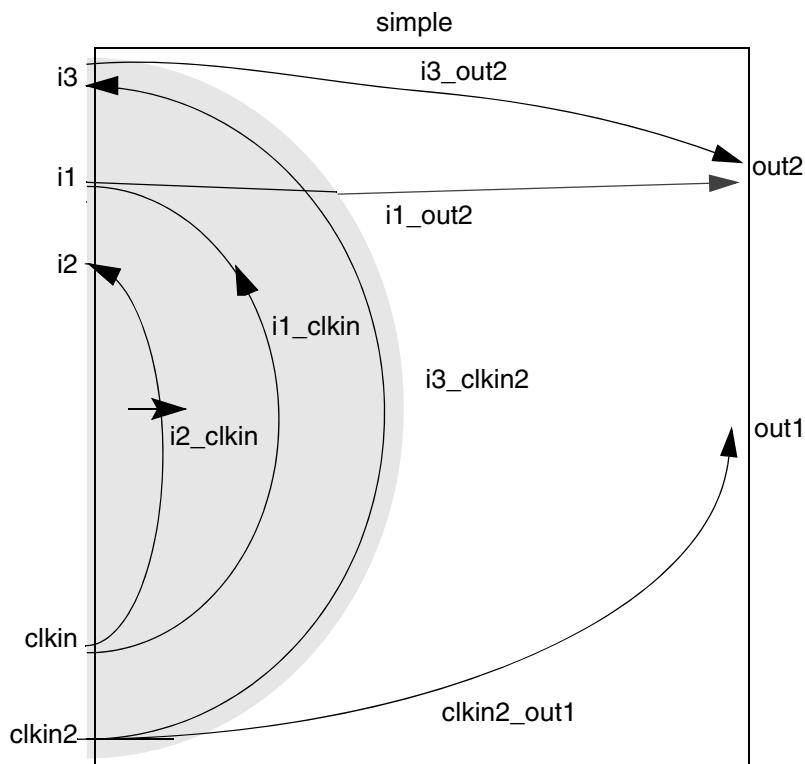
[Figure 20-28](#) shows the core cell and its timing arcs.

Figure 20-28 Core Cell and Timing Arcs



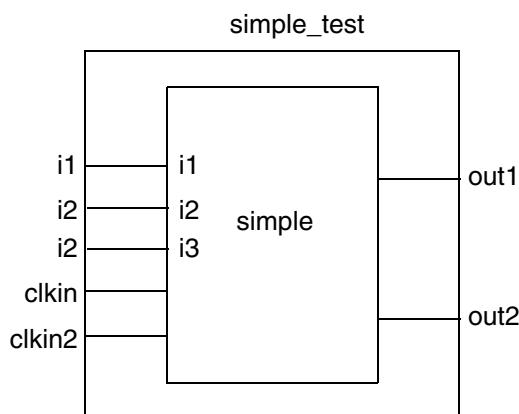
[Figure 20-29](#) shows the model generated when you specify the `-library_cell` option. The model is a library cell called simple. This cell has the same input ports and output ports as the model. It also contains all the timing arcs. This model has no boundary nets. All boundary net delays are lumped into the generated model.

Figure 20-29 Model Extracted With -library_cell



[Figure 20-30](#) shows a test design generated when you specify the `-test_design` option. The design is called `simple_test`, which instantiates the model. This design is generated for your convenience in obtaining model and timing reports for the model if the model is generated as a library cell.

Figure 20-30 Test Design That Instantiates the Model



Extracting Clock-Gating Checks

The clock-gating setup and hold constraints, which are between the clock-gating pin and the clock, are converted to setup and hold checks between the primary input pin and input clock pin, and then written out to the model.

The following items can affect the way clock-gating checks are extracted:

Clock propagation

You can select whether to enable or disable clock delay propagation by using the `set_propagated_clock` and `remove_propagated_clock` commands.

Clock-gating checks

You can use the `set_clock_gating_check` command to add clock-gating setup or hold checks of any desired value to any design object. For information about clock-gating setup and hold checks, see [Specifying Clock-Gating Setup and Hold Checks](#).

Enabling or disabling gating checks

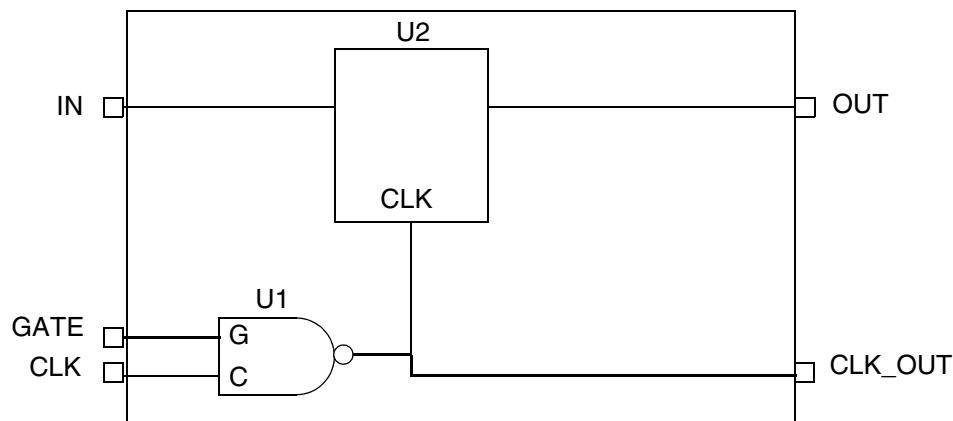
Extraction of clock-gating checks can be enabled or disabled with the `timing_disable_clock_gating_checks` variable before extraction.

Extraction of Combinational Paths Through Gating Logic

Combinational paths that start at input or inout ports, go through clock-gating logic, and end at output or input ports are extracted in the model.

In [Figure 20-31](#), the path from GATE through U1/G to CLK_OUT is a combinational path in the clock-gating network and is extracted if the `timing_clock_gating_propagate_enable` variable is set to its default of `true`.

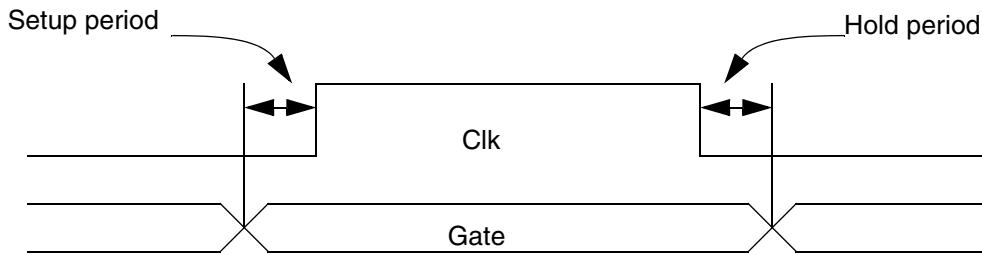
Figure 20-31 Combinational Path Through Clock-Gating Logic



Extraction of Clock-Gating Checks As No-Change Arcs

Clock-gating setup and hold checks can be grouped and merged to form a no-change constraint arc. The no-change arc is a signal check relative to the width of the clock pulse. PrimeTime establishes a setup period before the start of the clock pulse and a hold period after the clock pulse (see [Figure 20-32](#)).

Figure 20-32 No-Change Arc



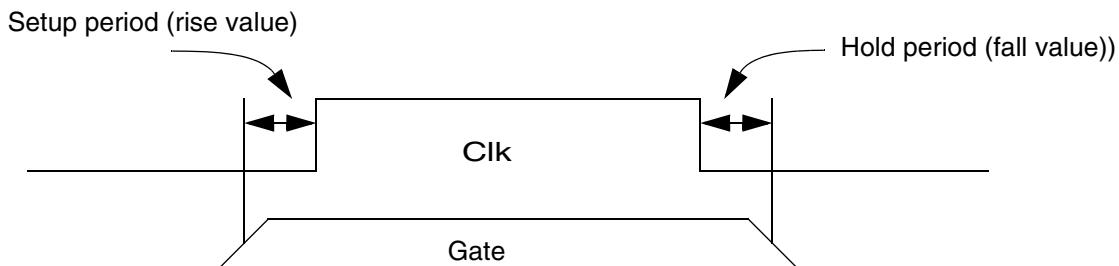
A clock-gating setup check can be combined with its corresponding clock-gating hold check to produce two no-change arcs. One arc is the no-change condition with the clock-gating signal low during the clock pulse, the other is the gating signal high during the clock pulse.

The `extract_model_gating_as_nochange` variable, when set to `true`, causes the model extractor to convert all clock-gating setup and hold arcs into no-change arcs. When set to its default of `false`, the clock-gating checks are represented as a clock-gating constraint. For example, if the variable is set to `true`, and if the clock pulse leading edge is rising and the trailing edge is falling, the no-change arcs produced are as follows:

- NOCHANGE_HIGH_HIGH

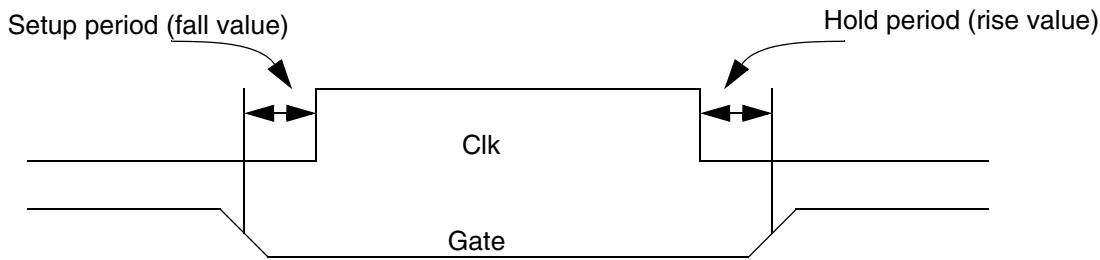
Rise table taken from the setup arc rise table. Fall table taken from the hold arc fall table. See [Figure 20-33](#).

Figure 20-33 NOCHANGE_HIGH_HIGH Arc



- NOCHANGE_LOW_HIGH

Rise table taken from the hold arc rise table. Fall table taken from the setup arc fall table. See [Figure 20-34](#).

Figure 20-34 NOCHANGE_LOW_HIGH Arc

Extracting Constant Values

If the output ports are driven to a constant value, this value is preserved in the model. If the extracted model is used in a design, the constant value is propagated into its fanout logic.

Extracting Timing Exceptions

When you specify a timing exception for a path, PrimeTime does not treat that path as an ordinary single-cycle path. The most common type of exception is a false path. Other exception types are multicycle, max_delay, and min_delay paths. You can also use mode analysis (`define_design_mode`, `set_mode`) to control the timing analysis applied to certain defined paths.

The following rules apply to conflicts between timing exceptions and mode analysis:

- If a conflict arises between a specified false path and a moded endpoint, the false path specification prevails.
- If a conflict arises between a specified multicycle path and a moded endpoint, the mode is propagated.

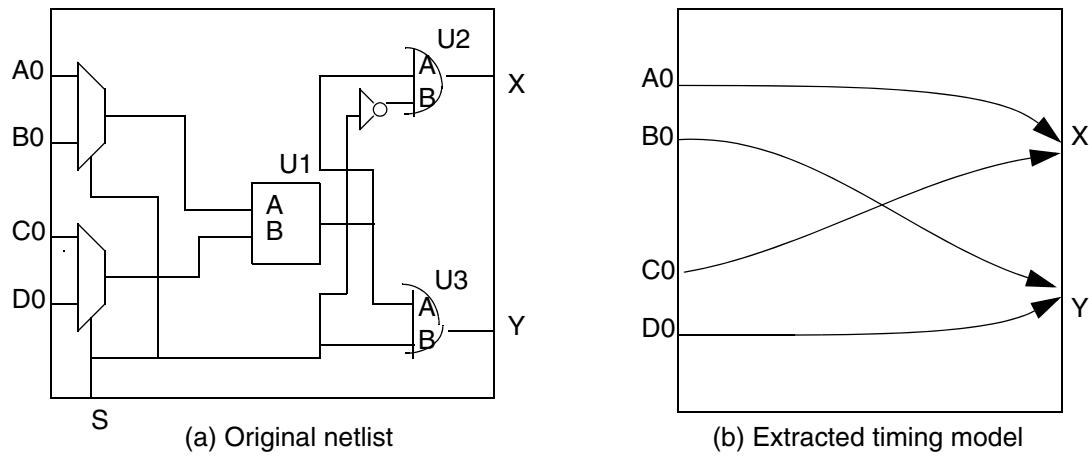
PrimeTime extracts paths with timing exceptions automatically, including exceptions you specify using the `-through` option of the exception-setting commands.

PrimeTime supports the following exceptions:

- False paths
- Multicycle paths
- Moded paths

[Figure 20-35](#) shows an original design and an extracted timing model when false paths are present.

Figure 20-35 Original Design With False Paths and Extracted Model

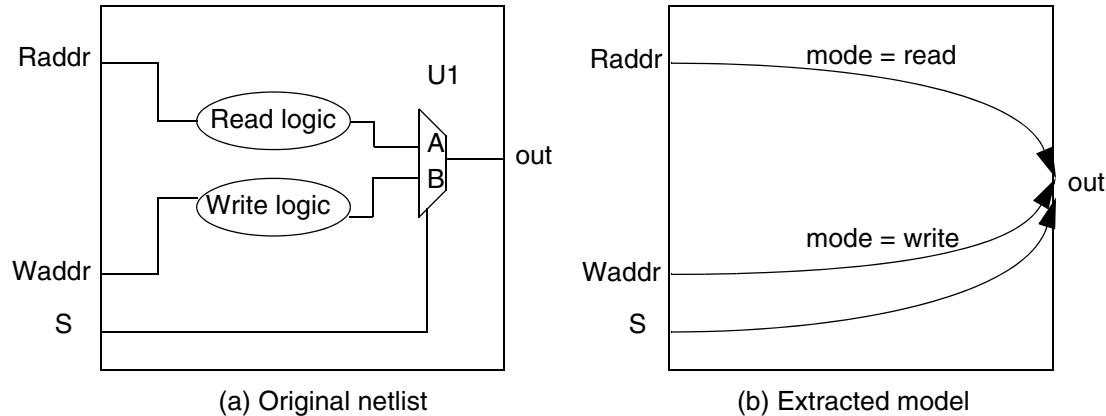


If you set the following false paths shown in (a), the extracted model represented by (b) is the result.

```
pt_shell> set_false_path -from {A0 C0} -through U3/A
pt_shell> set_false_path -from {B0 D0} -through U2/A
```

There are timing arcs from A0 and C0 to X, but none from A0 and C0 to Y. Similarly, there are timing arcs from B0 and D0 to Y, but none from B0 and D0 to X. [Figure 20-36](#) shows a design in which modes are defined with the `-through` option, together with the extracted timing model.

Figure 20-36 Extracting Modes With the -through Option



If you define the following modes for the netlist shown in (a), the result is the set of arcs represented in (b):

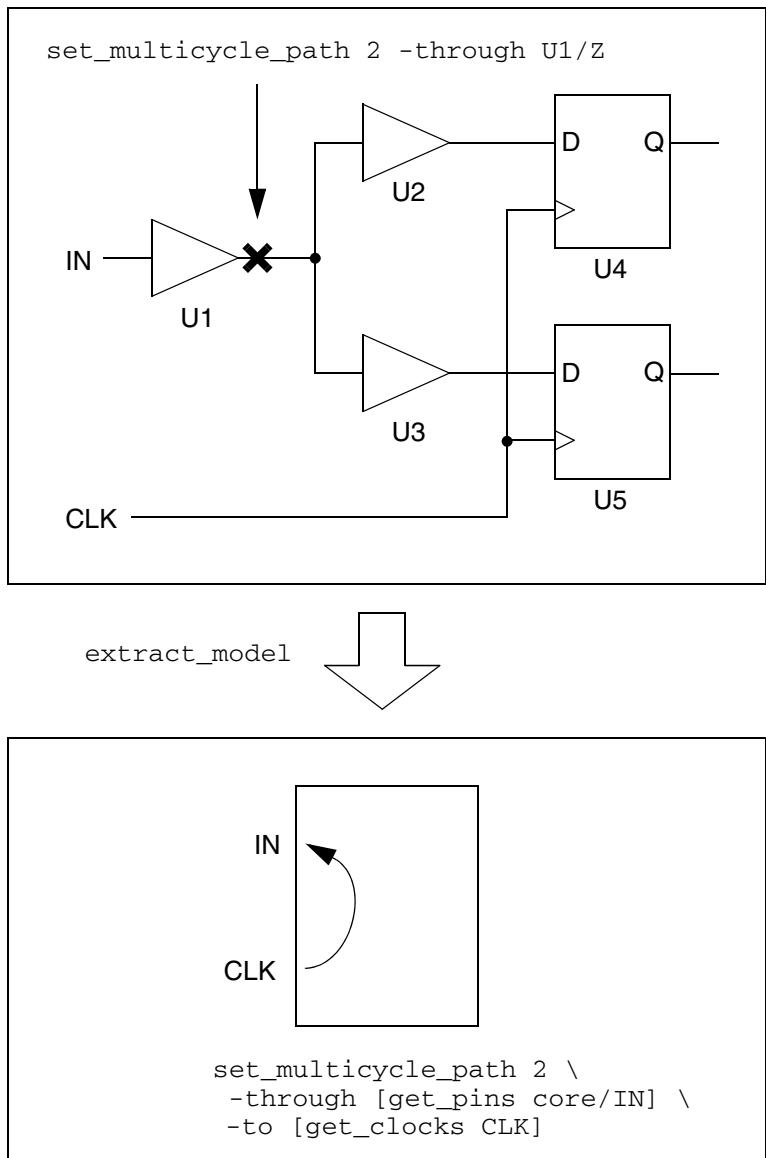
- Raddr to out in read mode
- Waddr to out in write mode
- An unmoded arc from S to out

```
pt_shell> set_mode -from Raddr -through U1/A read
pt_shell> set_mode -from Waddr -through U1/B write
```

If the original design has multicycle paths, the model extractor analyzes the multicycle paths and then writes out a script containing a set of equivalent `set_multicycle_path` commands that can be applied to the extracted model. When necessary, it also adjusts the setup, hold, and delay values for the timing arcs to correctly model the multicycle paths.

The name of the script file containing the multicycle path definitions is `output_constr.pt`, where `output` is the name specified by the `-output` option of the `extract_model` command. You need to apply this script when you use the extracted timing model. For example, [Figure 20-37](#) shows a design in which a multicycle path has been defined at a point just inside the interface logic. The model extractor creates a setup arc from CLK to IN and writes the `set_multicycle_path` command shown in [Figure 20-37](#), which must be applied to the extracted model for accurate results.

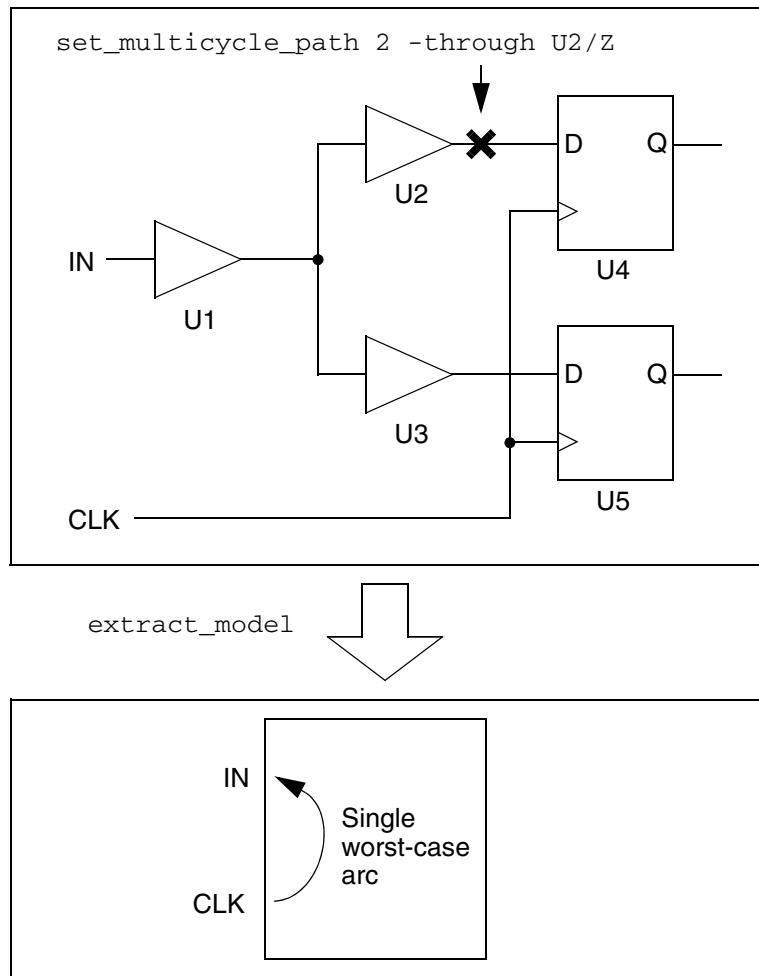
Figure 20-37 Multicycle Path Extraction With an output_mcp.pt File



A timing exception that only applies to clocks (for example, a false path between two clocks) is written to the same constraint file as the multicycle paths, *output_constr.pt*. You need to apply this script when you use the extracted timing model.

If a multicycle path has been defined that cannot be moved to the boundary of the design, the model extractor does not write out a multicycle path definition. Instead, it creates worst-case setup, hold, and delay arcs using the clock period defined at the time of model extraction, as demonstrated in [Figure 20-38](#).

Figure 20-38 Multicycle Path Extraction Without an output_mcp.pt File



Restricting the Types of Arcs Extracted

By default, the `extract_model` command extracts a full set of timing arcs for the model, including the following types: minimum sequential delay, maximum sequential delay, minimum combinational delay, maximum combinational delay, setup, hold, recovery, removal, clock gating, and pulse width arcs.

You can optionally extract only certain arc types for a model. This feature can be useful for debugging purposes when you are only interested in one type of arc, and you want to run multiple extractions under different conditions. Model extraction runs faster with this option because PrimeTime only spends time extracting the requested arcs.

To restrict the types of arcs extracted, use the `-arc_types` option of the `extract_model` command, and specify the types of arcs you want to extract. These are the allowed arc type settings:

- `min_seq_delay` – minimum-delay sequential arcs
- `max_seq_delay` – maximum-delay sequential arcs
- `min_combo_delay` – minimum-delay combinational arcs
- `max_combo_delay` – maximum-delay combinational arcs
- `setup` – setup arcs
- `hold` – hold arcs
- `recovery` – recovery arcs
- `removal` – removal arcs
- `pulse_width` – pulse width arcs

Back-Annotated Delay and Layout Information

You can back-annotate two types of information to a design in PrimeTime:

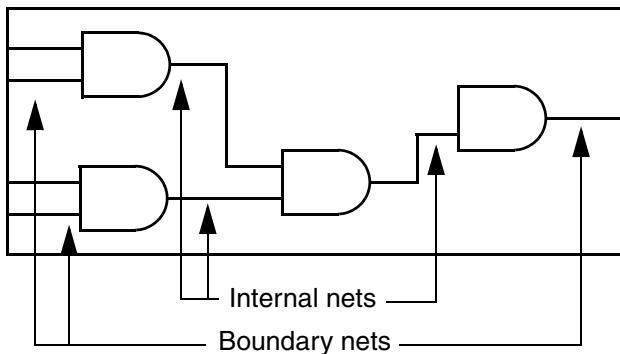
- Layout information, including lumped net capacitance or resistance and detailed RC information using standard parasitic exchange formats.
- Delay information, including net delays, cell delays, or both, using SDF. These delays are calculated for a given transition time.

In the absence of both types of information, PrimeTime calculates arc delays in the model using the cell libraries and calculates net delays using the wire load model set on the design.

Back-Annotated Delay on Nets and Cells

PrimeTime handles back-annotation of delays on internal nets and boundary nets in different ways. [Figure 20-39](#) shows the locations of boundary nets and internal nets in a simple design.

Figure 20-39 Boundary Nets and Internal Nets



Internal net delays are included in the extracted arcs of the model. Boundary net delays cannot be computed until the model is used because the delays depend on the connection of the model to the outside world, unless the `-library_cell` option is used for model extraction, in which case boundary net delays are included in the extracted timing arcs.

Internal Nets

When the internal nets are back-annotated with lumped capacitance, PrimeTime computes the net delays, cell delays, and transition times at cell outputs for model generation using this capacitance instead of the wire load models.

When SDF is back-annotated to internal nets, PrimeTime uses the back-annotated net delays during model extraction. For the transition times at cell outputs to be accurately computed, the net capacitance must also be back-annotated.

The extractor also supports extraction of designs in which internal nets are back-annotated with detailed parasitics in RSPF or SPEF.

Boundary Nets

By default, PrimeTime writes all annotated parasitics of the boundary nets to the extracted timing model, including both detailed and lumped RC information. This preserves the dependence of the net delays on the environment when the model is used. However, using the `ignore_boundary_parasitics` option of the `extract_model` command causes PrimeTime to ignore the boundary parasitics for timing model extraction, including both detailed and lumped RC information.

When the extracted model type is a library cell (created by using the `-library_cell` option of `extract_model`), PrimeTime lumps all of the boundary net capacitors onto the ports of the model. For primary input nets, it adds the wire delay resulting from using the driving cell present on the input port at the time of model extraction. For primary output ports, it calculates the wire delay by using the range of external loads characterized for the driving device.

Cells With Annotated Delays

If the cells are back-annotated with SDF delay information, PrimeTime uses this information when it extracts arc delays in the model. Because the SDF information is computed at a particular transition time, the delays of arcs in the model are accurate for this transition time. The model can be inaccurate for other transition times.

To ensure that the model is context independent, do not annotate SDF to cell delays. If you need to annotate cell delays, the extracted model is accurate for the transition time at which the SDF is generated. To create some dependence on transition time and capacitive load, you can remove the annotations on boundary cells by using the `remove_annotated_delay` and `remove_annotated_check` command before you extract the model.

Guidelines for Back-Annotation

When you perform model extraction, keep in mind the following guidelines for various types of layout and delay data back-annotation:

Internal nets

Use detailed RC data if available. This data generates the most accurate context-independent model.

- Use both SDF and lumped RC data if both are available.
- Use lumped RC data if only this data is available.
- Use SDF data alone if only this data is available. In this case, the extracted model is accurate only for the transition times at which the SDF is generated. However, using SDF data alone is not recommended.

Cell delays

To ensure that the model is context-independent, do not annotate SDF to cell delays. If you need to annotate cell delays, the extracted model is accurate only for the transition times at which the SDF is generated. To create some dependence on transition time, you can remove the annotations on boundary cells by using the `remove_annotated_delay` command and the `remove_annotated_check` command before you extract the model.

Boundary nets

If you have the detailed parasitics for the boundary nets, annotate the information before you extract.

Performing Model Extraction

After setting up your modeling environment (see [Preparing for Extraction](#)), you can extract a timing model. To learn how to do this, see

- [Loading and Linking the Design](#)
- [Preparing to Extract the Model](#)
- [Generating the Model](#)

Loading and Linking the Design

To load and link your design into PrimeTime,

1. Set the search path of your library. For example, enter

```
pt_shell> set_app_var search_path ". /abc/xyz/libraries/syn"
```

2. Set the link path of your library:

```
pt_shell> set_app_var link_path "* class.db"
```

3. Read your design into PrimeTime:

```
pt_shell> read_db design.db
```

4. Link your design:

```
pt_shell> link_design design
```

Preparing to Extract the Model

To preparing a model for extraction,

1. Define wire load models for your design:

```
pt_shell> set_wire_load_model -name wire_model_name
```

2. Define the clocks in your model. For example,

```
pt_shell> create_clock -period 10 CLK1  
pt_shell> create_clock -period 20 CLK2
```

3. Set the false paths:

```
pt_shell> set_false_path -from IN4 -to OUT1
```

4. Check your design for potential timing problems:

```
pt_shell> check_timing
```

5. Verify that the design meets timing requirements:

```
pt_shell> report_timing
```

6. Set the model extraction variables, if applicable. PrimeTime uses the defaults for any variables you do not set. For example,

```
pt_shell> set_app_var extract_model_capacitance_limit 5.0
```

Generating the Model

Determine the options you want to use for the `extract_model` command and issue the command. For example, enter the following command to extract a timing model for the current operating conditions and create a .db model file and a design in .db format:

```
pt_shell> extract_model -output example_model -format {db}
```

PrimeTime displays a report similar to this:

```
Warning: Environment variable 'extract_model_min_resolution'  
        is not defined.  
Using default '0.1'. (MEXT-6)  
Using default '0.45'. (MEXT-6)  
Wrote model library core to './example_model_lib.db'  
Wrote model to './example_model.db'
```

If you have generated a .lib version of the timing model, you might notice it contains user-defined attributes. For more information about these attributes, see [PrimeTime User-Defined Attributes in .lib](#).

PrimeTime User-Defined Attributes in .lib

User-defined attributes are used by PrimeTime and can be used by other Synopsys tools. These attributes do not affect timing analysis when using the model.

- `min_delay_flag` – Used to help `merge_models` separate minimum and maximum arcs between a pair of pins with the same sense. This attribute is used to properly merge the arcs and maintain the arc configuration across different modes.
- `original_pin` – Used to help maintain the mapping of extracted time model cell pins to the original pin names that were defined in the block netlist before extraction.

Merging Extracted Models

A module can have different operating modes, such as test mode and normal operating mode. The timing requirements for a module can be quite different for different operating modes. In these cases, you need to extract a separate model for each mode. For each extraction, place the module into the applicable mode by setting the instance or design modes, by using case analysis, and by setting the applicable timing exceptions.

To use different models extracted under different operating modes, you can swap each model into the higher-level design by using `swap_cell`. However, instead of keeping and using multiple extracted models, you can optionally merge them into a single, comprehensive model that has different timing arcs enabled for different operating modes. Then you can use this single model and change its behavior by setting it into different modes.

This is the basic procedure for creating and merging timing models for different operating modes:

1. Load and link the module netlist.
2. Back-annotate the SDF data or detailed parasitics.
3. Apply the constraints for the first operating mode.
4. Extract a model in .lib format for the operating mode.

```
pt_shell> extract_model -format lib -output model_1
```

5. Remove all constraints on the design.
6. Repeat steps 3, 4, and 5 for each additional operating mode.

7. Merge the generated .lib models.

```
pt_shell> merge_models \
    -lib_files {model_1.lib model_2.lib ...} \
    -mode_names {mode_1 mode_2 ...} \
    -group_name etm_modes \
    -output my_model \
    -formats {db lib} \
    -tolerance 0.1
```

Each `extract_model` command generates a .lib file for the extracted model. The `merge_models` command merges the extracted models into a single model that has different operating modes. You specify the operating mode when you use the model.

In the `merge_models` command, you specify the .lib files, the names of the modes corresponding to those models, the name of the new model being generated, the model formats to be generated (.db and .lib), an optional mode group name, and an optional tolerance value.

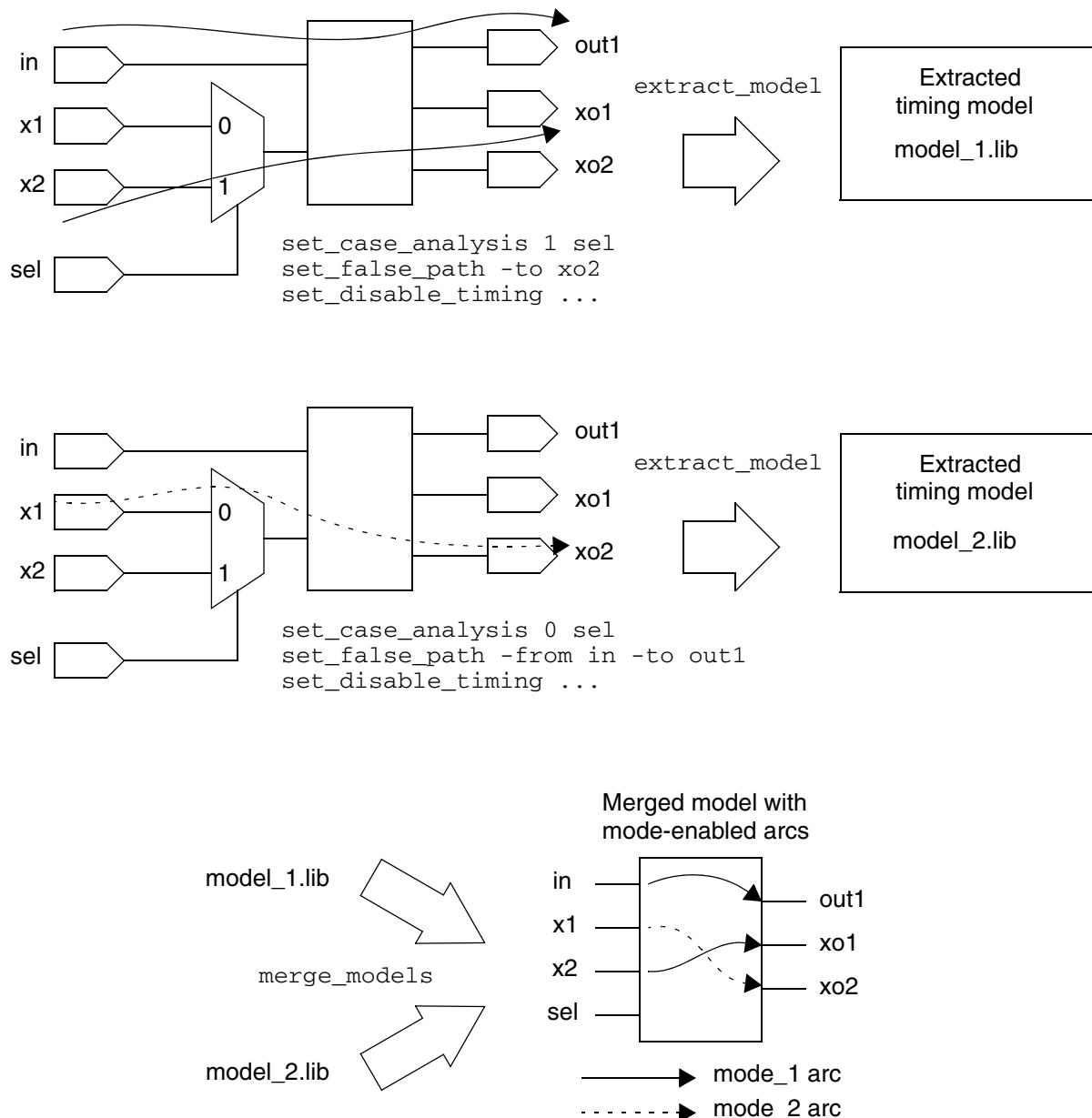
The tolerance value specifies how far apart two timing values can be to merge them into a single timing arc. If the corresponding arc delays in two timing models are within this tolerance value, the two arcs are considered the same and are merged into a single arc that applies to both operating modes. Use the `-tolerance` option of the `merge_models` command to specify the float tolerance. Ensure that the tolerance value is greater than or equal to zero. The default tolerance value is 0.04 time units. The values from the first .lib file listed using the `-lib_files` option is used in the merged model.

PrimeTime can handle more than one mode per timing arc, but some tools cannot. To generate a merged model that can work with these tools, use the `-single_mode` option of the `merge_models` command. This forces the merged model to have no more than one mode per timing arc, resulting in a larger, less compact timing model.

It might be necessary to disable all arc merging. For instance, if you are performing two independent merges and the resulting merged models must have the same number of arcs. This occurs when one merged model has the maximum and another has the minimum operating condition. To use these two models in the `set_min_library` command, they must have exactly the same number of arcs. To disable all arc merging, you can use the `-keep_all_arcs` option of the `merge_models` command.

The models being merged must be consistent, with the same I/O pins, operating conditions (process/voltage/temperature), and so on. Timing arcs that are different are assigned to the modes specified in the `merge_models` command. Design rule constraints, such as minimum and maximum capacitance and transition time, are allowed to be different. In those cases, the more restrictive value is retained in the merged model.

[Figure 20-40](#) shows an example of a module from which two timing models are extracted, where the extracted models are merged into a single timing model having two operating modes.

Figure 20-40 Model Extraction and Merging Example

To use a merged timing model with modes:

1. Set the `link_path` variable to include the path to the merged model in .db format.
2. Load and link the top-level design.
3. Apply the constraints and back-annotation on the design.

4. Using the `set_mode` command, set the mode on the module instance to the desired mode.
5. Run the timing analysis.
6. Repeat steps 4 and 5 for each mode that you want to analyze.

Here are some points to consider for model merging:

- To retain in the merged model, case values propagated to the output pins of the models being merged must be the same in all models. If there are mismatching case values, PrimeTime ignores them and issues a warning message.
- Any model that already has modes or moded arcs defined in it cannot be merged.
- Any generated clocks in the models to be merged must be exactly the same.

Extracting Internal Pins

Internal pins are pins that are not defined as ports of the model. In the .db representation of the extracted model, these pins are defined as internal pins of the model core cell. The `extract_model` command creates several types of internal pins.

To learn about extracting internal pins, see

- [Clocks on Internal Pins of a Design](#)
- [Generated Clocks](#)
- [Check Pins for Clock Networks](#)
- [Check Pins for Bidirectional Ports](#)

Clocks on Internal Pins of a Design

When a clock is defined on an internal pin of the design (not on a port), the `extract_model` command stores this timing information by creating an internal pin for each source pin of the clock. The name assigned to the internal pin is just the clock name. If the clock name conflicts with a port name, PrimeTime appends _1, _2, or something similar to make the internal pin name.

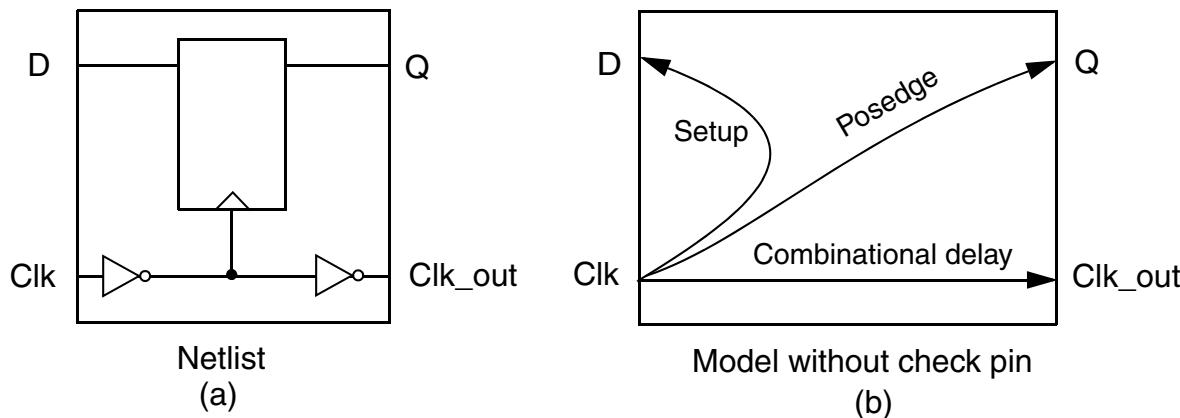
Generated Clocks

When a generated clock is defined with a generated clock source on an internal pin, the `extract_model` command stores this timing information by creating an internal pin for the source pin of the generated clock. The name assigned to the internal pin is just the generated clock name. If the generated clock name conflicts with a port name, PrimeTime appends _1, _2, or something similar to make the internal pin name.

Check Pins for Clock Networks

If the design has combinational paths in the clock network, then the extracted model has a combinational arc from the clock input to the clock output, along with setup and hold arcs from the same clock to latched inputs. [Figure 20-41](#) shows what the extracted model would look like in the absence of check pins.

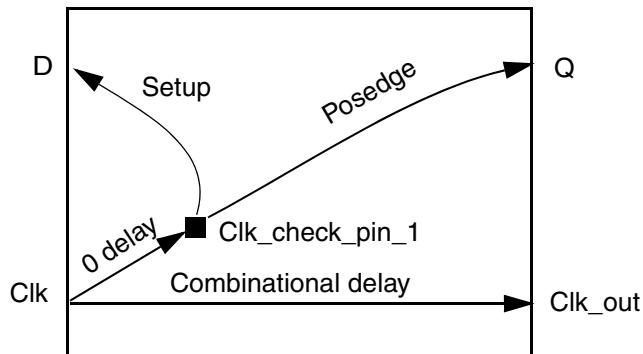
Figure 20-41 Original Circuit and Extracted Model Without Check Pins



If you were to use this model in PrimeTime, the combinational delay path from the clock would not be traversed because clock tracing stops when PrimeTime detects a setup or hold arc from the clock pin.

If a clock pin or data pin of a setup constraint also has a combinational delay path originating from it, PrimeTime creates a check pin to allow clock tracing to continue to the clock output. For example, to trace the delay path from the clock pin in [Figure 20-41](#), the model extractor creates an extra internal pin in the extracted model as shown in [Figure 20-42](#). The setup and hold arc between the clock pin and the input is changed to be between the check pin and the input pin. This causes clock tracing to stop at the check pin, while allowing clock tracing to continue from the real clock input pin to the clock output.

Figure 20-42 Model With Check Pin



For the first constraint needing to be moved, PrimeTime names the check pin as follows:

transformed_pin_name__check_pin_1

For the second constraint needing to be moved on the same transformed pin, PrimeTime names the check pin as follows:

transformed_pin_name__check_pin_2

In this example, the transformed pin name is the clock port name. In other cases, it is the data pin name for the data pin of a setup constraint that also has a combinational delay path from the pin.

The check pins can only be seen in the .db format by extracting directly to the .db format.

Check Pins for Bidirectional Ports

If a model has one or more combinational arcs to an INOUT (bidirectional) pin and one or more constraint arcs to that same INOUT pin, the .db writer of `extract_model` creates an internal check pin to handle different paths through the bidirectional pin.

The .db writer alters the arcs as follows:

- It creates a zero-delay arc from the INOUT pin to the check pin.
- It moves the constraint arcs that formerly ended at the INOUT pin so that they end at the check pin.

Model Validation

When you create an interface logic model (ILM) or extracted timing model (ETM), it is recommended that you validate the timing characteristics of the new model against the original gate-level netlist. To accomplish this task, use the `write_interface_timing` and `compare_interface_timing` commands.

To learn about model validation, see

- [Model Validation Overview](#)
 - [Viewing the write_interface_timing Report](#)
 - [Viewing the compare_interface_timing Report](#)
 - [Comparison Tolerance](#)
 - [Debugging Timing Arc Comparison Failures](#)
 - [Debugging Other Comparison Failures](#)
-

Model Validation Overview

PrimeTime can create two kinds of timing models from a gate-level netlist: an ILM or ETM. An ILM is a structural model that maintains the interface logic at the input and output ports of the block, as explained in [Interface Logic Models](#). An ETM is a purely behavior model, consisting of a set of timing arcs between the ports of the block, as explained in [Extracted Timing Models](#).

Automatic Model Validation

When you create an interface logic model (ILM) or extracted timing model (ETM), validate the timing characteristics of the new model against the original gate-level netlist. During validation, you might see timing mismatches between the model and netlist. These failures could occur because of setup issues, pessimism in graph-based analysis, or limitations in modeling.

PrimeTime has unified the validation process to a single flow for both ETMs and ILMs. It also provides an automatic model validation feature that allows you to avoid setup errors and to resolve failures using path-based analysis. With automatic model validation, most of the common model validation failures are now resolved. If a mismatch does occur, you are provided with the reason that it occurred.

To use the automatic model validation feature,

1. Specify the current modeling technique and enable automatic model validation by setting the following variable:

```
pt_shell> set_app_var hier_modeling_version 2.0
```

2. Customize the automatic model validation process by optionally setting the following variables:

- pt_model_dir – Specifies a directory for PrimeTime to save model related files. The default is the current directory.

- ILM files are saved in a subdirectory of the \$pt_model_dir/ILM directory. For example:

```
pt_shell> set_app_var pt_model_dir ./xyz  
pt_shell> set design_name [get_object_name [current_design]]
```

The ILM files are saved in the ./xyz/ILM/\$design_name directory.

- ETM files are saved in a subdirectory of the \$pt_model_dir/ETM directory. For example:

```
pt_shell> set_app_var pt_model_dir ./xyz  
pt_shell> extract_model -output blk_etm
```

The ETM files are saved in the ./xyz/ETM/blk_etm directory.

- model_validation_output_file – Specifies the output file that summarizes the results of the verification. By default, the results of the verification are written to the verif.out and verif.detail.gz files of the model validation subdirectory under the \$pt_model_dir directory.
- model_validation_reanalyze_max_paths – Specifies the maximum number of failed paths that PrimeTime analyzes again. The default is 1000.
- model_validation_report_split – Enables line-splitting in the output report. This is useful for comparing scripts or for postprocessing the script. The default is true.
- Variables to specify tolerance:
 - model_validation_timing_tolerance – Specifies the absolute error tolerance for time data during model validation. The default is 0.01 units.
 - model_validation_capacitance_tolerance – Specifies the absolute error tolerance for capacitance data. The default is 0.
 - model_validation_percent_tolerance – Specifies the percent relative error tolerance for timing data during model validation. The default is 0.

- `model_validation_ignore_pass` variable – Displays only failed comparisons if set to `true` (the default).
 - `model_validation_section` – Specifies a list of data sections to include in the comparison.
 - `model_validation_sort_by_worst` – Sorts the output from worst to best, with the worst failure first, if set to `true` (the default). After the FAIL section, all PASS lines are sorted alphabetically within each path attribute.
 - `model_validation_significant_digits` – Specifies the number of digits to the right of the decimal point that is reported in the validation results. The allowed values are 0–13, and the default is 2.
 - `model_validation_verbose` – Shows detailed debugging of mismatched paths If set to `true`.
3. Specify the host that is used by the automatic model validation by setting the `host_names` option of the `set_host_options` command.
 4. Use the `create_ilm` or `extract_model` command with the `-validate` option to specify what PrimeTime automatically validates after the ILM or ETM is created.

Note:

If you do not set the `hier_modeling_version` variable to 2.0, the `-validate` option is not available in these commands, and you receive an error message explaining that you are using an old modeling version.

When you use the `create_ilm` or `extract_model` command with the `-validate` option, the following tasks occur automatically:

- a. A separate validation process is started to compare the interface timing results between the netlist and the model.
- b. The validation process reads the Verilog, SDC, and parasitic files generated during the model creation process. The interface timing results are passed on to the main process.
- c. The main process compares the interface timing between the netlist and the model. The comparison results are printed if no failures occur. Otherwise, the following actions occur, based on the model type:
 - For ILMs, as part of the main process, the tool sends the startpoint and endpoints of failed paths back to the validation process. In the validation process, the tool performs path-based analysis on the paths between these points and then sends the updated interface timing results back to the main process. Simultaneously and as part of the main process, the tool performs path-based analysis on the failed paths and also compares the interface timing results between the netlist and the model.

- For ETMs, as part of the main process, the tool finds the worst paths in the netlist and updates the interface timing results of the netlist after path-based analysis. Simultaneously, the main process compares the updated timing results of the netlist with the timing results of the model, and it prints the comparison results.
- d. If the `model_validation_verbose` variable is set to `true` (the default), PrimeTime performs a detailed comparison for each failed path.
- e. The validation process, used by PrimeTime to perform the complex comparison and generated detailed reports, is terminated.

Note:

If you are in the multicore analysis mode and use either the `create_ilm` command to create an ILM for the current block-level netlist or the `extract_model` command to automatically generate a timing model from a gate-level netlist, the session is automatically triggered to switch to single-core analysis mode.

5. If interactive debugging is needed, you can launch a separated PrimeTime session and use the generated `validation.pt.gz` file in the model validation subdirectory to interactively see what is wrong in the model.

Manual Model Validation Using Commands

You can validate a new ILM or ETM against the original gate-level netlist, or compare any two valid timing models in PrimeTime, by using the `write_interface_timing` and `compare_interface_timing` commands.

The `write_interface_timing` command reports the interface timing of a specified netlist or model. This report includes the worst-case slacks or timing arc values for various types of paths, the transition times at the output ports, the lumped and extracted capacitance on all ports, and the design rules on the ports. It considers the input, output, and combinational paths, but not register-to-register paths. Static noise response characteristics can be included as well, if desired, by using the `-include {timing noise}` option. To generate only noise information in the report, use the `-include {noise}` option. If you have not specified the `-include` option, only timing information is written.

The `compare_interface_timing` command compares two reports generated by the `write_interface_timing` command. It lets you specify the reference file, the comparison file, the types of paths and timing parameters to compare or not compare, and the allowed tolerance levels that trigger comparison failures. If the timing parameter values in the two files are the same or within the specified tolerance, you receive a pass result. Otherwise, you receive a fail result.

To generate a new timing model and validate it against the original gate-level netlist, use the following procedure:

1. Read and link the original gate-level netlist and apply the applicable timing constraints and environment.

2. Run the `check_timing` command to make sure all paths are constrained. Otherwise, some paths might not be checked properly because their slack cannot be analyzed.

3. Write the interface timing data for the netlist design using the `write_interface_timing` command. For example,

```
pt_shell> write_interface_timing net.rpt
```

4. Generate the extracted timing model or interface logic model.

5. Remove the netlist design.

6. Read and link the model extracted in step 4, and apply the same environment as in step 1.

7. Write the interface timing data for the model using the `write_interface_timing` command. For example,

```
pt_shell> write_interface_timing model.rpt -timing_type slack
```

Note:

Instead of comparing slack values, you can compare timing arc values (setup, hold, and delay) by using the `-timing_type arc` option of the `write_interface_timing` command.

8. Compare the interface timing reports using the `compare_interface_timing` command. For example,

```
pt_shell> compare_interface_timing net.rpt model.rpt \
           -absolute_tolerance 0.1 -output compare.rpt
```

The return code reported by PrimeTime is 0 if the two files match (pass) or 1 if there was any difference (fail). Any other return code or no return code indicates a command error. The `-absolute_tolerance` option sets the tolerance thresholds for comparison failure.

9. Examine the comparison report. If necessary, debug the cause of any comparison failures. Use the `report_etm_arc` command to debug timing differences or the `report_port` command to debug transition time, capacitance, or design rule differences.

Slew Propagation

PrimeTime propagates the worst slew selected from the inputs. This can lead to differences between the extracted model and the original netlist. To reduce these differences, set the `extract_model_use_conservative_current_slew` variable to `true`. In that case, the model extractor adjusts the timing arcs of the model to more closely resemble the netlist behavior. The result is a more pessimistic model that is more likely to pass validation.

Viewing the write_interface_timing Report

The `write_interface_timing` command reports the interface timing of a specified netlist or model. The report contains the following major sections:

- Slack or Arc Value – This section reports the worst-case slack or arc value for each path from input port to clock, from clock to output port, and from input port to output port.
- Transition Time – This section reports the actual transition time at each port for the four delay types: `min_fall`, `min_rise`, `max_fall`, and `max_rise`.
- Capacitance – This section reports the maximum total (lumped) capacitance at each port, and if available, the effective capacitance.
- Design Rules – This section reports all design rules that apply to each port, including maximum capacitance, minimum capacitance, maximum transition time, maximum fanout (for input ports), and fanout load (for output ports).
- Noise Detection – This section reports the noise slack at the inputs.
- Noise Calculation – This section reports the steady-state I-V characteristics of the outputs.

The following example is a typical interface timing report showing arc values:

```
pt-shell> write_interface_timing demo.rpt \
           -timing_type arc
1
pt-shell> sh cat demo.rpt
*****
Command: write_interface_timing
          demo.rpt
          -timing_type arc
Design : top
...
*****
Section: arc_values
Info   : Worst-case arc values for each port and path group
Design : top
*****
Generated Clock and Source Info:

Attribute:
  L<n> - latch level where <n> is 0 for first level
```

From	To	Arc Type	Arc Value		
<hr/>					
in(r)	out(r)	max_combo_delay	17.35		
in(f)	out(r)	max_combo_delay	17.29		
in(r)	out(f)	min_combo_delay	17.35		
in(f)	out(f)	min_combo_delay	17.29		
in(r)	clk(r)	setup	2.63 L1		
in(f)	clk(r)	setup	2.29 L2		
in(r)	clk(f)	hold	0.17		
in(f)	clk(f)	hold	0.51		
clk(r)	out(r)	max_seq_delay	17.79		
clk(r)	out(f)	max_seq_delay	18.20		
clk(f)	out(r)	min_seq_delay	17.79		
clk(f)	out(f)	min_seq_delay	18.20		
<hr/>					
Section: transition_time					
Info : Actual transition time on each port					
Design : top					
<hr/>					
Port	MAX_RISE	MAX_FALL	MIN_RISE	MIN_FALL	
<hr/>					
in	0.00	0.00	0.00	0.00	
clk	0.00	0.00	0.00	0.00	
out	0.06	0.02	0.06	0.02	
<hr/>					
Section: capacitance					
Info : The total and effective capacitance on each port					
Design : top					
<hr/>					
Max					
Port	Ctot	Ceff			
<hr/>					
in	1.39	--			
clk	1.39	--			
out	0.39	--			
<hr/>					
Section: design_rules					
Info : Design rules on each port					
Design : top					
<hr/>					
Max Port	Min Cap	Max Cap	Max Trans	Fanout	Fanout Load
<hr/>					
in	--	--	--	--	n/a
clk	--	--	--	--	n/a
out	--	--	--	n/a	0.00

Arc Types

In the slack or arc value section of the interface timing report, each reported worst-case value has an associated arc type:

- min_seq_delay – Minimum delay arc from clock to output port
- max_seq_delay – Maximum delay arc from clock to output port
- min_combo_delay – Minimum delay arc from input port to output port
- max_combo_delay – Maximum delay arc from input port to output port
- setup – Setup arc from input port to clock
- hold – Hold arc from input port to clock
- recovery – Recovery arc from input port to clock
- removal – Removal arc from input port to clock
- clock_gating_setup – Clock-gating setup arc from input port to clock
- clock_gating_hold – Clock-gating hold arc from input port to clock

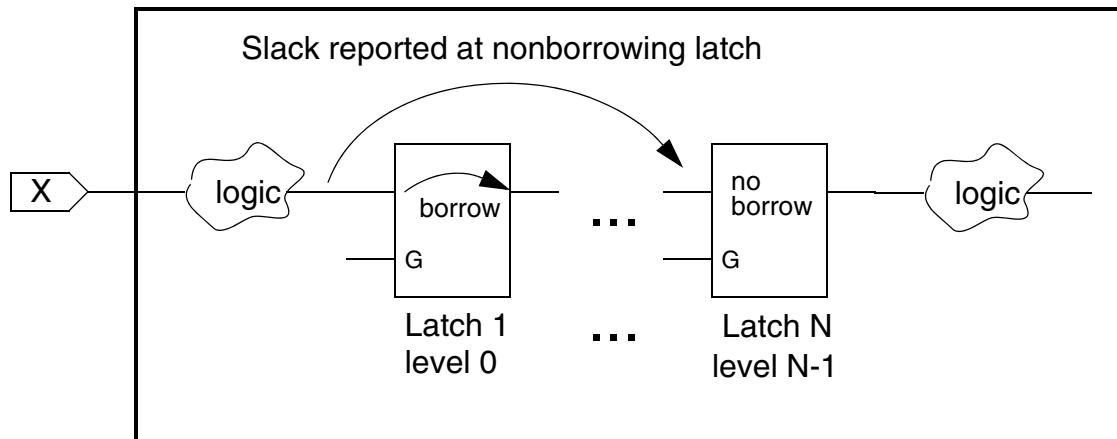
When you use the `compare_interface_timing` command to compare two interface timing reports, you can selectively include only the reports associated with specific arc types. The arc type can also be important when you want to verify a comparison failure with the `report_timing` command. For more information, see [Debugging Other Comparison Failures](#).

Sometimes the notation `Ln` appears after the slack or arc value in the interface timing report. When present, it indicates the level at which borrowing stopped in a chain of transparent latches in the path. The number n indicates the latch level number, starting with zero for the first latch. For example, the notation `L2` means borrowing stopped at the third latch encountered in the path.

Time Borrowing by Transparent Latches

The `write_interface_timing` command, like other PrimeTime analysis tools, recognizes that level-sensitive latches can borrow time in a path. The command traces through borrowing latches at the interface of a design until it encounters a flip-flop, port, or nonborrowing latch. In the latter case, the slack is reported at the nonborrowing latch, as shown in [Figure 20-43](#).

Figure 20-43 Path Tracing Stopped at a Nonborrowing Latch



You can use the `-latch_level` option of the `write_interface_timing` command to specify the number of levels of latch borrowing that are to occur at the interface of a design. Using this option is recommended only when you are certain of the borrowing behavior of latches at the interface of the design and you are able to specify a single latch chain length for the entire design. The `write_interface_timing` command reports the level of latch borrowing as an attribute, starting with zero for the first latch in the path. If the first latch encountered is nonborrowing, it is reported as level zero. If the design contains latch-borrowing feedback loops, PrimeTime prevents borrowing at the startpoint of the loop.

When PrimeTime creates an extracted timing model or interface logic model, it assumes that the netlist being modeled has already had its internal timing verified. Therefore, the `write_interface_timing` command only considers the borrowing behavior of latches on the interface timing paths. Note that borrowing can only occur on a maximum-delay (not a minimum-delay) path.

Viewing the `compare_interface_timing` Report

The `compare_interface_timing` command compares two reports generated by the `write_interface_timing` command. You specify the two reports to compare, the types of paths and timing parameters to compare or not compare, and the allowed tolerance levels that trigger comparison failures. The resulting report shows the comparison results for individual paths, ports, and timing parameters. You can have the report sent to the screen or written to a file.

By default, the `compare_interface_timing` command compares all of the parameters contained in the timing interface reports. The resulting comparison report has the same sections as the interface timing report: slack or arc value, transition time, capacitance, design rules, and noise. You can restrict the scope of the report by using the `-include` option.

The following example is a typical comparison report. For each path, the report shows the path type, the value in the reference file, the value in the comparison file, the difference, and pass/failure status for that individual comparison.

```
pt-shell> compare_interface_timing demo_net.rpt demo_etm.rpt \
           -include arc_values -percent_tolerance {10 5}

*****
Command: compare_interface_timing
          demo_net.rpt demo_etm.rpt
                  -include arc
          -percent_tolerance 10.0 5.0
Design : top
...
*****

      Arc          Arc Value
From   To    Type       Ref      Cmp  %Error  Status
-----+
in(r)  out(r) max_combo_delay 2.63    2.21   15.97  FAIL
in(f)  out(r) max_combo_delay 2.28    2.32   -1.75  PASS
in(r)  out(f) min_combo_delay 0.17    0.30   -76.47  FAIL
in(f)  out(f) min_combo_delay 0.51    0.51   0.00   PASS
in(r)  clk(r) setup          2.63 (L1) 2.63   0.00   PASS
in(f)  clk(r) setup          2.29 (L2) 2.29   0.00   PASS
in(r)  clk(f) hold          0.1     0.17   0.00   PASS
in(f)  clk(f) hold          0.51(f) 0.51(r) 0.00   PASS
clk(r) out(r) max_seq_delay 17.79   17.79   0.00   PASS
clk(r) out(f) max_seq_delay 18.20   18.20   0.00   PASS
clk(f) out(r) min_seq_delay 2.21    2.21   0.00   PASS
clk(f) out(f) min_seq_delay 1.80    1.80   0.00   PASS

      Transition
      Totals  Arc Value Time      Capacitance  Rules
-----+
Passed   10     10    0        0
Failed    2      2     0        0
Total    12     12    0        0
```

If all comparisons in the report are “pass,” the return code for the `compare_interface_timing` command is 0. If one or more comparisons result in “fail,” the return code is 1. Any other return code or no return code indicates a command error.

The L# notation indicates the latch level number. This same notation appears in the right column of a `write_interface_timing` report when a path involves a transparent latch.

Comparison Tolerance

In the `compare_interface_timing` command, you can optionally specify tolerance values for the slack, arc value, transition time, capacitance, and noise comparisons. If the two values being compared are within the specified tolerance, it is considered “pass.” If you do not specify a tolerance, the default is zero, which means that any amount of difference triggers a comparison failure.

There are four tolerance settings, called the absolute, percentage, capacitance, and noise tolerance settings:

- The absolute tolerance setting specifies the absolute amount of difference for slack, arc value, and transition time comparisons, in library time units such as nanoseconds.
- The percentage tolerance setting specifies the percentage amount of difference for arc value, transition time, and capacitance comparisons. A setting of 1.0 means a difference of 1 percent.
- The capacitance setting specifies the absolute amount of difference for capacitance comparisons, in library capacitance units such as picofarads.
- The noise setting specifies the absolute amount of difference for noise slack comparisons, in library voltage units times library time units, such as volt-nanoseconds.

None of these settings apply to design rule comparisons. Design rules must match exactly to pass a comparison test.

Arc value and transition time comparisons can accept both absolute and percentage tolerance settings. In that case, both types of comparisons are done, and both types of comparisons must fail to trigger a comparison failure report. Similarly, capacitance comparisons can accept both percentage and absolute capacitance tolerance settings; both comparisons must fail to trigger a comparison failure report.

An example of a `compare_interface_timing` command that specifies both absolute and percentage tolerances is as follows:

```
pt_shell> compare_interface_timing net.rpt model.rpt \
           -absolute_tolerance {0.1 0.2} \
           -percent_tolerance 1.0 \
           -output compare.rpt
```

The command is a request to compare the two reports `net.rpt` (the reference file) and `model.rpt` (the comparison file). The `-output` option causes the comparison report to be written to a file rather than displayed in the transcript.

Absolute Tolerance

In the preceding example, two absolute tolerance values are specified, 0.1 and 0.2 time units. This means that the result of subtracting the comparison value from the reference value must be between -0.1 and +0.2 to pass the comparison test. For example, if the slack in the reference file is 5.0 time units, the slack in the comparison file must be less than 5.1 and more than 4.8 time units for the comparison to pass.

If you specify only one absolute tolerance value of the command, that same value applies to both the positive and negative directions. If no tolerance value is specified, the values must match exactly to pass.

The `-capacitance_tolerance` setting is an absolute tolerance in library capacitance units. It works in the same manner as the `-absolute_tolerance` setting, except that it applies to capacitance comparisons rather than slack, arc value, and transition time comparisons.

Similarly, the `-noise_tolerance` setting is an absolute tolerance in library voltage-time units. It works in the same manner as the `-absolute_tolerance` setting, except that it applies to noise slack comparisons.

Percentage Tolerance

In the preceding example, the percentage tolerance is set to a single number, 1.0, which represents plus or minus 1.0 percent of the arc value, transition time, or capacitance value in the reference file. If the value in the comparison file is outside of this range, the result is a comparison failure. For example, if the arc value in the reference file is 10.0, the arc value in the comparison file must be between 9.9 and 10.1 for the comparison to pass.

The percentage tolerance setting does not apply to slack comparisons. Only the absolute tolerance setting applies to slack.

Debugging Timing Arc Comparison Failures

If the `compare_interface_timing` command reports a comparison failure, the reason for the failure might not be obvious. To find out more about the timing arcs related to the failure, use the `report_etm_arc` command. In this command, you specify the startpoint and endpoint of the arc of interest, as indicated in the comparison report. You also specify the conditions for model extraction, just as you do in the `extract_model` command.

The `report_etm_arc` command generates an extracted timing model (ETM) report. This report shows the details of the data path used by the `extract_model` command to generate the extracted timing arc, including the capacitance, transition time, and delay values calculated at intermediate points along the path. You can optionally report the clock path in addition to the data path.

Another option is to report the critical path between the same two endpoints, as determined by the `report_timing` command operating on the original netlist. This is called a netlist report.

By comparing the ETM report and the netlist report at each point along the path, you can determine the point of divergence and the likely cause of the comparison failure. You might be able to fix this problem by editing or deleting the arc in the extracted model or by extracting a new model under different conditions. In some cases, you might decide that the extracted model is accurate enough, and you can use the model without further changes.

Validation Flow With Timing Arc Debugging

To generate, compare, and debug an extracted timing model, you can use a procedure similar to the following example.

This example uses two pt_shell sessions running at the same time: one to analyze the netlist and the other to analyze the model. If you have only one PrimeTime license available, you can perform the same tasks by removing and reloading the netlist and model each time for analysis.

1. In the first pt_shell window, read and link the original gate-level netlist and apply the applicable timing constraints and environment. Run the `check_timing` command to make sure all paths are constrained.

2. Extract the timing model using the `extract_model` command. For example,

```
pt_shell> extract_model -output etm
```

3. Make sure that the slew propagation mode is set properly for model validation and debugging (see [Slew Propagation](#)).

4. Write the interface timing data for the netlist using the `write_interface_timing` command. For example,

```
pt_shell> write_interface_timing net.rpt
```

5. In the second pt_shell window, load the extracted model and apply the applicable timing constraints and environment. Set the slew propagation mode (if applicable), and report the interface timing of the model using the `write_interface_timing` command. For example,

```
pt_shell> write_interface_timing model.rpt
```

6. Examine the generated report. For each comparison failure, use the `report_etm_arc` command to report the failing arc. Use the first pt_shell window, where the netlist is

loaded (or reload the netlist first). For example, if the arc type shown in the comparison report is max_seq_delay, use a debugging command in the following form:

```
pt_shell> report_etm_arc -from clock -to port \
    -arc_type max_seq_delay \
    -include {clock_path netlist_path}
```

The `report_etm_arc` command generates an ETM report and a netlist report for the specified arc.

In the `report_etm_arc` command, you must use the same options as in the original `extract_model` command, such as `-library_cell` or `-context_borrow`.

The `-from` and `-to` options specify the startpoint and endpoint of the arc to be reported. Depending on the arc type, the startpoint and endpoint each might be either a clock or a port. Instead of using the `-from` option, you can use either the `-rise_from` or `-fall_from` option to restrict the report to only rising edges or only falling edges at the startpoint of the arc. Similarly, instead of using the `-to` option, you can use either the `-rise_to` or `-fall_to` option.

The `-include` option specifies the types of paths you want included in the report. By default, only the netlist path is reported.

Here is another example. Suppose that the failure report looks like this:

	Arc					
From	To	Type	Ref	Cmp	%Error	Status
in(r)	clk(r)	setup	2.63 (L1)	2.53	3.80	FAIL

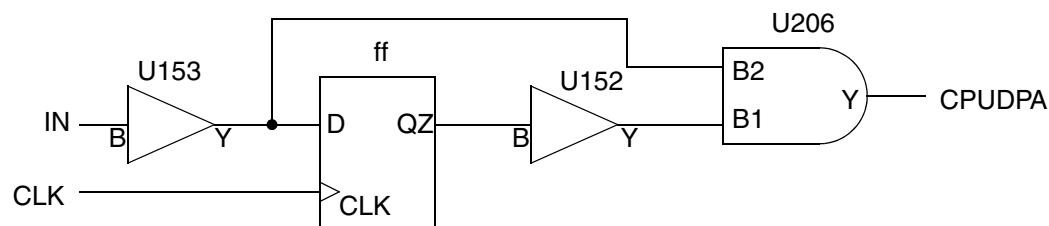
For this type of failure, use this command:

```
pt_shell> report_etm_arc -rise_from in -rise_to clk \
    -arc_type setup \
    -include {clock_path netlist_path}
```

Debugging Report Example

Consider the circuit shown in [Figure 20-44](#).

Figure 20-44 Circuit Used to Generate an Extracted Timing Model



Suppose that you generate an extracted timing model from this design. You generate an interface timing report for the original design and for the extracted model, and then generate a comparison report, which shows a comparison failure for one arc. You use the following commands to define the clock and set the input transition time:

```
pt_shell> create_clock -name CLKr -period 6.0 \
           -waveform {0.0 3.0} [get_ports {CLK}]
pt_shell> set_input_transition 0.700000 CLK
```

You generate a debugging report on a min_combo_delay arc from a falling edge on input IN to a falling edge on output CPUDPA:

```
pt_shell> report_etm_arc -fall_from IN -fall_to CPUDPA \
           -arc_type min_combo_delay -library_cell \
           -include {clock_path netlist_path}
```

ETM report for arc : IN_CPUDPA_min				
Point	Cap	Trans	Incr	Path
IN (in)	3.6344	0.0000	0.0000	0.0000 f
U153/B (NA220)		0.0000	0.0042	0.0042 f
U153/Y (NA220)	3.8579	0.6339	0.2897	0.2939 r
U206/B2 (BF056)		0.6339	0.0013	0.2952 r
U206/Y (BF056)	121.6144	11.3933	8.3614	8.6566 f
CPUDPA (out)		11.3933	0.0000	8.6566 f

Netlist path:				
Point	Cap	Trans	Incr	Path
IN (in) <-	3.6344	0.0000	0.0000	0.0000 f
U153/B (NA220)		0.0000	0.0042	0.0042 f
U153/Y (NA220)	3.8579	0.6339	0.2897	0.2939 r
U206/B2 (BF056)		0.6339	0.0013	0.2952 r
U206/Y (BF056)	61.6144	11.3933	8.0973	8.3925 f
CPUDPA (out) <-		11.3933	0.0711	8.4636 f

The ETM and netlist reports diverge at U206/Y because of different capacitance values defined on the CPUDPA output port. This was the cause of the comparison failure.

Causes of Validation Failure

An extracted model might have slack or delay values different from those reported by the `report_timing` command operating on the original netlist, possibly because of differences in interpreting timing exceptions or differences in handling slew propagation. For more information, see [Slew Propagation](#).

The model extractor does not always interpret timing exceptions the same as the `report_timing` command. For example, it does not support “rise” or “fall” type exceptions. This can cause the extracted model to use a false path. In that case, `report_etm_arc`

command shows the different paths taken in the extracted model and in the netlist. You might be able to fix this condition by changing the constraint (for example, from the `set_false_path` to `set_disable_timing` commands) and running model extraction again.

Debugging Other Comparison Failures

The `compare_interface_timing` command can report comparison failures not related to timing arcs, such as a difference in transition time, output capacitance, or design rule parameter. You might want to manually examine the information in the comparison report or the corresponding lines in the interface timing report to determine the causes of these differences.

Transition Time

To verify the transition times reported in the Transition Time section of the `compare_interface_timing` report, use this command:

```
pt_shell> get_attribute [get_port port_name] attribute
```

where *port_name* is the name of the port and *attribute* is one of the following delay types:

```
actual_rise_transition_min  
actual_rise_transition_max  
actual_fall_transition_min  
actual_fall_transition_max
```

For example,

```
pt_shell> get_attribute [get_port out1] actual_rise_transition_max
```

Capacitance

To verify the total or effective capacitance of a port, use one of the following commands:

```
pt_shell> get_attribute [get_net port_name] total_capacitance_max
```

```
pt_shell> get_attribute [get_port port_name] effective_capacitance_max
```

Design Rules

To verify the design rules that apply to a port, use this command:

```
pt_shell> report_port -design_rule port_name
```

Hierarchical Scope Checking

When you perform hierarchical analysis using interface logic models or extracted timing models, you can have PrimeTime check the “scope” of each timing model to verify that the analysis conditions at the top-level are within the ranges used for checking the block at the time of model creation.

To learn more about hierarchical scope checking, see

- [Scope Checking Overview](#)
 - [Types of Block Scope Checking](#)
 - [Block Scope Files](#)
 - [Checking the Block Scope at the Chip Level](#)
-

Scope Checking Overview

Interface logic models and extracted timing models are designed to be context-independent within certain ranges of conditions. Before a timing model is created, the internal logic of the block is checked for timing violations, subject to external conditions such as input delay and output delay. After it is verified, the internal timing of the model is guaranteed to have no violations if the external conditions are within the ranges specified when the internal logic was checked.

For hierarchical crosstalk analysis using PrimeTime SI, the internal logic of the timing model is checked with crosstalk, taking into account certain ranges of conditions on the interface nets. If any interface nets are cross-coupled to internal nets (for an interface logic model) or non-port nets (for an extracted timing model), the external arrival times must be within the specified ranges to guarantee that there are not any timing violations inside the block.

When you use a timing model for chip-level analysis, you can have PrimeTime check the actual chip-level conditions for the block instance against the ‘scope’ of the block checked before model creation, and verify that the actual conditions are within the ranges specified at the time of model validation.

The scope checking process is shown in [Figure 20-45](#). After you perform a block-level analysis, you generate a timing model and a block scope file. In the chip-level analysis, PrimeTime verifies that the timing conditions in the context of the top level are within the ranges recorded in the scope file.

The commands used in the scope checking flow are shown in [Figure 20-46](#). After you read in the block-level design, you set the external timing constraints, specifying a range of values, from minimum to maximum, for each constraint. After performing a timing analysis, you generate a timing model using `create_ilm` or `extract_model`, using the

`-block_scope` option to also generate a block scope file. The file contains information about the timing parameter ranges used for block timing validation.

Figure 20-45 Hierarchical Scope Checking Summary

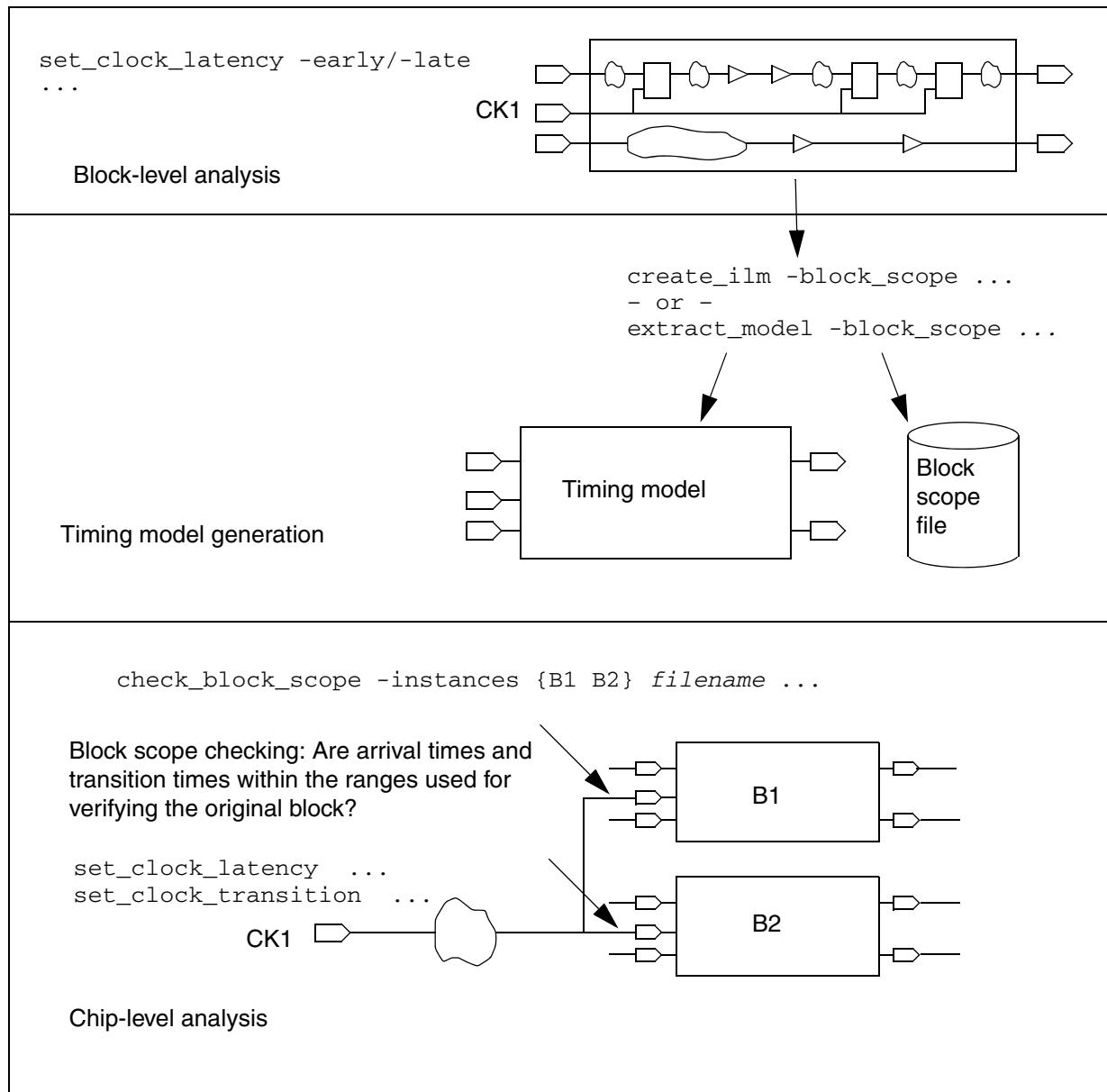
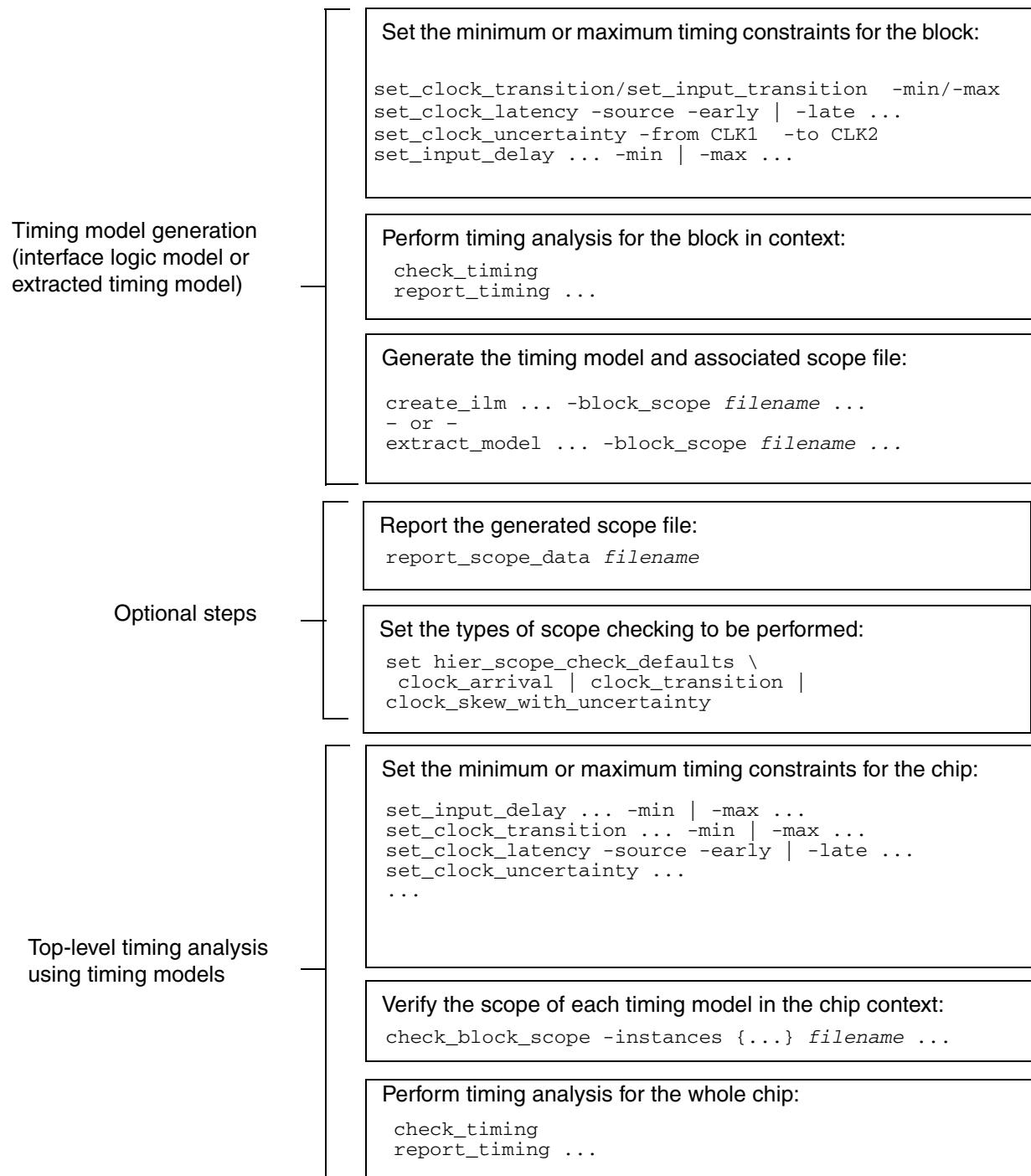


Figure 20-46 Hierarchical Scope Checking Commands



To report the generated scope file, use the `report_scope_data` command. To specify the types of scope checking to perform (clock arrivals, interclock skew, clock uncertainty, clock transition, and so on) set the `hier_scope_check_defaults` variable.

When you use the generated timing model in a chip-level analysis, you can check the timing parameter scope of block instances by using the `check_block_scope` command. This command compares the actual in-context timing parameter values against the contents of the block scope file. If the timing parameters are within the ranges originally defined at the time of model validation, it means that the hierarchical analysis is also valid.

If the timing parameters are outside of the defined ranges, the `check_block_scope` command reports an error and you need to change the constraints, either at the top level or in the lower-level block analysis. Changing the constraints at the top level is preferable because the block level analysis need not be performed again.

Types of Block Scope Checking

The `hier_scope_check_defaults` variable specifies the types of block scope checking performed at the top level. The variable is a string that lists one or more of the following types of checking:

- `clock_arrival` – Clock arrival times
- `clock_transition` – Clock transition times
- `clock_skew_with_uncertainty` – Clock skew and uncertainty together
- `clock_uncertainty` – Clock uncertainty alone without skew
- `data_input_arrival` – Captures the input data arrival times for crosstalk analysis with PrimeTime SI
- `data_input_transition` – Captures the input data transition times for crosstalk analysis with PrimeTime SI

By default, the variable is set to the following string:

```
clock_arrival clock_transition clock_skew_with_uncertainty
```

Therefore, by default, using the `check_block_scope` command checks information about clock arrival times, clock transition times, and clock skew together with uncertainty. In addition, the command always checks the clock waveforms for consistency, irrespective of the variable setting.

To specify a different list of checks, set the variable as in the following example:

```
pt_shell> set_app_var hier_scope_check_defaults "clock_arrival \
    clock_transition clock_skew_with_uncertainty \
    data_input_arrival data_input_transition"
```

Clock Arrival, Transition, and Waveform

In a clock arrival check or clock transition time check, the `check_block_scope` command verifies that the clock arrival time or transition time is within the min/max range originally specified for the clock in the timing model. When you generate the timing model, be sure to specify both the minimum and maximum arrival times and transition times for each clock. For example,

```
pt_shell> set_clock_latency -early 3.2 [get_clocks CLK1]
pt_shell> set_clock_latency -late 4.5 [get_clocks CLK1]

pt_shell> set_clock_transition 0.38 -rise [get_clocks CLK1]
pt_shell> set_clock_transition 0.25 -fall [get_clocks CLK1]
```

When you generate the timing model using the `create_ilm` or `extract_model` command with the `-block_scope` option, PrimeTime finds a leaf-level pin in the fanout of each source clock. It records the name of the pin and the min/max arrival times, transition times, and waveform characteristics of the clock at that pin. If more than one pin meets these requirements, PrimeTime chooses just one pin to use for performing the checks.

During chip-level analysis using the timing model, the `check_block_scope` command reads the name of the pin from the block scope file, finds the pin in the design, and performs the types of checks listed in the `hier_scope_check_defaults` variable. For clock arrival clock transition checking, it verifies that the actual clock arrival times and transition times at the pin are within the min/max ranges recorded in the block scope file.

The `check_block_scope` command always verifies that the clock waveform characteristics, clock period and nominal edge times, are the same as in the block scope file.

Interclock Skew and Uncertainty

The `hier_scope_check_defaults` variable can specify either of two different types of clock skew and uncertainty checking: interclock skew and uncertainty together using the `clock_skew_with_uncertainty` value or uncertainty alone using the `clock_uncertainty` value. In an interclock skew with uncertainty check, the `check_block_scope` command verifies that the amount of skew between two clocks is within the range originally specified for the block with the `set_clock_uncertainty` and `set_clock_latency` commands.

When you generate the timing model using the `create_ilm` or `extract_model` command with the `-block_scope` option, PrimeTime stores the clock source latency values and clock uncertainty values separately in the block scope file. It stores the information relative to pins in the fanout of each input port.

During chip-level analysis using the timing model, the `check_block_scope` command checks the clock latency and clock uncertainty values separately using the `clock_arrival` and `clock_uncertainty` type checks.

In addition, the `clock_skew_with_uncertainty` type check measures and verifies the combined effects of interclock uncertainty and clock source skews at block boundary. For example, for timing paths from CLK1 to CLK2 within the block, the combined required maximum effective skews are computed as follows:

```
required_max_setup_skew =
    min_arrival(CLK2) + max_arrival(CLK1) + setup_uncertainty(CLK1 to CLK2)

required_max_hold_skew =
    max_arrival(CLK2) - min_arrival(CLK1) + hold_uncertainty(CLK1 to CLK2)
```

This same calculation also applies to a single clock used for clocking different paths (intraclock skew). There is separate accounting for rising/falling clock edge combinations when rising and falling edges have different constraints specified on them.

If you specified interclock uncertainty in the timing model, but you only want to check the uncertainty (and not the skew) at the chip level, use the `clock_uncertainty` setting in the `hier_scope_check_defaults` variable. Otherwise, the default, `clock_skew_with_uncertainty`, performs both skew and uncertainty checking.

Data Input and Output Conditions for Crosstalk

For crosstalk analysis using PrimeTime SI, specify the ranges of arrival times and transition times at the block inputs. For example,

```
pt_shell> set_input_delay -min 3.2 [get_ports IN*]
pt_shell> set_input_delay -max 3.8 [get_ports IN*]

pt_shell> set_input_transition -min 0.4 [get_ports IN*]
pt_shell> set_input_transition -max 0.6 [get_ports IN*]
```

This input signal information affects the arrival windows and slew values of block interface nets that are acting as aggressors. When the block is used at a higher level of hierarchy, the actual arrival times and transition times should be within the specified ranges to guarantee the absence of crosstalk timing violations.

When you generate the timing model using the `create_ilm` or `extract_model` command with the `-block_scope` option, PrimeTime finds records the clock skew and uncertainty characteristics on selected clock pins.

If `data_input_arrival` and `data_input_transition` checking are enabled in the `hier_scope_check_defaults` variable, the `check_block_scope` command verifies that the input arrival times and transition times at the block in the top-level design are within the ranges used for analysis of the original block.

At each output of the block, the size of the load affects the transition characteristics of the output operating as an aggressor net. For a conservative analysis, specify the smallest

expected load on the output. If the load on the output is unknown, you can specify a load of zero to ensure a conservative analysis. For example,

```
pt_shell> set_load -min 0.0 [get_ports OUT*]
```

Block Scope Files

A block scope file is a file containing information about the context of a hierarchical timing block. The file is generated by the `create_ilm` or `extract_model` command when the `-block_scope` option is used. When you perform a chip-level analysis using the timing model, the `check_block_scope` command checks the context of the block instance against the context specified at the time of model generation.

Generating a Block Scope File

To generate a block scope file, use the `-block_scope` option of the `create_ilm` or `extract_model` command. When the command generates a model, it also generates the block scope file. For an interface logic model, the name of the generated file is `ilm.scope`. For an extracted timing model, the name of the file is `output_filename.scope`. The block scope file appears in the same directory as the other model files.

You can rename a block scope file and still use it for block scope checking. You specify the block scope file name in the `check_block_scope` command.

To generate the block scope file without generating the timing model files, use the `-block_scope_only` option rather than the `-block_scope` option. This is useful if you have already generated timing model and you only need to make the block scope file.

In a multiscenario analysis, use the `-scope_scenario` option to specify the scenarios in which to generate the block scope data. Block scope information from multiple scenarios can be stored in the same file. If the scope data file already exists and the specified scenario is not already in the file, PrimeTime appends the new data to the existing file.

When PrimeTime generates a block scope file, it puts all types of block context information into the file, not just those listed in the `hier_scope_check_defaults` variable. The variable setting only affects the type of checking done by the `check_block_scope` command.

Reporting Block Scope Files

Block scope files are in binary format, so you cannot examine them directly. To report the contents of a block scope file, use the `report_scope_data` command. For example,

```
pt_shell> report_scope_data myblock/ilm.scope
*****
Report : scope_data
scope_file: /vobs/clt/cltsh/regr/xtalk_2ff/ilm.scope
...
```

```
*****
Scope Data Summary
Block           Scenario          Model Type
-----
xtalk_2ff       <default>        ILM
1

By default, a summary report is generated. To get a detailed report, use the -verbose
option:

pt_shell> report_scope_data myblock/ilm.scope -verbose

*****
Report : scope_data
scope_file: /vobs/clt/cltsh/regr/xtalk_2ff/ilm.scope
...
*****
Summary
-----
Block name: xtalk_2ff
Scope scenario: <default>
Model type: ILM
-----

Clock Arrival Time
Min Condition Arrival Time      Max Condition Arrival Time
-----
Clock Early_r   Early_f   Late_r   Late_f   Early_r   Early_f   Late_r   Late_f Ref_pin
-----
CLK    1.00      1.00      2.00      2.00     1.00      1.00      2.00      2.00   u1/A

Clock Transition Time
Reference
Clock   Min_rise   Min_fall   Max_rise   Max_fall Leaf-pin
-----
CLK      0.00      0.00      0.00      0.00          u1/A
1
```

You must specify at least the name of the block scope file. It is not necessary for any design to be loaded.

The block scope file can contain information from multiple blocks. In that case, you can restrict to the report to a particular block by specifying the block name with the `-block_name` option, or to particular block instances in the current design with the `-instance` option. For example,

```
pt_shell> report_scope_data myblock/ilm.scope -instance B1
...
```

This reports the block scope data for block instance B1 in the current linked design.

By default, the `report_scope_data` command reports the types information listed in the `hier_scope_check_defaults` variable. To override the default list, use the `-check_types` option. To restrict the report to specific ports or clocks, use the `-port_names` or `-clock_names` option, and specify the list of port names or clock names as defined in the original model.

Updating Block Scope Data

To change or update your block scope data files, use the `update_scope_data` command. Use this command to

- Remove the scope data for specific blocks when that information is no longer needed
- Merge two or more block scope files into one file

The `update_scope_data` command changes only the file being updated. It does not change any other files.

Note:

Any changes made to the file being updated are not reversible, so you might want to make a backup copy of the current file.

Checking the Block Scope at the Chip Level

To verify that the context of a timing model is correct in the top-level design, the top-level design using the timing models must be loaded and linked. Then you can use the `check_block_scope` command to perform the scope check. For example,

```
pt_shell> read_verilog mychip.v
pt_shell> link_design
pt_shell> source myconstraints.pt
pt_shell> update_timing
pt_shell> check_block_scope -instances "B1 B2" myblock/ilm.scope
*****
Report : scope_check
  -instance blockA
  -scope_scenario <default>
Design : top
...
*****
1. Matching top-level and block-level clocks.
  (Information) Found top-level clock 'CLK' matching block-level
clock 'CLK'.
  Clock mappings
    Top-level          Block-level        Ref_pin
    -----              -----            -----
      CLK                  CLK           blockA/u1/A       (MET)
```

2. Checking interclock relationships at block boundary.

interclock arrival skews adjusted by uncertainty

From	To	Type	Required	Actual	Slack

No interclock uncertainty violations.					

3. Checking arrival and transition times at block boundary pins.

Clock min condition early rise

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

blockA/u1/A	CLK	1.00	0.00	-1.00 (VIOLATED)

Clock min condition late rise

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

No violations to report.				

Clock min condition early fall

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

blockA/u1/A	CLK	1.00	0.00	-1.00 (VIOLATED)

Clock min condition late fall

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

No violations to report.				

Clock max condition early rise

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

blockA/u1/A	CLK	1.00	0.00	-1.00 (VIOLATED)

Clock max condition late rise

Pin	Clock	Required	Actual	Slack
		Arrival	Arrival	

No violations to report.				

Clock max condition early fall

Pin	Clock	Required Arrival	Actual Arrival	Slack
blockA/u1/A	CLK	1.00	0.00	-1.00 (VIOLATED)

Clock max condition late fall

Pin	Clock	Required Arrival	Actual Arrival	Slack
No violations to report.				

Clock min rise transition

Pin	Required Transition	Actual Transition	Slack
No violations to report.			

Clock max rise transition

Pin	Required Transition	Actual Transition	Slack
No violations to report.			

Clock min fall transition

Pin	Required Transition	Actual Transition	Slack
No violations to report.			

Clock max fall transition

Pin	Required Transition	Actual Transition	Slack
No violations to report.			

1

To get a detailed report, use the `-verbose` option. For a multi-scenario analysis, use the `-scope_scenario` option to specify the scenario name.

Context Characterization

Context characterization is the process of deriving the timing context of a subdesign from its environment in the parent design. The resulting information can be used for hierarchical timing analysis in PrimeTime or for synthesis or logic optimization in Design Compiler.

To learn more about context characterization, see

- [Context Characterization Overview](#)
 - [Deriving the Context of a Subdesign](#)
-

Context Characterization Overview

Context characterization captures the timing context of instances of subdesigns in the chip-level timing environment. The timing context of an instance includes clock information, input arrival times, output delay times, timing exceptions, design rules, propagated constants, wire load models, input drives, and capacitive loads.

Use context characterization for the following purposes:

- To set the timing constraints of a subdesign during synthesis or logic optimization in Design Compiler.
- To perform timing analysis hierarchically in PrimeTime while observing chip-level timing constraints.

Setting Synthesis or Optimization Constraints

The context characterization steps for setting synthesis or optimization constraints in Design Compiler are as follows:

1. Read the top design into PrimeTime.
2. Identify the subdesigns that require timing optimization.
3. Characterize the timing context for each subdesign.
4. Generate a Design Compiler script containing the context information.
5. Read the subdesign into Design Compiler.
6. In Design Compiler, read the script generated in step 4.
7. Perform module-level optimization of the subdesigns.

Performing Subdesign Timing Analysis

The context characterization steps for performing subdesign timing analysis in PrimeTime are as follows:

1. Read the design into PrimeTime.
2. Identify the subdesigns for timing analysis.
3. Characterize the timing context for each subdesign.
4. Generate a PrimeTime script containing the timing assertions.
5. Read a subdesign into PrimeTime.
6. Read the script generated in step 4.
7. Analyze the timing of the subdesign.

Deriving the Context of a Subdesign

The `characterize_context` command derives the timing and environment context from instances of subdesigns in the timing environment. The context of an instance is defined as

- Waveforms of the clocks affecting the instance
- Input arrival times
- Output required times
- Timing exceptions
- Design rules
- Logic constants
- Case analysis
- Wire load models
- Input drives
- Capacitive loads

Use the `characterize_context` command with the `write_physical_annotations` and `write_context` commands to export timing and physical information for a block from the chip-level environment.

The `characterize_context` command does not capture delays and parasitics annotated on the internal nets of the instance being characterized. To capture this information, use the

`write_physical_annotations` command without the `-boundary_nets` option. If you omit the options, PrimeTime characterizes all the information.

To generate the timing-related context information for instance I1 and I2, enter

```
pt_shell> characterize_context -timing {I1 I2}
```

To generate the design rule and the logic constant information from the context of instance I2, enter

```
pt_shell> characterize_context -design_rules I2 -constant_inputs
```

Clock Information

PrimeTime records all clock signals that affect the instance being characterized. Clock information includes signals that feed an input pin of the instance and signals that have sources within the instance. PrimeTime derives information about all clocks driving registers or ports that launch data signals to or receive data signals from the instance.

PrimeTime characterizes the clock waveform, clock signal latency, and the clock uncertainty information for each clock being characterized. PrimeTime does not characterize propagated skew information for clocks feeding input pins of the instance being characterized.

The `write_context` command writes characterized clock information as a series of `create_clock`, `set_clock_latency`, and `set_clock_uncertainty` commands.

Input and Output Delay Times

PrimeTime characterizes arrival times of data signals at input pins of the instance being characterized and their launching clocks. PrimeTime derives required times of data signals at output pins and their related clocks.

The `write_context` command writes input arrival times as `set_input_delay` commands and writes output required times as `set_output_delay` commands.

Point-to-Point Timing Exceptions

PrimeTime derives point-to-point timing exceptions that affect the instance being characterized. The derived exceptions include

- False paths
- Multicycle paths
- `max_delay` and `min_delay` exceptions between clocked startpoints and endpoints

The `characterize_context` command calculates arrival and required times along with port capacitances so that the characterized design has the same timing as the chip-level design.

PrimeTime writes point-to-point exceptions as a series of `set_false_path`, `set_multicycle_path`, `set_max_delay`, and `set_min_delay` commands, depending on their type.

Constant Logic Values on Inputs

The `characterize_context` command derives constant logic values that are propagated to input pins of the instance being characterized.

PrimeTime derives logic constant values as logic constants and case analysis values as case analysis. For more information, see [Case and Mode Analysis](#).

Input Drive Strength and Port Capacitance

PrimeTime derives the following input drive strength and port capacitance information:

Driver information

The drivers of input pins of the instance being characterized are characterized. If the input pin is driven by a port that has a linear drive specified, this drive is also characterized. Drive information is written as `set_drive` and `set_driving_cell` commands.

Port capacitance information

Both the pin capacitance and the wire capacitance on input and output pins are characterized. Pin capacitance is written as a `set_load` command. Wire capacitance is characterized as a fanout number that derives the number of external loads at the port. This number is used together with the wire load model to estimate the boundary net capacitance. If the wire load mode is `enclosed` or `segmented`, each pin of the instance being characterized has a wire load model specified for it, depending on the enclosing hierarchy of the boundary net to which it is connected.

The fanout number is saved as a `set_port_fanout_number` command for PrimeTime and as a `set_fanout_load` command for Design Compiler. The wire load model is saved as a `set_wire_load_model` command for PrimeTime and Design Compiler. In addition to the fanout number, PrimeTime sets a fixed wire capacitance on the port in certain situations to ensure that the pin-to-pin wire capacitance of the boundary net after characterization is the same as that in the top-level design before characterization. This fixed value is written as a `set_load` command.

Wire Load Models

The `characterize_context` command derives the wire load mode and model for characterized instances. PrimeTime always inherits the top-level wire load mode from subdesigns. The wire load model for subdesigns is determined as follows:

- If the top-level mode is “top,” the subdesigns inherit the top-level wire load model.
- If the top-level mode is “enclosed” or “segmented,” the subdesigns inherit the wire load model from the enclosing hierarchy of the net or net segment.

Design Rule Checks

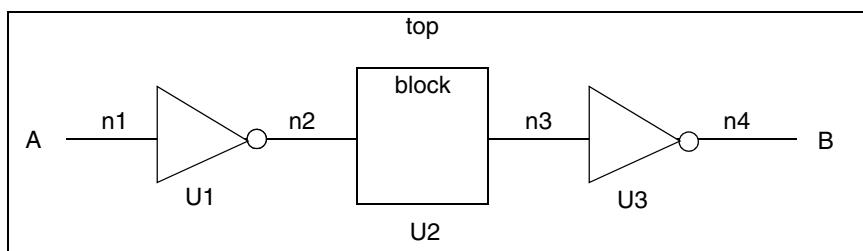
The `characterize_context` command derives design rule checks such as `max_capacitance`, `min_capacitance`, `max_fanout`, `min_fanout`, `max_transition`, and `min_transition` from their context. It also derives `fanout_load` for output pins of the instance being characterized. Each design rule check is saved by means of the corresponding command; for example, `max_fanout` is saved as a `set_max_fanout` command.

Annotated Delays and Parasitics

The `characterize_context` command does not capture delays and parasitics annotated on the internal nets of the instance being characterized. Use the `write_physical_annotations` command to capture this information.

[Figure 20-47](#) shows a simple combinational subdesign called `block`. Script 1 manually sets the port interface attributes and Script 2 starts characterization.

Figure 20-47 Simple Combinational Design



Setup Script

```
current_design top
create_clock -name clk -period 10
set_input_delay 1.0 -clock clk A
set_output_delay 2.0 -clock clk B
```

Script 1

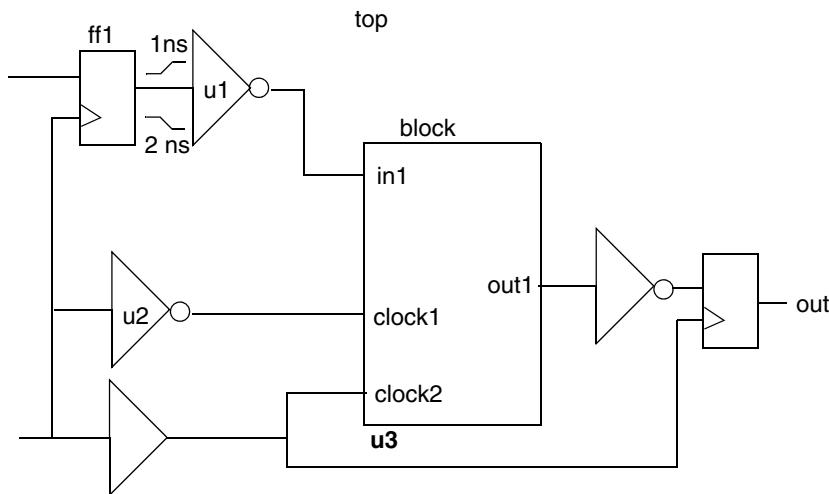
```
current_instance block
create_clock -name clk -period 10
set_driving_cell -lib_cell INV {C}
# derive driving cell information
set_input_delay 3.3 C -clock clk
# arrival time of net n2 is 3.3
set_load 1.3 D
# load of U3 is 1.3
set_output_delay 3.5 D -clock clk
current_instance top
```

Script 2

```
current_design top
characterize_context U2
write_context -output U2.ptsh U2
```

[Figure 20-48](#) shows a subdesign block in the sequential design called top. Script 1 sets port interface attributes manually and Script 2 invokes characterization.

Figure 20-48 Combinational Subdesign Block in Sequential Design



Script 1

```
current_design top
create_clock -period 10 -waveform {0 5} clock

current_instance block
create_clock -name clock -period 10 -waveform {0 5} clock1
create_clock -name clock_bar -period 10 -waveform {5 10}
clock2
# delay from ff1/CP to u5/in1 is 1.8
set_input_delay -clock clock 1.8 in1
# delay of u4 plus ff2 setup time is 1.2
set_output_delay -clock clock 1.2 out1
set_driving_cell -lib_cell INV -input_transition_rise 1 in1 \
    -input_transition_fall 2 in1
set_driving_cell -lib_cell CKINV clock1
set_driving_cell -lib_cell CKBUF clock2
# outside load on out1 net
set_load 0.85 out1
current_instance top
```

Script 2

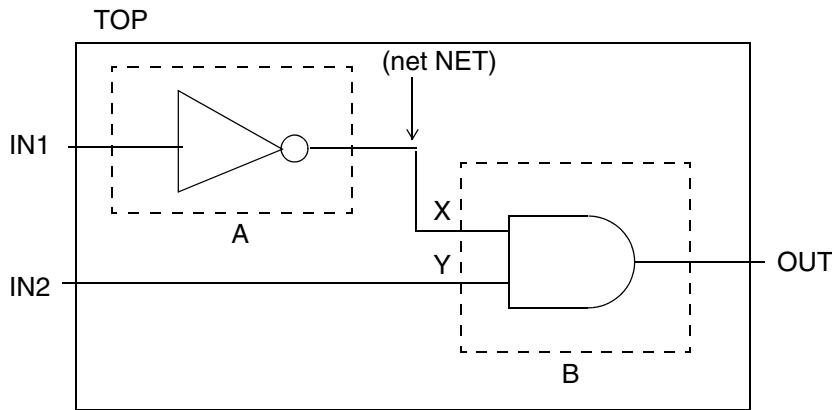
```
current_instance top
create_clock -period 10 -waveform {0 5} clock
characterize_context u5
write_context -output u5.ptsh u5
```

Input Delay and Port Capacitance

The `characterize_context` command calculates arrival and required times along with port capacitance so that the characterized design has the same timing as the chip-level design.

The design in [Figure 20-49](#) is used for the following examples of using the `characterize_context` command:

- [Calculating Loads](#)
- [Calculating Input Delays](#)

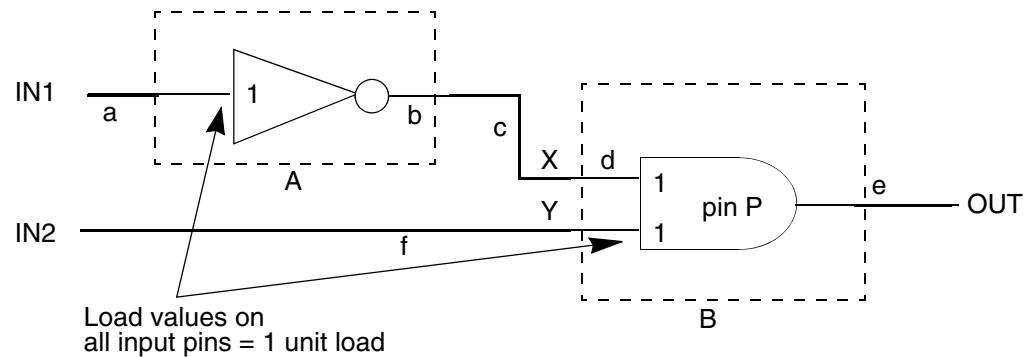
Figure 20-49 Hierarchical Design Example

Calculating Loads

[Figure 20-50](#) shows how PrimeTime annotates the loads of ports and nets in the example in [Figure 20-49](#) and provides some values for wire loads and input pin capacitance.

Figure 20-50 Design With Annotated Ports and Nets

Wire load at TOP level = 5 units load/fanout Wire load for blocks A and B = 2 units load/fanout



In [Figure 20-50](#), a through f represent wire segments used in the calculations. For this example, a segmented wire load model takes into account the interconnection net loads on the blocks and uses a linear function for the wire loads.

For the outside load for pin P in hierarchical block B, the calculation is

```
outside load =
  sum of the loads of all pins on the net loading P
    which are not in B
  + sum of the loads of all segments of net
    driving or loading P which are not in B
```

Each segment's load is

```
segment load = number of fanouts * wire load
```

In the following example, the outside load on input IN1 to block A is calculated by using 0 driving pins, a fanout count of 1 for segment a, and the TOP wire load of 5 loads per fanout:

```
load pins on driving net + load of segment a
= 0 + (1 * 5)
= 5
```

For the outside load on the output of block A, the calculation is

```
load pins on net
+ load of segment c
+ load of segment d
= 1 (for load pin P)
+ (1 * 5)
+ (1 * 2)
= 1 + 5 + 2
= 8
```

For each segment in the calculation, the local wire load model is used to calculate the load. The calculation for block A's output pin uses the TOP wire load of 5 loads per fanout for segment c and block B's wire load of 2 loads per fanout for segment d.

For the outside load on the output of block B, the calculation is

```
load pins on net + load of segment e
= 0 + (1 * 5)
= 5
```

For the outside load on the input pin X of block B, the calculation is

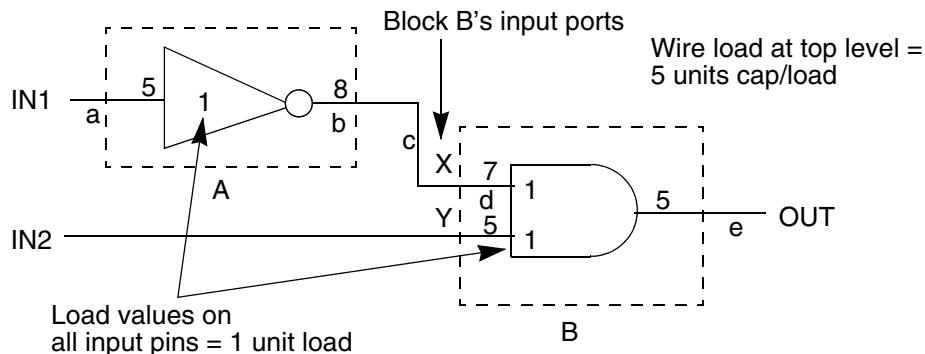
```
load pins on net
+ load of segment b
+ load of segment c
= 0 + (1 * 2) + (1 * 5)
= 7
```

For the outside load on the input pin Y, the calculation is

```
pin loads on driving net + load of segment f
= 0 + 1 * 5
= 5
```

[Figure 20-51](#) shows the same example with the outside loads annotated after characterization.

Figure 20-51 Design With Outside Loads After Characterization

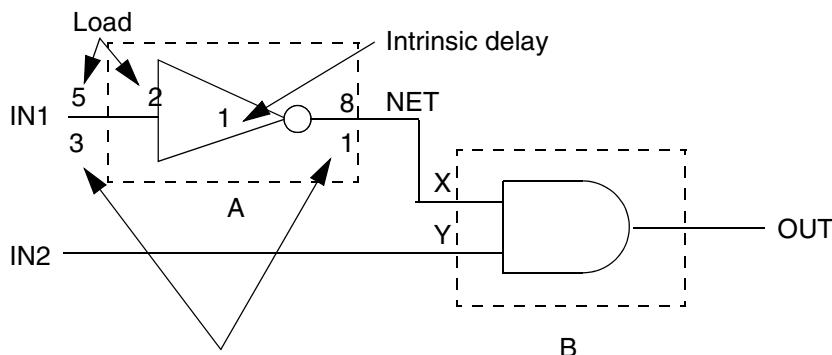


Calculating Input Delays

Because characterization provides accurate details of the outside loads, the input delays or path delays of input signals reflect only the delay through the last gate driving the port. They do not include the connect or transition delays. For example, the characterized arrival times on the input pins of block B are calculated from the delay to the pin that drives the port being characterized, without the transition or connect delays.

[Figure 20-52](#) shows a design example annotated with default drive strengths and intrinsic delays of block A and signal IN1.

Figure 20-52 Design Annotated for Timing Calculations



For input pin X, the delay calculation is

$$\begin{aligned}
 & \text{drive strength at IN1} * (\text{wire load} + \text{pin load}) \\
 & + \text{intrinsic delay of A's cell} \\
 & = 3 * (5 + 2 + 1) \\
 & + 1 \\
 & = 25
 \end{aligned}$$

Writing Physical Information

The `write_physical_annotations` command writes physical information for a hierarchical block in a design. The exported information includes annotated net and cell delays using SDF. PrimeTime can also export annotated parasitics as a series of `pt_shell` and `dc_shell` (dctcl mode) commands.

To write the annotated delays on nets for the hierarchical cell U1, enter

```
pt_shell> write_physical_annotations -sdf U1.sdf -nets_only U1
```

To use the `-append` option to augment the script that PrimeTime generates by the `write_context` command with commands to import the generated delay and parasitic files, enter

```
pt_shell> characterize_context U2
pt_shell> write_context -format dtcl -output U2.dtcl U2
pt_shell> write_physical_annotations -sdf U2.sdf \
-parasitics U2.rc -format dcsh -append U2.dcsh U2
```

Reporting the Timing Context

The `report_context` command reports the timing context derived by the `characterize_context` command. If you do not specify an option, all context information is reported.

To report the timing-related context information for instance I1 and I2, use this command. The report shows clocks and their waveforms, point-to-point timing exceptions, input external delay, and output external delays.

```
pt_shell> report_context -timing {I1 I2}
```

To report the environment and design rule context information for I2, use this command. The report shows design rule checks, wire load models, driving cell information about input pins, and capacitive load on input and output pins of I2.

```
pt_shell> report_context -environment -design_rules I2
```

Generating Scripts for Characterized Contexts

The `write_context` command generates the timing of characterized contexts as a Design Compiler script or a PrimeTime script. Use this command to export context information to other tools. Use `write_context` with the `write_physical_annotations` and `characterize_context` commands to export timing and physical information for a block from the chip-level environment. If you do not specify any options, PrimeTime writes all context information.

To write the timing-related context information for instance I1 and I2 as a `dc_shell` script to standard output, enter

```
pt_shell> write_context -timing -format dcsh {I1 I2}
```

To write the environment and constant input context information for I2 as a PrimeTime shell script into a file called des1.ptsh file, use this command:

```
pt_shell> write_context -environment \  
-constant_inputs -output des1.ptsh I2
```

Removing Context Information

If you no longer need to report or write the context of an instance, you can delete the context information using the `remove_context` command. For the specified list of instances, the `remove_context` command deletes the timing context derived by the `characterize_context` command. If you do not specify an option, PrimeTime deletes all context information.

To delete the timing-related context information for instances I1 and I2, enter the following command:

```
pt_shell> remove_context -timing {I1 I2}
```

To delete the environment and design rule context information for instance I2, enter the following command:

```
pt_shell> remove_context -environment -design_rules I2
```

Limitations of Context Characterization

The following limitations apply to the `characterize_context` command:

Timing exceptions

To accurately constrain certain cases of timing exceptions involving registers, the `characterize_context` command can create virtual clocks and set input and output delays relative to that virtual clock instead of to the source register's clock. You might see these additional virtual clocks in the resulting constraint files.

The `characterize_context` command ignores

- Combinational timing exceptions starting outside the cell being characterized
- Timing exceptions specified with the `-through` option

The `characterize_context` command does not derive all combinational timing exceptions affecting paths in the instance being characterized. Combinational timing exceptions are exceptions defined between unclocked points using the `set_max_delay` and `set_min_delay` commands.

Attributes in a design

The `characterize_context` command ignores clock latency or uncertainty and `max_time_borrow` attributes placed on a hierarchical boundary (generally not an issue because these attributes are usually placed on clocks and cells). The `characterize_context` command does not preserve `path_group` information when deriving output constraints for subdesigns; PrimeTime supports the default clock-based path groups.

Generated clock information

Generated clocks are expanded from the master clock and are characterized as separate clocks. They are saved by means of the `create_clock` command.

Modes

The `characterize_context` command does not characterize the mode information in the top design.

Back-annotation information

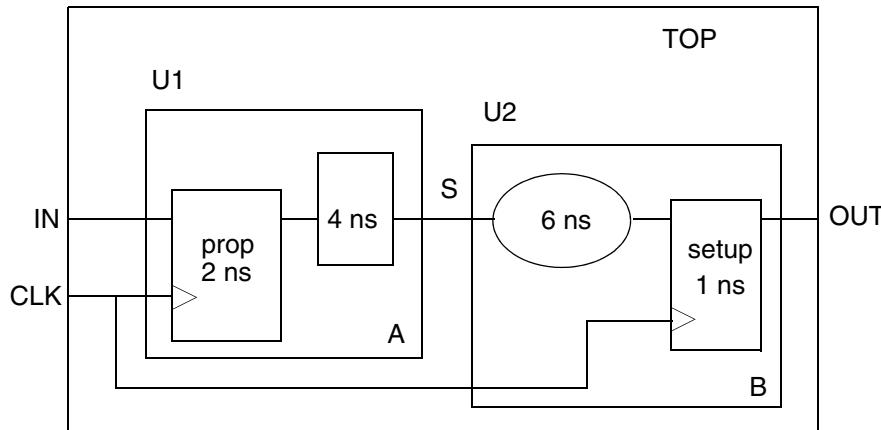
The `characterize_context` command does not derive back-annotation information, such as back-annotated delays and wire capacitance set on nets contained in the instance being characterized. Use the `write_physical_annotations` command in PrimeTime to do this.

Timing budgets

The `characterize_context` command generates subdesign constraints on the basis of absolute path delays in the current design. Timing budgets are not usually needed when the outputs of modules are registered.

The `characterize_context` command generates constraints consistent with optimal timing budgets, if the design is already fully optimized and all paths have zero slack. The best design methodology in many cases is to use the `characterize_context` command to derive noncritical constraints and use timing budgets to constrain critical paths. Timing budgets give you more direct, repeatable control of optimization than characterized constraints. Timing budgets can also reduce the number of `characterize_context` and `compile` iterations necessary to converge on a coherent design hierarchy.

[Figure 20-53](#) shows a design targeted to run at 50 MHz. The path delay through the circuit is 13 ns (including propagation delay through the first flip-flop and the setup time requirement of the second flip-flop). This design meets the timing requirement, but it is faster than necessary.

Figure 20-53 50-MHz Design

You can improve the circuit area by recompiling this design.

- In PrimeTime, use these commands:

```
pt_shell> create_clock CLK -period 20 -waveform {0 10}
pt_shell> characterize_context {A B}
pt_shell> write_context -output A.dcsch -format dcsh A
pt_shell> write_context -output B.dcsch -format dcsh B
```

- In Design Compiler, use these commands:

```
dc_shell> current_design A
dc_shell> include A.dcsch
dc_shell> compile
dc_shell> current_design B
dc_shell> include B.dcsch
dc_shell> compile
dc_shell> current_design TOP
dc_shell> report_timing
```

PrimeTime generates these constraints for design A:

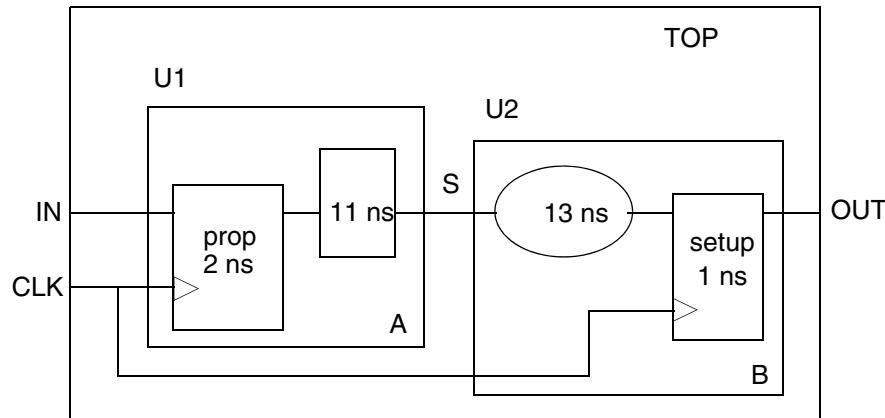
```
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 7 -clock CLK S
```

PrimeTime generates these constraints for design B:

```
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 6 -clock CLK S
```

When design A is compiled, Design Compiler can relax the path delay to 13 ns without violating the generated constraint. Similarly, when design B is compiled, Design Compiler can relax the path delay to 14 ns without violating the generated constraint. The resulting design, shown in [Figure 20-54](#), does not run at the required speed.

Figure 20-54 Design Resulting From Generated Constraint



If you repeat the `characterize_context` and `compile` commands on the design, `characterize_context` generates different constraints.

PrimeTime generates these constraints for design A:

```
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 14 -clock CLK S
```

PrimeTime generates these constraints for design B:

```
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 13 -clock CLK S
```

This resulting design might have the same constraints as the original design. It runs at speed, but it is unnecessarily large. When two blocks are characterized at the same time, then compiled in parallel, each block is overconstrained by the entire amount of negative slack on critical paths between the block and underconstrained by the entire amount of positive slack on noncritical paths between the blocks. To solve this problem, you can use the `characterize_context` command to compile the blocks repeatedly.

- In PrimeTime, use these commands:

```
pt_shell> create_clock CLK -period 20 -waveform {0 10}
pt_shell> characterize_context A -format dcsh -output A.dcsh
```

Use the following commands for design B:

```
current_design TOP
read_ddc A.ddc
link TOP
characterize_context B -format dcsh -output B.dcsh
```

- In Design Compiler, use these commands:

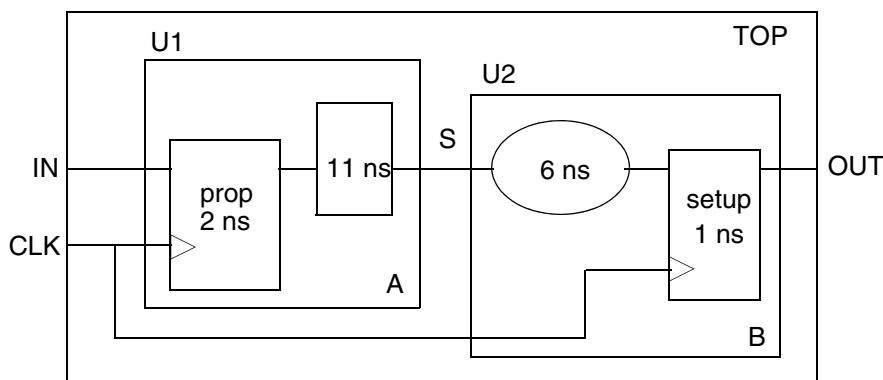
```
dc_shell> current_design A
dc_shell> include A.dcsh
dc_shell> compile
```

Use the following commands for design B:

```
current_design B
include B.dcsh
compile
current_design TOP
report_timing
```

Although this approach produces a smaller design that runs at speed, it might not produce the smallest design that runs at speed. In the preceding command sequence, the available slack is consumed by design A during the first compile. [Figure 20-55](#) shows the resulting design.

Figure 20-55 Design Resulting From Compiling Blocks Repeatedly



To realize the optimal design, you must set time budgets without using the `characterize_context` command. Instead, in Design Compiler, use these commands:

```
current_design A
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 9 -clock CLK S
compile
current_design B
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 11 -clock CLK S
compile
current_design TOP
report_timing
```

21

Using PrimeTime With SPICE

The PrimeTime tool can generate a SPICE deck from the design database for a particular path of interest, including the cross-coupling capacitances. By using the output SPICE deck, you can simulate the path of interest, examine the circuit topology, and verify the crosstalk analysis results.

In addition, the PrimeTime tool provides a simulation link that compares PrimeTime and SPICE results for a timing path or arc.

To learn how to use PrimeTime with SPICE, see

- [Generating a SPICE Deck With the write_spice_deck Command](#)
- [Correlating PrimeTime and SPICE Results With the Simulation Link](#)

Generating a SPICE Deck With the write_spice_deck Command

To generate a SPICE deck, use the `write_spice_deck` command; this command requires a PrimeTime SI license. The command generates a SPICE netlist that includes the cells of a specified path or arc, together with the resistors, ground capacitors, and coupling capacitors to aggressor nets related to the nets in the timing path or arc. The command also provides options to generate the stimuli to sensitize the victim path and aggressors.

To learn about generating and using the SPICE deck, see

- [Writing a SPICE Deck for a Timing Path](#)
 - [Writing a SPICE Deck for a Timing Arc](#)
 - [Library Sensitization in write_spice_deck](#)
 - [Library Driver Waveform](#)
 - [Additional Required Information for SPICE Simulation](#)
 - [Example of write_spice_deck Output](#)
 - [Limitations of Using write_spice_deck for SPICE Correlation](#)
-

Writing a SPICE Deck for a Timing Path

To analyze analyze crosstalk delay effects (but not static noise effects) for a timing path, generate a SPICE deck that represents the timing path. Use the `get_timing_paths` command to collect paths for the `write_spice_deck` command, as shown in [Example 21-1](#).

Example 21-1 Writing a SPICE Deck for a Timing Path

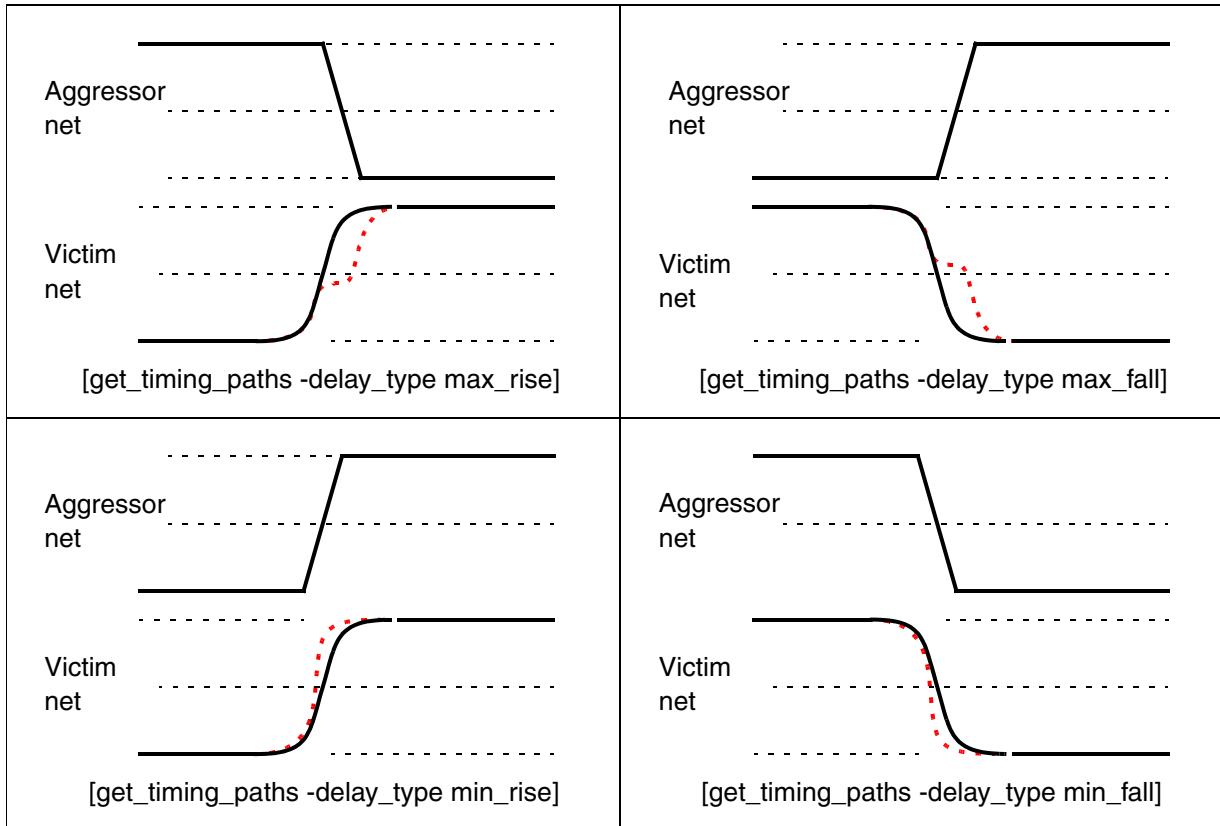
```
pt_shell> write_spice_deck -header header.spi \
    -output testcase.spi \
    -logic_one_voltage 1.5 \
    -logic_zero_voltage 0.0 \
    -sub_circuit_file ./SPICE/subckt.spi \
    [get_timing_paths -from A2 -to buf5/A]
```

To collect specific paths or transitions, use the `get_timing_paths` command with a combination of the `-from`, `-to`, or `-delay_type` options. If you do not use any of these options, the `get_timing_paths` command collects the path with the worst timing slack.

To generate the circuit stimulus waveforms for a particular set of victim and aggressor transitions, use the `-delay_type` option of the `get_timing_paths` command. The option, when set to `max_rise`, `max_fall`, `min_rise`, or `min_fall`, specifies the type of delay (maximum or minimum) and the type of transition considered at the path endpoint (rising or falling). For crosstalk occurring at the path endpoint, this option setting specifies the crosstalk conditions shown in [Figure 21-1](#). For each victim net along the path, the

`write_spice_deck` command sensitizes the aggressor nets appropriately to test the specified maximum or minimum delay transition.

Figure 21-1 Specifying Crosstalk Delay Transitions for Paths



The generated SPICE deck includes all cross-coupled aggressor nets and capacitors along the full path. A long or complex path with a lot of coupling capacitors can result in a very large data file, too large to be practical for SPICE simulation. In that case, try using `get_timing_arcs` to select just a portion of the path that has the victim net and the immediate surrounding circuitry.

Writing a SPICE Deck for a Timing Arc

To analyze crosstalk delay effects or static noise effects for a timing arc, generate a SPICE deck that represents the timing arc. Use the `get_timing_arcs` command with a combination of the `-from`, `-to`, and `-of_objects` options to specify the arc for the `write_spice_deck` command, as shown in [Example 21-2](#).

Example 21-2 Writing a SPICE Deck for a Timing Arc

```
pt_shell> write_spice_deck -header header.spi \
    -analysis_type above_high \
    -output ../SIMUL_BEYOND_HIGH/new_general.spi \
    -logic_one_voltage 1.5 -logic_zero_voltage 0.0 \
    -sub_circuit_file ./SPICE/subckt.spi \
    [get_timing_arcs -to buf5/A]
```

To specify the type of analysis, use the `-analysis_type` option with the following values:

- For crosstalk delay analysis, use `max_rise`, `max_fall`, `min_rise`, or `min_fall`, as shown in [Figure 21-2](#).
- For static noise analysis, use `above_high`, `below_high`, `above_low`, or `below_low`, as shown in [Figure 21-3](#).

Figure 21-2 Crosstalk Delay Transitions for Arcs

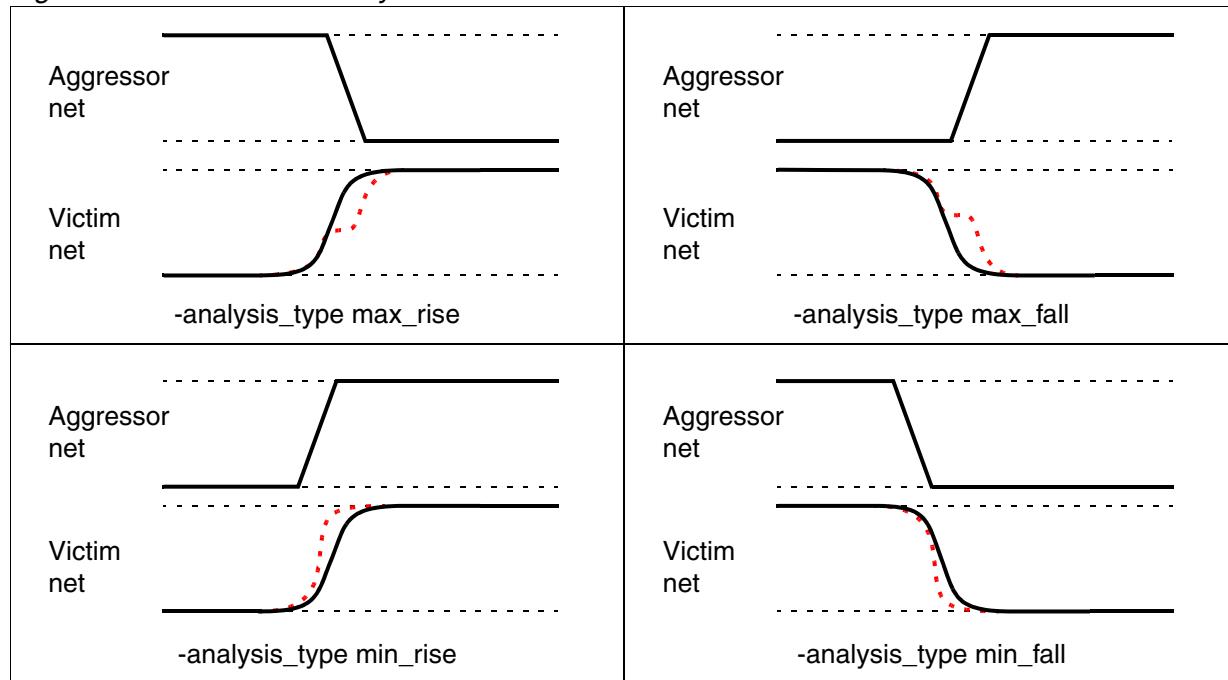
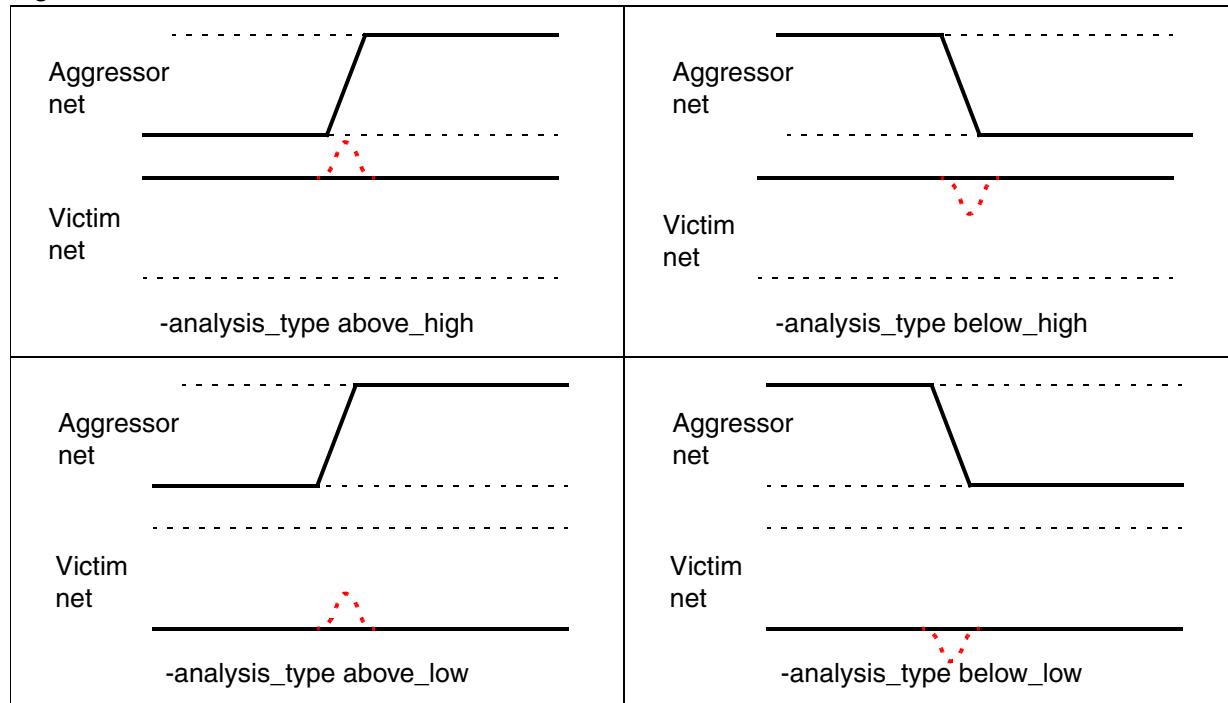


Figure 21-3 Crosstalk Noise Transitions for Arcs

By default, the `write_spice_deck` command places an aggressor transition in the middle of the arrival timing window, not necessarily at the time with worst-case crosstalk effects. To get agreement between PrimeTime SI and SPICE, it is necessary to “sweep” the transition time using a sequence of SPICE simulations, to find the worst-case transition time.

To reduce the simulation effort, use the `-align_aggressors` option of the `write_spice_deck` command. This places the aggressor transitions where they produce the worst-case crosstalk effects for the conditions specified by the `-analysis_type` option. The specified condition can be `max_rise`, `max_fall`, `min_rise`, or `min_fall` for crosstalk delay analysis or `above_high`, `below_high`, `above_low`, or `below_low` for crosstalk noise analysis. The aggressor alignment feature is available for timing stages obtained by the `get_timing_arcs` command, but not for timing paths obtained by the `get_timing_paths` command.

Aggressor alignment is done only for a net timing arc (an arc from the output pin of a cell, through a net, to the input pin of another cell), not for a cell timing arc (an arc from an input pin to an output pin of a cell), even though the coupled RC network of the driven net is written.

When the `write_spice_deck` command uses aggressor alignment for a net arc, it chooses the same driving cell arc that was used during the `update_timing` command in PrimeTime. Aggressor alignment does not work for an aggressor that is directly driven with the `set_driving_cell` command.

Library Sensitization in write_spice_deck

The `write_spice_deck` command can use the stimulus information available in the library for sensitizing the logic cells of a timing path or a stage. This sensitizes the timing arc with the same stimulus used when the arc was characterized. As a result, the PrimeTime-to-SPICE correlation for both coupled and uncoupled analysis is more accurate.

PrimeTime uses the sensitization information contained in the library. In the absence of library-specified sensitization, it applies default sensitization based on the “when” conditions of the timing arc and the logic functions for the combinational logic or the binary state table for sequential logic.

For more information about the cell sensitization syntax in Liberty format, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

The following example demonstrates how the `write_spice_deck` command uses library-defined sensitization information. In this example, the information is specified in the library as follows:

```
sensitization (2in_1out){
    pin_names (IN1, IN2, OUT);
    vector (0, "0 0 0");
    vector (1, "0 0 1");
    vector (2, "0 1 0");
    vector (3, "0 1 1");
    vector (4, "1 0 0");
    vector (5, "1 0 1");
    vector (6, "1 1 0");
    vector (7, "1 1 1");

    cell(my_cell){
        sensitization_master : 2in_1out;
        pin_name_map (A, B, Z);
        ...
        pin(Z) {
            ...
            timing() {
                related_pin : A;
                wave_rise (0, 4, 2, 6, 3);
                wave_fall (1, 5, 3, 6);
                wave_rise_sampling_index : 4;
                wave_fall_sampling_index : 2;
            }
        }
    }
}
```

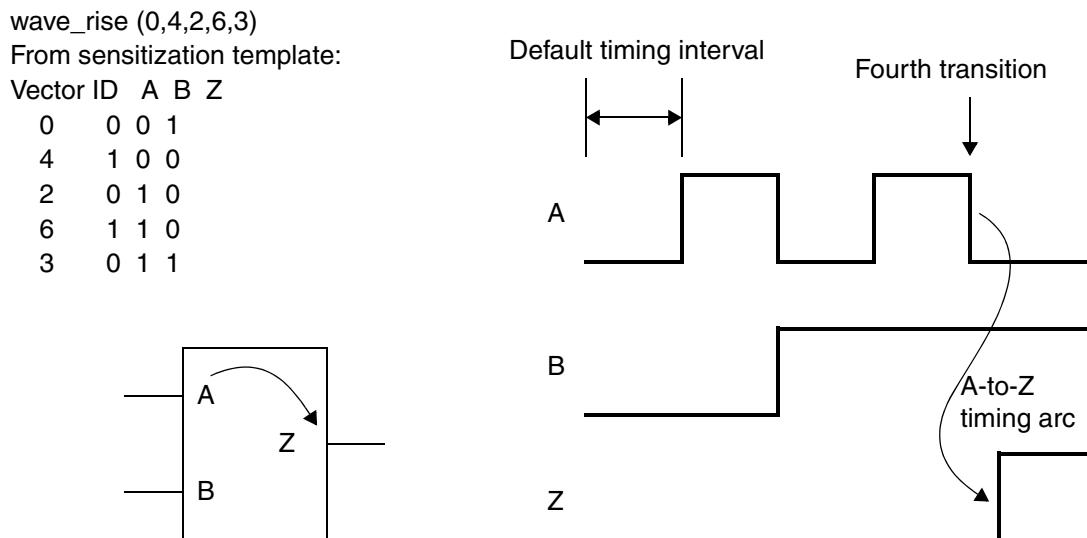
The sensitization template `2in_1out` defines eight vectors. Each vector defines a logic value for the cell pins in the order corresponding to the `pin_names` list.

The `cell(my_cell)` statement instantiates the sensitization template `2in_1out` and maps its pins A, B, and Z to IN1, IN2, and OUT, respectively.

For the timing arc from A to Z, the `wave_rise` attribute defines waveforms at pins A and B that result in a rising edge at pin Z, using a sequential list of vectors previously defined in the `2in_1out` template. The `wave_rise_sampling_index` attribute defines the transition number corresponding to `wave_rise` on which the delay and slew values are to be measured for the timing arc. In this example, it is set to the fourth transition. The sensitization for a falling transition at the output is similarly defined by the `wave_fall` and `wave_fall_sampling_index` attributes.

[Figure 21-4](#) shows the interpretation of the `wave_rise` attribute in PrimeTime SI. The vector sequence (0, 4, 2, 6, 3) represents the waveforms applied to pins A and B to generate a rising edge at pin Z. The delay and slew measurements of the A-to-Z timing arc occur on the fourth sequence transition (in this case, the transition between vector ID number 6 and vector ID number 3).

Figure 21-4 Usage of wave_rise in Library Sensitization Definition



PrimeTime SI assigns a time interval between transitions to a fixed interval such that the associated logic cell has sufficient time to settle down and initialize itself to a proper state. A different time interval can be specified with the `wave_rise_timing_interval` attribute. The following example shows how to set the time interval to 0.5 library time units:

```
wave_rise_timing_interval : 0.5;
```

Due to the adaptive time-stepping used by SPICE simulators, providing more simulation time than necessary for the logic to stabilize does not increase simulation runtime.

Library Driver Waveform

When the `write_spice_deck` command sensitizes the input of a cell, it uses the same waveform that was used for characterization of the timing data. PrimeTime SI gets the predriver modeling information from the library, if available.

If the library does not contain the predriver information, the standard Synopsys predriver is used by default. To specify the predriver type explicitly in PrimeTime SI, use the `set_library_driver_waveform` command.

The `-type` setting of the `set_library_driver_waveform` command specifies the predriver type, either a simple ramp or the standard Synopsys predriver. If you specify one or more library objects (a collection of libraries or library cells) in the command, it applies only to those objects. Otherwise, the command applies to the whole design.

The library driver setting affects not only the `write_spice_deck` command, but also the predriver used for advanced delay calculation using CCS models. For more information, see [PrimeTime SI Crosstalk Delay Calculation Using CCS Models](#).

Additional Required Information for SPICE Simulation

Before you use the deck generated by the `write_spice_deck` command, you must add the following information:

- SPICE subcircuit definitions for all gates used in the SPICE deck; you can use the `.include` statement.
- SPICE models for all transistors and other devices used in the gates
- Definitions for all power and ground nets in the SPICE deck; the header of the SPICE deck contains comment lines that show how to define VDD as power and VSS as ground. All generated ground capacitors are connected to net 0, and all generated piecewise linear (PWL) delay models are relative to net 0.

Check all comment lines in the SPICE deck output for notes, warnings, and error messages. Typical errors include missing subcircuit files, missing definitions in the subcircuit file, incompatible pin order, or an unconstrained path that prevents PrimeTime SI from determining the arrival window.

PrimeTime SI tries to generate all required stimuli for the SPICE deck. However, it might not be able to do so in certain cases, such as for a RAM lacking a definition of the appropriate data.

Example of write_spice_deck Output

Here is an example of the `write_spice_deck` command:

```
pt_shell> write_spice_deck -output my_output \
    -sub_circuit_file spice_subckt \
    -logic_one_voltage 1.8 \
    [get_timing_paths]
```

This example produces a SPICE deck file named `my_output`, based on the definitions given in the SPICE subcircuit file called `spice_subckt`. The `-logic_one_voltage` option specifies the upper voltage swing of the gate input pins to 1.8 volts, which affects piecewise linear (PWL) model generation.

[Example 21-3](#) shows the output file generated by this command. For the applicable input SPICE subcircuit definitions, see [Example 21-4](#).

Example 21-3 SPICE Deck Output

```
MAX. critical path section: (falling) SecIn -> (falling)
SecondReg/D.
*.global vdd vss
*vvdd vdd 0 1.8
*vvss vss 0 0
.include "./spice/spi_deck.subckt"

*** critical path 0 has 7 pins.
***** Arrival Window Info. for pin 'SecIn' *****
* {myclock} pos_edge {min_r_f 0.1 0.1} {max_r_f 0.1 0.1}
* --clock-- {0 2}
*****
* !!! INFO: PrimeTime created the following critical path falling
input port 'SecIn' waveform.
* For rising pwl
* vSecIn SecIn 0 pwl(0.0ns 0 10.0995ns 0 10.1005ns 1.8)
* For falling pwl
vSecIn SecIn 0 pwl(0.0ns 1.8 10.0995ns 1.8 10.1005ns 0)
*****
* resistor(s) for net 'SecIn'.
r0      SecIn:4      SecIn:8      4.000000
r1      SecIn:3      SecIn:4      4.000000
r2      PreInv/A     SecIn:3      4.000000
r3      SecIn:5      SecIn:6      1.368000
r4      SecIn:6      SecIn:8      0.049228
r5      SecIn:5      SecIn       0.053363
* ground capacitor(s) for net 'SecIn'.
c0      PreInv/A     0          0.052000ff
* c1      SecIn:4      0          0.000000ff
c2      SecIn       0          0.081000ff
c3      SecIn:5      0          0.404000ff
c4      SecIn:6      0          0.009000ff
* c5      SecIn:8      0          0.000000ff
```

```

c6      SecIn:3      0      0.026000ff
* cross capacitance of net 'SecIn'.
cc0     SecIn:4      PreInv/Y    0.008000ff
cc1     SecIn:4      PreNet:14   0.026000ff
cc2     SecIn:4      PreNet:8    0.023000ff
cc3     SecIn:4      PreNet:4    0.023000ff
cc4     SecIn:8      PreInv/Y    0.009000ff
cc5     SecIn:8      PreNet:14   0.029000ff
cc6     SecIn:8      PreNet:8    0.026000ff
cc7     SecIn:8      PreNet:4    0.026000ff
cc8     SecIn:3      PreInv/Y    0.008000ff
cc9     SecIn:3      PreNet:14   0.026000ff
cc10    SecIn:3      PreNet:8    0.023000ff
cc11    SecIn:3      PreNet:4    0.023000ff

```

Example 21-4 SPICE Subcircuit Input File

```

.SUBCKT buf1a3 A Y
MU11 N1N4 A VSS VSS n L=0.24 W=1.66
MU12 N1N4 A VDD VDD p L=0.24 W=2.50
MU21 Y N1N4 VSS VSS n L=0.24 W=1.66
MU22 Y N1N4 VDD VDD p L=0.24 W=2.50
.ENDS
.SUBCKT fdf1a6 CLK D Q
M1 N1N56 TP2 N1N119 VSS n L=0.24 W=1.00
M2 N1N119 D VSS VSS n L=0.24 W=1.00
M3 N1N56 TP3 N1N35 VSS n L=0.24 W=0.58
M4 N1N35 TP7 VSS VSS n L=0.24 W=0.58
M5 TP7 N1N56 VSS VSS n L=0.24 W=1.00
M6 TP7 TP3 N1N46 VSS n L=0.24 W=0.58
M7 N1N46 TP2 N1N37 VSS n L=0.24 W=0.58
M8 N1N37 N1N108 VSS VSS n L=0.24 W=0.58
M9 TP3 TP2 VDD VDD p L=0.24 W=0.78
.ENDS
.SUBCKT inv1a6 A Y
M1 Y A VSS VSS n L=0.24 W=3.32
M2 Y A VDD VDD p L=0.24 W=5.00
.ENDS
.SUBCKT or2c3 A B Y
M1I551 N1I551N10 B VSS VSS n L=0.24 W=1.66
M1I552 Y A N1I551N10 VSS n L=0.24 W=1.66
M1I553 Y A VDD VDD p L=0.24 W=2.50
M1I554 Y B VDD VDD p L=0.24 W=2.50
.ENDS

```

Limitations of Using `write_spice_deck` for SPICE Correlation

The `write_spice_deck` command is useful for creating a structural representation of the victim path and its aggressor nets. However, you cannot expect to run a single SPICE deck simulation for a direct, one-to-one comparison with the PrimeTime SI results because of the following reasons:

- The path that you select might be so long, with many cross-coupling capacitors, that even a single SPICE simulation run might take too long to be practical.
- PrimeTime SI does the calculation using multiple iterations, taking into consideration the whole environment of the victim path, using multiple switching cycles. The SPICE deck generator can generate only one victim path (with its aggressor nets) and consider only one switching cycle of the path.
- Other factors, such as the slew propagation scheme and accuracy of the library, can affect the comparison between the PrimeTime SI and SPICE results.

Correlating PrimeTime and SPICE Results With the Simulation Link

PrimeTime static timing analysis operates by using a gate-level netlist representation of the design, combined with library-based timing characterization of the gate-level cells. By analyzing timing at the gate level, the tool performs exhaustive timing analysis of very large designs in a reasonable amount of time.

There can be cases where you want to perform transistor-level timing analysis using circuit simulation. This high level of accuracy might be useful for the following purposes:

- Validation of library characterization methodology in PrimeTime for a new process technology node
- Further analysis of the quality of signals, such as high-fanout nets in a clock network
- Further analysis of signal integrity effects, such as double switching for design fixing

Because it is not practical to perform SPICE simulation on a netlist with millions of gates, PrimeTime provides a simulation link to perform the following SPICE analyses:

- [Path-Based Uncoupled SPICE Analysis](#)
- [Arc-Based Coupled SPICE Analysis](#)

Summary of Simulation Link Commands

Table 21-1 summarizes the simulation link commands that are available to facilitate SPICE analysis in PrimeTime.

Table 21-1 Summary of Simulation Link Commands

Task	Command
Set up the SPICE models for simulation by specifying the library name, subcircuit directory name, and header file name.	<pre>sim_setup_library -library <i>library</i> -sub_circuit <i>dir file</i> -file_name_pattern <i>pattern</i> -header <i>file</i></pre>
Specify the name of the HSPICE simulator executable, simulator options for comparisons between PrimeTime and the simulator.	<pre>sim_setup_simulator [-simulator <i>sim_executable</i>] [-simulator_type hspice nanosim hsim] [-sim_options <i>options_string</i>] [-work_dir <i>dir</i>] [-preserve <i>log_type</i>]</pre>
Validate the simulation setup by performing a simulation on a combinational gate. Specify the name of the cell, the test load, and the input transition time used for simulation.	<pre>sim_validate_setup -lib_cell <i>cellname</i> [-from <i>from_pin</i>] [-to <i>to_pin</i>] [-capacitance <i>value</i>] [-transition_time <i>value</i>] [-verbose]</pre>
Perform path-based uncoupled SPICE analysis on a specified path segment and compares the simulation results against the static timing results.	<pre>sim_validate_path [-from <i>start_pin_object</i>] [-to <i>end_pin_object</i>] [-transition_time] [-verbose] <i>path_objects</i></pre>
Enable nets for coupled one-and-a-half-stage correlation.	<pre>sim_enable_si_correlation <i>list_of_nets</i></pre>
Perform arc-based coupled SPICE analysis on a specified stage or one-and-a-half stage and compares the simulation results against the static timing results.	<pre>sim_validate_stage [-analysis_type min_rise min_fall max_rise max_fall] [-transition_time] [-verbose] <i>net_arc_object</i></pre>

Path-Based Uncoupled SPICE Analysis

With the PrimeTime simulation link, you can perform path-based uncoupled SPICE analysis by following these steps:

1. Configure PrimeTime static timing analysis by using the commands in [Table 21-2](#).

Table 21-2 Configuration Commands for Path-Based Uncoupled SPICE Analysis

Command	Description
<code>set_app_var timing_reduce_parallel_cell_arcs false</code>	Disables parallel arc reduction to ensure that the sensitized cell arc corresponds to the cell arc reported during timing analysis.
<code>set_app_var timing_report_use_worst_parallel_cell_arc false</code>	Reports all paths through parallel cell arcs to ensure that the sensitized cell arc corresponds to the cell arc reported during timing analysis.
<code>set_app_var timing_disable_cond_default_arcs true</code>	Disables usage of non-conditional timing arc between pins that have at least one conditional timing arc. Ensures that the sensitized cell arc corresponds to the cell arc reported during timing analysis.
<code>set_app_var timing_aocvm_enable_analysis false</code>	Disables advanced on-chip variation (AOCV).
<code>remove_aocvm</code>	Disables AOCV for SPICE correlation at specific corners.
<code>reset_timing_derate</code>	Resets user-specified derate factors. Timing derates should be disabled for SPICE correlation.
<code>remove_driving_cell [all_inputs]</code>	Removes all input port driving cell information. <code>set_input_transition</code> should be applied to input ports.
<code>remove_annotated_parasitics [get_nets -of [all_inputs]]</code>	Remove all annotated parasitics at input ports.

Table 21-2 Configuration Commands for Path-Based Uncoupled SPICE Analysis (Continued)

Command	Description
<code>set_app_var timing_keep_waveform_on_points true</code>	When CCS waveform propagation is enabled, setting this variable to <code>true</code> indicates that the waveform data should be stored. The tool accesses the waveform data to write the SPICE stimulus for the first cell in the path. For SPICE correlation where the first cell in the path is driven by another cell, setting this variable to <code>true</code> ensures that the SPICE stimulus matches PrimeTime.

2. Set up the SPICE simulation library by using the `sim_setup_library` command, which specifies the location of the library subcircuits and header file for simulation. For example:

```
pt_shell> sim_setup_library -library mylib \  
-sub_circuit my_subckt_dir -header my_header_file
```

3. Specify the path to the SPICE simulator. For example:

```
pt_shell> sim_setup_simulator \  
-simulator /abc/tools/bin/hspice \  
-simulator_type hspice
```

4. Validate the simulation setup by using the `sim_validate_setup` command, which performs simulation on a combinational gate in the library. This command ensures that the SPICE setup for the simulation link is consistent to that used in the library characterization. For example:

```
pt_shell> sim_validate_setup -lib_cell k04_lib/inv7Q \  
-from A -to Y \  
-capacitance 0.35 -transition_time 0.12
```

PrimeTime invokes the simulation link for the specified cell and compares against PrimeTime static timing analysis results. To check that the SPICE environment is consistent to that used in library characterization, specify the slew and load capacitance values that match an index point in the library.

5. Create a collection of path-based analysis paths for simulation by using the `get_timing_paths` command with the `-pba_mode path` option. For example:

```
pt_shell> set my_paths \  
[get_timing_paths -pba_mode path -delay_type max]
```

6. Submit the paths for SPICE analysis by using the `sim_validate_path` command, which simulates the specified path segment and compares the delay and transition times. For example:

```
pt_shell> sim_validate_path $my_paths
```

Alternatively, you can compare a path segment by using the `-from` and `-to` options. For example:

```
pt_shell> sim_validate_path -from u2/A -to u5/Z \
    -transition $my_paths
```

The validation report shows the original reference value, simulated value, absolute difference, and percentage difference for delay and transition time. For example:

Comparing path number 0 :									
Pin	Sense	Transistion time				Delay			
		Ref	Val	Diff	%	Ref	Val	Diff	%
U3/A	f	0.01971	0.018702	-0.001008	-5.12	0.0176131	# 0.017496	-0.0001173	-0.67
U4/A	r	0.01756	0.017306	-0.000254	-1.45	0.0141726	# 0.014043	-0.0001292	-0.91
U5/A	f	0.134	0.13712	0.00312	2.33	0.083447	# 0.084128	0.0006812	0.82
U5/Z	f	0.2395	0.24316	0.003656	1.53	0.1319	# 0.13457	0.002671	2.03
<hr/>									
Path segment		0.08301	0.083853	0.0008425	1.01	0.247133	0.25024	0.003106	1.26

Note:

- The runtime for SPICE simulation is expected to be longer than the `report_timing` command on the same paths.
- Path-based SPICE simulation of crosstalk is not supported due to timing window alignment required at every stage in SPICE. Voltage and temperature scaling are also not supported.

Arc-Based Coupled SPICE Analysis

With the PrimeTime SI simulation link, you can analyze the signal integrity behavior of coupled delay in SPICE, write out a SPICE netlist based on the victim net arc, simulate it with the HSPICE simulator, and automatically compare the HSPICE and PrimeTime SI results.

By using PrimeTime SI, you can select a net arc or multiple net arcs for simulation. A one-and-a-half-stage delay is recommended for correlation purposes. This feature is applicable only if the net has an annotated coupled RC network. Filtered aggressors are not considered effective during calculation, and their coupling capacitances are grounded in the SPICE netlist.

Arc-based coupled SPICE analysis requires that PrimeTime SI is enabled. Voltage and temperature scaling are not supported.

To perform arc-based coupled SPICE analysis, follow these steps:

1. Enable PrimeTime SI analysis by using the `si_enable_analysis` variable:

```
pt_shell> si_enable_analysis true
```

2. Configure PrimeTime SI analysis by using the `sim_enable_si_correlation` command:

```
pt_shell> sim_enable_si_correlation $nets
```

3. Set up the SPICE simulation library by using the `sim_setup_library` command, which specifies the location of the library subcircuits and header file for simulation. For example:

```
pt_shell> sim_setup_library -library mylib \
           -sub_circuit my_subckt_dir -header my_header_file
```

4. Specify the path to the SPICE simulator. For example:

```
pt_shell> sim_setup_simulator \
           -simulator /abc/tools/bin/hspice \
           -simulator_type hspice
```

5. Validate the simulation setup by using the `sim_validate_setup` command, which performs simulation on a combinational gate in the library. This command ensures that the SPICE setup for the simulation link is consistent to that used in the library characterization. For example:

```
pt_shell> sim_validate_setup -lib_cell k04_lib/inv7Q \
           -from A -to Y \
           -capacitance 0.35 -transition_time 0.12
```

PrimeTime invokes the simulation link for the specified cell and compares against PrimeTime static timing analysis results. To check that the SPICE environment is consistent to that used in library characterization, specify the slew and load capacitance values that match an index point in the library.

6. Create a collection of coupled net arcs for simulation by using the `get_timing_arcs` command. For example:

```
pt_shell> set my_arcs [get_timing_arcs -from U1/Z -to U2/A]
```

7. Enable coupled one-and-a-half-stage correlation for \$net, and perform a timing update. For example:

```
pt_shell> sim_enable_si_correlation \
           [get_net -of [get_attribute $my_arcs from_pin]
pt_shell> update_timing -full
```

8. Perform SPICE analysis by using the `sim_validate_stage` command, which simulates the specified one-and-a-half stage and compares the stage delay. The validation report

shows the original reference value, simulated value, absolute difference, and percentage difference for delay. For example:

```
pt_shell> sim_validate_stage $my_arcs
```

Pin	Sense	Delay			
		Ref	Val	Diff	%
U2/Z	f	0.59594 #	0.594371	-0.00156873	-0.26
Tolerance	MET			0	3.00

If the arc includes the receiver stage, one-and-a-half stage analysis is done. If the arc does not include the receiver stage, one stage correlation is reported instead. For example:

```
pt_shell> set my_arcs [get_timing_arcs -from U1/z -to FF1/D]
pt_shell> sim_validate_stage $my_arcs
```

Pin	Sense	Delay			
		Ref	Val	Diff	%
FF1/D	f	0.59594 #	0.594371	-0.00156873	-0.26
Tolerance	MET			0	3.00

22

Using PrimeTime With Design Compiler

PrimeTime works very well with the Design Compiler synthesis tool. However, there are some differences in command syntax and behavior between the two tools.

To learn how to make the two tools work together efficiently, see

- [Features Specific to PrimeTime](#)
- [Timing Analysis Differences](#)
- [Command Scripts](#)
- [Reading .db Files With Back-Annotated Data](#)
- [Path Groups in Design Compiler](#)

Features Specific to PrimeTime

Some features of PrimeTime are not supported by Design Compiler, such as internal clocks, certain features of extracted timing models, and mode analysis (timing models with operating modes).

Design Compiler does not support the creation of clocks on internal pins of a leaf cell. You must specify these clocks at input or output pins that are contained in the fanin or fanout of the internal pin.

Design Compiler cannot analyze and optimize designs with timing models that use mode analysis. You must manually disable the inactive timing arcs to analyze and optimize a module in a particular operating mode.

Timing Analysis Differences

To learn about the differences in timing analysis behavior between PrimeTime and Design Compiler, see

- [Paths](#)
- [Transition Time](#)
- [current_design Command](#)

Paths

The following differences apply to paths.

Time Borrowing for Hold Checks

PrimeTime disables short-path borrowing by default. (Short-path borrowing is time borrowing for hold checks at level-sensitive latches.) Design Compiler always allows borrowing for short paths.

To simulate the Design Compiler behavior in PrimeTime, set the `timing_allow_short_path_borrowing` variable to `true`. For more information about path borrowing, see the *Synopsys Timing Constraints and Optimization User Guide*.

Segmenting Paths: Load-Dependent Delays

In Design Compiler, if a timing report originates from a segmented path's new startpoint, Design Compiler transfers the load-dependent portion of the source gate's delay to the output net delay. This method can provide a better synthesis result, but it is not appropriate for static timing analysis. PrimeTime does not add the load-dependent delay to the net.

Transition Time

For bidirectional (inout) pins, PrimeTime maintains separate transition time information for the driver and load parts of the pin. Design Compiler keeps the maximum of driver and load transition times.

current_design Command

Design Compiler requires that you change the current design used to specify information locally, such as wire load models:

```
dc_shell> current_design BOTTOM
dc_shell> set_wire_load_model -name small
dc_shell> current_design MID
dc_shell> set_wire_load_model -name medium
dc_shell> current_design TOP
dc_shell> set_wire_load_model -name large
```

In PrimeTime, you achieve this by using the `current_instance` command or by specifying hierarchical cell names. Do not use `current_design` for this purpose. For example,

```
pt_shell> current_design TOP
pt_shell> set_wire_load_model -name large
pt_shell> set_wire_load_model -name medium [get_cells U1]
pt_shell> set_wire_load_model -name small [get_cells U1/U2]
```

Command Scripts

You can use command scripts in PrimeTime. A command script is a sequence of `pt_shell` commands in a text file. The file can be in ASCII, gzip, or bytecode format. The `source` command executes scripts in PrimeTime.

Sharing Design Compiler and PrimeTime Scripts

You can share scripts when you use both Design Compiler and PrimeTime on the same design. The easiest way to do so is to use Synopsys Design Constraints (SDC), as described in [Synopsys Design Constraints Formatted Script Files](#). Another method is to keep scripts that contain shared commands separate from scripts that contain tool-specific commands, as demonstrated in the following examples.

Note:

You cannot convert dc_shell Tcl scripts to pt_shell scripts, but you can source some dc_shell Tcl scripts directly into pt_shell. An alternative method is to write shared constraint files in Tcl using the SDC format. For more information, see [Synopsys Design Constraints Formatted Script Files](#).

The following dc_shell script includes another script called timing_assertions.scr:

```
current_design MID
set_wire_load_model -name medium
current_design TOP
set_wire_load_mode enclosed
set_wire_load_model -name large
set_max_area 3000
set_dont_touch u13
include timing_assertions.scr
compile
```

The following script is part of the timing_assertions.scr script, which contains commands that both Design Compiler and PrimeTime understand:

```
set_operating_conditions WCCOM
create_clock -period 10 CLK
set_input_delay ...
set_output_delay ...
```

To use the Design Compiler script in PrimeTime, use the timing assertions script and a script that contains PrimeTime commands. Some PrimeTime commands are identical to Design Compiler commands in syntax. However, as mentioned earlier, you need to specify tool-specific commands differently.

In this example, the pt_shell script contains commands that are equivalent to the commands in the dc_shell script shown earlier:

```
set_wire_load_mode enclosed
set_wire_load_model -name medium {u2 u4}
set_wire_load_model -name large
source timing_assertions.scr
set_max_area 3000
set_dont_touch u13
```

Synopsys Design Constraints Formatted Script Files

SDC formatted script files are Tcl scripts that use a subset of the commands supported by PrimeTime and Design Compiler. There are limitations placed on some of the supported commands. For example, not all options are supported in some cases. Although SDC is a subset of Tcl, it is not a full-function scripting language. It is intended to support the specification of timing constraints across various electronic design automation tools. It is not intended for complex scripting.

Within SDC, there is only one design: the current design. You cannot change the current design with SDC.

SDC files are read into PrimeTime with the `read_sdc` command, and they are written from PrimeTime using the `write_sdc` command.

Checking the Syntax of the SDC File

Check the SDC file and verify that it is syntactically and semantically correct by using the `read_sdc -syntax_only` command. This validates the script without having any effect on the design. For more information about SDC, see the *Using the Synopsys Design Constraints Format Application Note*, which is available on [SolvNet](#).

Reading .db Files With Back-Annotated Data

Improve the performance of reading netlist .db format files with back-annotation information into PrimeTime by using one of the following procedures.

Procedure 1

1. In dc_shell, save the .db format file with no back-annotation information and write the back-annotation information using the Standard Delay Format (SDF) file.
2. In pt_shell, read the .db format file, which is not back-annotated, and the generated SDF file. This is faster than reading the back-annotated .db format file directly in PrimeTime.

Procedure 2

1. In dc_shell, save the timing assertions set on the design with the `write_script` command. Make sure you remove commands that are not timing related.
2. In pt_shell, read the back-annotated .db format file, but use the `-netlist_only` option of the PrimeTime `read_db` command to instruct PrimeTime to ignore the back-annotation.
3. Apply the timing assertions to the design by sourcing the generated script. Make sure the script is converted from a Design Compiler script to a PrimeTime script.

Path Groups in Design Compiler

In Design Compiler, use path groups to modify the cost function for optimization. You can assign paths or endpoints to named groups. Each group has a weighting value that you can specify, which contributes to the maximum delay cost for the design.

PrimeTime checks each timing path for maximum delay violations: it compares the actual path delay for rising and falling signals to a target path delay. The worst violation for a group is the path with the largest negative slack.

You can specify path groups in PrimeTime. When you request a path-timing report, PrimeTime lists a report for each path group. Export this information to Design Compiler by using the `write_script` or `write_context` command or by budgeting the design.

When you create an explicit path group, you can also assign a critical range for the path group. The critical range defines a margin of delay for the path group during optimization in Design Compiler. A nonzero value allows the optimization of near-critical paths if the worst violation is not increased.

23

Attributes

An attribute is a string or value associated with an object that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can write programs in Tcl to get attribute information from the design database and generate custom reports on the design.

To learn how to use attributes, see

- [Using Attributes](#)
- [Saving Design Attributes](#)
- [Using Paths to Generate Custom Reports](#)
- [Using Arcs to Generate Custom Reports](#)

Using Attributes

PrimeTime provides a set of commands for setting, reporting, listing, and creating attributes, as summarized in [Table 23-1](#).

Table 23-1 Attribute Commands

Attribute command	Description
define_user_attribute	Creates a new attribute for one or more classes
get_attribute	Retrieves the value of any attribute from a single object
list_attributes	Shows the attributes defined for each object class or a specified object class; optionally shows application attributes
remove_user_attribute	Removes a user-defined attribute from one or more objects
report_attribute	Displays the value of all attributes on one or more objects; optionally shows application attributes
set_user_attribute	Sets a user-defined attribute on one or more objects

The `get_attribute`, `report_attribute`, `set_user_attribute`, and `remove_user_attribute` commands accept an object specification, which can be a collection or a list of collections. The object specification for the `get_attribute` command is limited to one collection containing one object.

For descriptions of the predefined application attributes of each object class, see the man pages. For example, to see the descriptions of library cell attributes, use the following command:

```
pt_shell> man lib_cell_attributes
```

Creating User-Defined Attributes

The `define_user_attribute` command defines a new attribute. PrimeTime has a set of attributes that it considers application attributes. You can view these attributes by using `list_attributes -application`. You must define any other attribute before you can use it in PrimeTime. One such use is importing an attribute from the .ddc or .db file.

You can apply attributes to most object classes in PrimeTime. These can mark interesting cells or nets, store values you have computed, and so on. PrimeTime cannot use these attributes, but you can use them in scripts, procedures, and so on. You can list the attributes

you define using the `list_attributes` command. When you define an attribute, decide on the appropriate data type.

Use the following commands to define attributes of various types. You can specify more than one class. Enter

```
pt_shell> define_user_attribute attr_s -classes {cell net} -type string
pt_shell> define_user_attribute attr_i -classes cell -type int
```

The following example defines attribute `attr_ir1` as ≥ 0 and ≤ 100 , attribute `attr_ir2` as ≥ 0 with no maximum, and attribute `attr_ir3` as ≤ 100 with no minimum.

```
pt_shell> define_user_attribute attr_ir1 -classes cell -type int \
           -range_min 0 -range_max 100
pt_shell> define_user_attribute attr_ir2 \
           -classes cell -type int -range_min 0
pt_shell> define_user_attribute attr_ir3 \
           -classes cell -type int -range_max 100
```

The following example defines the `attr_oo` attribute for cells. The attribute is a string, but you can set it only to A, B, C, or D:

```
pt_shell> define_user_attribute attr_oo \
           -classes cell -type string -one_of {A B C D}
```

Importing User-Defined Attributes

To import user-defined attributes (attributes that are not PrimeTime application attributes) from a .ddc or .db file into PrimeTime, you must define the attributes in PrimeTime before you read any designs.

Assume you created a Boolean attribute in Design Compiler called `MarkedPin` and placed this attribute on several pins. In PrimeTime, before you read any designs you must define the `MarkedPin` attribute. For example,

```
pt_shell> define_user_attribute -classes pin -type Boolean \
           -import MarkedPin
```

After you define the attribute in this way, PrimeTime extracts it from the .ddc or .db file as it would extract any other attribute. For example,

```
pt_shell> list_attributes -class pin
*****
Report : List of Attribute Definitions
...
*****
Properties:
  A - Application-defined
  U - User-defined
  I - Importable from db (for user-defined)

Attribute Name      Object     Type      Properties   Constraints
-----
MarkedPin           pin        Boolean    U,I
```

Attributes appear on design objects only after the design is linked.

User-defined attributes can be imported from both top-level designs and lower-level designs. To import user-defined attributes at the top level, there are no special considerations. You can use the `define_user_attribute -import` command; however, note that a top-level design attribute is attached to a design, but an inherited (lower-level) design attribute is attached to an instance of that design, which is a cell. Therefore, to import and inherit a lower-level design attribute, the attribute must be defined for both designs and cells. The `-import` option is only needed at the design level.

Similarly, a top-level port attribute is attached to a port, but an inherited port attribute is attached to an instance of that port, which is a pin. Therefore, to import and inherit a port attribute, the attribute must be defined for both ports and pins. The `-import` option is only needed at the port level.

To import and inherit a design attribute, the attribute must be defined for both designs and cells, although it only needs to be imported for the design. To import and inherit a port attribute, the attribute must be defined for both ports and pins, although it only needs to be imported for the port. A required attribute is defined automatically if necessary to import a

related attribute. After the design is linked, you can see that user-defined attributes have been applied. For example,

```
pt_shell> report_attribute [get_pins */CP]
*****
Report : Attribute
...
*****
Design      Object      Type      Attribute Name      Value
-----
M           o_reg1/CP   Boolean   MarkedPin      true
M           o_reg2/CP   Boolean   MarkedPin      true
M           o_reg3/CP   Boolean   MarkedPin      true
M           o_reg4/CP   Boolean   MarkedPin      true
```

You can create a collection of all pins with this attribute. For example,

```
pt_shell> set ipins [get_pins * -hierarchical -filter \
"MarkedPin == true"]
```

Saving Design Attributes

By default, PrimeTime saves nothing when it exits. To save design attributes that you applied during a session, use the `write_script` command.

The `write_script` command writes the following information:

- Clock-related information – Clock creation, generated clock creation, clock latency, clock uncertainty, and interclock uncertainty.
- Timing-related information – Disable timing, maximum time borrow.
- Point-to-point exceptions – False path, minimum delay, maximum delay, multicycle path, group path, input delay, output delay, annotated delay, and annotated checks.
- Net attributes – Capacitance and resistance.
- Port attributes – Fanout, capacitance, and resistance.
- Design environment – Wire load model, operating condition, drive, driving cell, and input transition.
- Design rules – Minimum capacitance, maximum capacitance, minimum transition, maximum transition, minimum fanout, and maximum fanout.

You can save the design attributes in Synopsys `pt_shell`, `dc_shell`, or `dctcl` script formats, then use the script to re-create the attributes on the design.

Using Paths to Generate Custom Reports

Use the `get_timing_paths` command to create a collection of paths for custom reporting and other processing. You can assign these timing paths to a variable or pass them into another command.

To list timing path object attributes, use the following command:

```
pt_shell> list_attributes -application -class lib_timing_path
```

Each use of the `get_timing_paths` command has its own context. You cannot compare, add, or remove objects taken from different contexts. For example,

```
pt_shell> set paths1 [get_timing_paths -nworst 10]
...
pt_shell> set paths2 [get_timing_paths -nworst 100]
...
```

Even though the variables `paths1` and `paths2` contain some of the same objects, the collections cannot be compared because they come from different contexts. To do such comparisons, create one large collection containing all the objects of interest, and then perform filtering or other manipulation on that collection.

Use the `foreach_in_collection` command to iterate among the paths in the collection. The collection commands `index_collection`, `copy_collection`, `add_to_collection`, and `remove_from_collection` are not applicable to timing path collections. Use the `get_attribute` command to obtain information about the paths.

One attribute of a timing path is the points collection. A point corresponds to a pin or port along the path. Iterate through these points using the `foreach_in_collection` command and get the attributes on them using the `get_attribute` command.

You can find some detailed examples of custom timing reports in `install_dir/auxx/pt/examples/tcl` (where `install_dir` is the installation directory for PrimeTime).

The following procedure prints out the startpoint name, endpoint name, and the slack of the worst path in each path group:

```
proc custom_report_worst_path_per_group {} {
    echo [format "%-20s %-20s %7s" "From" "To" "Slack"]
    echo -----
    foreach_in_collection path [get_timing_paths] {
        set slack [get_attribute $path slack]
        set startpoint [get_attribute $path startpoint]
        set endpoint [get_attribute $path endpoint]
        echo [format "%-20s %-20s %s" [get_attribute $startpoint full_name]
              [get_attribute $endpoint full_name] $slack]
    }
}
```

```
pt_shell> custom_report_worst_path_per_group
```

From	To	Slack
ffa/CP	QA	0.1977
ffb/CP	ffd/D	3.8834

The following example shows worst negative slack, total negative slack, and total positive slack for the current design:

```
proc report_design_slack_information {} {
    set design_tns 0
    set design_wns 100000
    set design_tps 0
    foreach_in_collection group [get_path_groups *] {
        set group_tns 0
        set group_wns 100000
        set group_tps 0
        foreach_in_collection path [get_timing_paths -nworst 10000 \
            -group $group] {
            set slack [get_attribute $path slack]
            if {$slack < $group_wns} {
                set group_wns $slack
                if {$slack < $design_wns} {
                    set design_wns $slack
                }
            }
            if {$slack < 0.0} {
                set group_tns [expr $group_tns + $slack]
            } else {
                set group_tps [expr $group_tps + $slack]
            }
        }
        set design_tns [expr $design_tns + $group_tns]
        set design_tps [expr $design_tps + $group_tps]
        set group_name [get_attribute $group full_name]
        echo [format "Group '%s' Worst Negative Slack : %g" $group_name
$group_wns]
        echo [format "Group '%s' Total Negative Slack : %g" $group_name
$group_tns]
        echo [format "Group '%s' Total Positive Slack : %g" $group_name
$group_tps]
        echo ""
    }
    echo -----
    echo [format "Design Worst Negative Slack : %g" $design_wns]
    echo [format "Design Total Negative Slack : %g" $design_tns]
    echo [format "Design Total Positive Slack : %g" $design_tps]
}
```

```
pt_shell> report_design_slack_information
```

```

Group 'CLK' Worst Negative Slack : -3.1166
Group 'CLK' Total Negative Slack : -232.986
Group 'CLK' Total Positive Slack : 4.5656

Group 'vclk' Worst Negative Slack : -4.0213
Group 'vclk' Total Negative Slack : -46.1982
Group 'vclk' Total Positive Slack : 0

-----
Design Worst Negative Slack : -4.0213
Design Total Negative Slack : -279.184
Design Total Positive Slack : 4.5656

```

Using Arcs to Generate Custom Reports

To create a collection of timing arcs for custom reporting and other processing, use the `get_timing_arcs` command. You can assign these timing arcs to a variable and get the desired attributes for further processing.

To list timing arc object attributes, use the following command:

```
pt_shell> list_attributes -application -class timing_arc
```

Use the `foreach_in_collection` command to iterate among the arcs in the collection. Use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index the collection of arcs. You can also use the `-filter` option of `get_timing_arcs` to obtain just the arcs that satisfy specified conditions, but you cannot use the `filter_collection` command to filter an already generated collection of arcs.

One attribute of a timing arc is `from_pin`, which is the pin or port from which the timing arc begins. In the same way, `to_pin` is the pin or port at which the timing arc ends. For more information about the `from_pin` and `to_pin` attributes, use the `get_attribute` command.

For example, to list the maximum rise delay value of all the timing arcs of the cell U1 that have the `positive_unate` arc, enter

```

pt_shell> set arcs [get_timing_arcs -of_objects U1 -filter \
    "sense == positive_unate"]
_pt13
pt_shell> foreach_in_collection arc $arcs {
    echo [get_attribute $arc delay_max_rise]
}
0.846862
0.846862

```

Creating a Collection of Library Arcs

To create a collection of library arcs for custom reporting and other processing, use the `get_lib_timing_arcs` command. You can assign these library arcs to a variable and get the desired attributes for further processing.

To iterate among the library arcs in the collection, use the `foreach_in_collection` command. To obtain information about the arcs, use the `get_attribute` command. However, you cannot copy, sort, or index a collection of library arcs. You can also use the `-filter` option of `get_lib_timing_arcs` to obtain just the library arcs that satisfy specified conditions, but you cannot use the `filter_collection` command to filter a collection of library arcs.

To list library timing arc object attributes, use the following command:

```
pt_shell> list_attributes -application -class lib_timing_arc
```

One attribute of a library timing arc is `from_lib_pin`, which is the library pin from which the timing arc begins. In the same way, `to_lib_pin` is the library pin at which the timing arc ends. To obtain information about the `from_lib_pin` and `to_lib_pin` attributes, use the `get_attribute` command.

For example, to list the senses of timing arcs starting from the clock pin of a flip-flop library cell, enter

```
pt_shell> set larcs [get_lib_timing_arcs -from class/FD1/CP]
           _sel5
pt_shell> foreach_in_collection larc $larcs \
           { echo [get_attribute $larc sense] }
hold_clk_rise
setup_clk_rise
rising_edge
rising_edge
```

Reporting Library Data and Driver Information

To validate library data and driver models used in delay calculation with annotated parasitics, you can report library data and the resulting driver model parameters for a specified library arc and conditions. To do this, use the `report_driver_model` command.

Glossary

active edge

The low-to-high or high-to-low transition of a clock signal that causes data to be latched into a flip-flop.

aggressor net

A net that causes undesirable crosstalk effects on another net (called the victim net) due to parasitic cross-capacitance between the two nets.

annotation

Information that is attached to an object, such as a delay value attached to a net; or the process of attaching information to an object.

AOCV

Advanced on-chip variation (AOCV) is a Synopsys technology that reduces unnecessary pessimism by using the location and logic depth of each path analyzed.

arc

A set of timing constraints associated with a particular path.

ASIC

Application-specific integrated circuit; a fabricated electronic device designed to perform a specific task.

ASIC vendor

A company that manufactures integrated circuits designed by others, using standard circuit elements having defined functional and timing characteristics.

asynchronous

The lack of a timing relationship between two different clocks or signals (transitions can occur at any time with respect to each other).

attribute

A string or value associated with an object that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can find out the values of attributes by using the `get_attribute` command.

back-annotation

The process of applying detailed parasitic data to a design, allowing a more accurate timing analysis of the final circuit. The data comes from an external tool after completion of layout.

borrowing

An analysis technique in which a portion of a path is allowed to borrow timing margin from the next stage of the design, when the next stage is a level-sensitive latch operating in transparent mode.

capture

The processes of clocking data into a latch or flip-flop at the endpoint of a path, thus getting the data that was launched by an earlier clock edge at the startpoint of the path.

cell

A component within an integrated circuit with known functional and timing characteristics, which can be used as an element in building a larger circuit.

cell delay

The delay from the input to the output of a cell (logic gate) in a path.

clock

A periodic signal that latches data into sequential elements. You define a signal to be a clock and specify its timing characteristics by using the `create_clock` or `create_generated_clock` command.

clock path

A timing path that starts at a clock input port or a pin of an internal cell and ends at a clock input pin of a sequential element.

clock gating

The control of a clock signal by a combinational logic element such as a multiplexer or AND gate, to either shut down the clock at selected times or to modify the clock pulse characteristics.

clock latency

See [latency](#).

clock reconvergence pessimism

An inaccuracy of static timing analysis that occurs when two different clock paths partially share a common physical path segment, and the analysis tool uses the

minimum delay of the shared segment for one path and the maximum delay of the same segment for the other path. Correction of this inaccuracy is called clock reconvergence pessimism removal (CRPR).

clock skew

See [skew](#).

closing edge

The edge of a clock signal that latches data into a level-sensitive latch, ending the transparent mode and desensitizing the data input.

collection

A set of objects taken from the loaded design. For example, the `set mylist [get_clocks "CLK*"]` command creates a collection of clocks and sets a variable called “mylist” to the collection. Then you can use commands that operate on the collection, such as `remove_clock $mylist`. You can also generate and operate on a collection within a single command, such as `remove_clock [get_clocks "CLK*"]`.

combinational feedback loop

A combinational logic network in which a signal path makes a complete loop back to its starting point. PrimeTime must break a loop (stop tracing the signal) at some point within the loop.

combinational logic

A logic network that only contains elements that have no memory or internal state. Combinational logic can contain AND, OR, XOR, and inverter elements, but cannot contain flip-flops, latches, registers, or RAM.

common point

In a path where clock reconvergence pessimism exists, the common point is the last cell output in the path segment shared between the launch and capture clock paths.

conditional timing arc

A timing arc whose delay values depend on some condition such as the logical state of an input.

conservative

An analysis algorithm or technique that favors pessimism (rather than optimism) to ensure detection of all violations.

constraint

A timing specification or requirement that applies to a path or a design. A timing constraint limits the allowed range of time that signals can arrive at a device input or be valid at a device output.

context characterization

The process of using the `characterize_context` command to capture the timing context of a subdesign in the chip-level timing environment. This process allows standalone timing analysis of the subdesign in PrimeTime or optimization of the subdesign in Design Compiler. The timing context of an instance includes clock information, input arrival times, output delay times, timing exceptions, design rules, propagated constants, wire load models, input drives, and capacitive loads.

context independent

Accurate in different contexts. Note that this term refers to the usability, not behavior, of a timing model. For example, a timing model created with the `extract_model` command is context independent, which means that the model can be used accurately in different contexts (its behavior changes according to the context in which it is used).

CPU time

The amount of time used by the central processing unit of the computer to accomplish a task, as reported by the software. This is a measure of computation resources required for the task. CPU time is less than the actual elapsed time.

critical path

The path in a path group that has the largest violation (or the least timing slack) of all paths in that path group.

crosstalk

An undesirable effect on a net caused by parasitic capacitance between the net and physically adjacent nets. Signal transitions on the adjacent nets can cause noise bump or a change in the transition time on the affected net.

current design

The loaded design currently considered to be the top-level design in a design hierarchy. You can specify the current design with the `current_design` command.

current instance

The instance (hierarchical cell) in a design hierarchy on which instance-specific commands operate by default. You can set the current instance with the `current_instance` command, which works like the `cd` command in UNIX.

data check

A timing check between two data signals, such as handshaking interface signals or recovery/removal checks between preset and clear pins. The two signals being checked are called the constrained and related signals. The related signal is the data signal treated as the clock in the timing relationship. For example, a setup data check verifies that the constrained signal is stable before the active edge of the related signal.

data path

A timing path that starts at a data launch point and ends at a data capture point. The term “path” usually means a data path, although there are other types of paths such as clock paths and asynch_default paths.

.db (database) format

A Synopsys file format used for storing designs, .lib logic libraries, clock information, back-annotated parasitic information, and timing models. You can read some or all of this information from a .db file into PrimeTime by using the `read_db` command. Tools such as Design Compiler and Formality also use .db files.

Design Compiler

The Synopsys core tool that can synthesize a design defined by a hardware description language into an optimized, technology-dependent, gate-level design. Design Compiler supports a wide range of flat and hierarchical design styles and can optimize both combinational and sequential designs for speed, area, and power. Design Compiler and PrimeTime work well together, but they are independent tools that can be used separately.

edge-sensitive latch

A flip-flop; a register element that captures data on each active edge on its clock input. The active edge is a transition from low to high or high to low, depending on the device type.

endpoint

The place in a design where a timing path ends. This is where data gets captured by a clock edge or where the data must be available at a specific time. Every endpoint must be either a register data input pin or an output port.

exception

See [timing exception](#).

exception transformation

The conversion of the current set of exception-setting commands into a new set of commands by the `transform_exceptions` command. The command operates by eliminating invalid, redundant, and overridden paths in the original command set.

exclusive clocks

Two clocks that are never active at the same time (for example, because they are multiplexed).

extracted timing model

A timing model created from a gate-level netlist by the `extract_model` command. This type of model represents the timing characteristics (arc values) of the block. However, it does not contain any functional information, so it cannot be synthesized.

false path

A path that exists in a design that should not be analyzed for timing, such as a path between two multiplexed blocks that are never enabled at the same time. For proper analysis by PrimeTime, you need to define false paths as timing exceptions with the `set_false_path` command or remove the related objects from analysis with the `set_disable_timing` command.

fanin

The set ports and pins that affect a specified timing endpoint (called the sink). A pin is considered to be in the timing fanin of a sink if there is a timing path through combinational logic from the pin to that sink. Fanin tracing stops at the clock pins of registers (sequential cells).

fanout

In PrimeTime, the term fanout usually means timing fanout, also known as transitive fanout. Timing fanout is the set ports and pins at timing endpoints that are affected by a specified source port or pin. A pin is considered to be in the timing fanout of a source if there is a timing path through combinational logic from the source to that pin. Fanout tracing stops at the inputs to registers (sequential cells).

Note that timing fanout is not the same as direct fanout. Direct fanout is the number of cell inputs connected to a cell output.

flip-flop

A memory element controlled by an edge-sensitive clock input. Typically, a flip-flop has an input D, and output Q, a clock input, and possibly asynchronous set and clear inputs. When the active edge occurs on the clock input, the value on the input D is latched and then held constant at the output Q. The active edge can be either rising or falling, depending on the type of flip-flop.

gated clock

A clock signal that can be modified by logic, such as a clock that can be turned off to save power.

generated clock

A clock signal that is generated internally by the integrated circuit itself; a clock that does not come directly from an external source. An example of a generated clock is a divide-by-2 clock generated from the system clock, having half the frequency of the system clock. You specify the characteristics of a generated clock by using the `create_generated_clock` command.

glitch

A noise bump that is large enough to cause a logic failure.

hold constraint

A timing constraint that specifies how much time is necessary for data to be stable at the input of a device after the clock edge that clocks the data into the device. This constraint enforces a minimum delay on a timing path relative to a clock.

I-V characteristics (steady-state)

The current-voltage characteristics of a cell output; the current as a function of voltage while the output is held constant at logic 1 or logic 0. PrimeTime SI uses this information to calculate crosstalk noise effects.

island

See [voltage area](#).

inout pin

A bidirectional pin of a cell; a pin that can operate as both an input and an output.

input delay

A constraint that specifies the minimum or maximum amount of delay from a clock edge to the arrival of a signal at a specified input port. PrimeTime uses this information to check for timing violations at the input port and in the transitive fanout from that input port.

intellectual property

Information owned by one company that is licensed for use by another company. For example, one company might offer a license to use a chip submodule design (such as a microprocessor core) in another company's application-specific circuit. The owner of the submodule design can provide the layout information, electrical specifications, and a timing model to the other company, without revealing the submodule netlist.

interface logic model

A timing model created from a gate-level netlist by the `write_ilm_netlist` command and related commands. This type of model represents the timing characteristics (arc values) of the block and includes the interface logic of the original netlist. However, it does not include any functional information from the inside of the original block, so it cannot be synthesized.

latch

A memory element controlled by level-sensitive gate input. The output of a latch follows the input if the gate signal is active. When the gate signal goes from active to inactive, the input value is latched and the output remains constant at that value. The inactive-to-active edge of the gate signal is called the opening edge. The active-to-inactive edge of the gate signal is called the closing edge.

latency

The amount of time that a clock signal takes to be propagated from the clock source to a specific point inside the design. The clock signal is “hidden” (latent) for this amount of time.

launch

The processes of clocking data out of a latch or flip-flop at the startpoint of a path, releasing data that is captured by another clock edge at the endpoint of the path.

layout

The process of generating the geometric location, size, and form of components and connections for an integrated circuit. From layout information, an external tool can generate accurate information about the parasitic resistance and capacitance of the components and connections. You can then back-annotate the design with this information in PrimeTime for a more accurate timing analysis.

leaf level

The level of design hierarchy containing the lowest-level library cells, like the leaves of a tree.

level-sensitive latch

A register that allows data to pass through from input to output while its gate input is active, and holds its data output constant when the gate input is inactive.

library

A database containing information about the functional and timing characteristics of circuit elements used in a design, also called the logic library or .lib logic library when specified in the syntax of Library Compiler. Logic libraries are typically provided by the ASIC vendor.

Library Compiler

A Synopsys tool used to create library databases for PrimeTime, Design Compiler, and other tools. Library Compiler reads the description of an ASIC library from a text file (.lib file) and compiles the description into a database in .db format for synthesis and analysis, or into a set of VHDL libraries for VHDL simulation.

logic library

See [library](#).

man page

A description of a command, variable, or message code displayed by using the `man` command in PrimeTime. For example, entering `man create_clock` at the `pt_shell` prompt displays the man page describing the `create_clock` command. This is similar to the UNIX `man` command (which is derived from the word “manual”).

multicycle path

A path that is designed to take more than one clock cycle for the data to propagate from the startpoint to the endpoint. For example, a multiplier circuit (a slow combinational logic element) might be designed to take four clock cycles to generate a valid result; the circuit is designed to capture the result after this amount of time has elapsed. For proper analysis, you need to define such a path as a timing exception with the `set_multicycle_path` command.

native

A command or process in PrimeTime that works entirely in PrimeTime itself and does not require an outside program.

net arc

A timing arc from a driver pin to a load pin connected to the same net, having a nonzero delay due to parasitic net capacitance.

net delay

The amount of delay from the output of a cell to the input of the next cell in a timing path. This delay is the result of parasitic capacitance of the interconnection between the two cells.

network latency

The amount of time a clock signal takes to propagate from the clock definition point to a register clock pin.

NLDM

Nonlinear delay model, a standard, table-based format for specifying cell delays in .lib logic libraries.

noise

A temporary deviation or change in the analog voltage of a steady-state net, such as a crosstalk-induced voltage bump at the output of a CMOS gate.

noise bump

An occurrence of noise that appears as hump shape (often modeled as a triangle) when plotted as voltage versus time.

noise immunity curve

A function that specifies the maximum size of a noise bump that can be allowed at the input of a cell without generating propagated noise at the output of the cell. The function is often approximated as a hyperbola.

noise margin

DC noise margin is the difference between the most extreme output voltage for a logic level (such as VOLmax) and the most extreme input threshold considered the same logic level (such as VILmax). AC noise margin is the maximum size of a noise bump, in

volts, that can be allowed at the input of a cell without generating a logic failure at the output of the cell.

noise slack

The amount by which a logic failure is avoided when a noise bump occurs at the input of a cell, equal to the failure threshold voltage minus the bump height, multiplied by the bump width. The units are in library voltage units times library time units.

no-change constraint

The amount of time that a data signal must remain unchanged before, during, and after a pulse of a control signal (for example, an address signal that must be stable during a write pulse). The data signal must be stable for a specified setup time before the leading edge of the control pulse, during the control pulse, and for a specified hold time after the trailing edge of the control pulse.

opening edge

The edge of a clock signal that asserts a gate input of a level-sensitive latch. The latch is transparent from the opening edge to the closing edge of the clock signal.

operating conditions

The process, voltage, and temperature conditions under which a circuit operates, which affect the timing characteristics of the cells.

optimistic

An analysis algorithm or technique with a known accuracy limitation, when the inaccuracy might indicate that a circuit has more timing slack than the real circuit. As a result, a violation in the real circuit might not be detected.

output delay

A constraint that specifies the minimum or maximum amount of delay from an output port to the external sequential device that captures data from that output port. This constraint establishes the times at which signals must be available at the output port to meet the setup and hold requirements of the external sequential element.

parasitic capacitance

Capacitance of a net due to the physical proximity between the net interconnections and adjacent nets or the substrate; or due to charge storage effects of p-n junctions in the net.

parasitic resistance

Resistance of an interconnection or net due to the finite conductivity of the interconnect material.

path

A point-to-point sequence through a design that starts at a register clock pin or an input port, passes through any number of combinational logic elements, and ends at a register data input pin or an output port. Data launched at the path startpoint by a clock

edge is propagated through the combinational logic to the path endpoint, where the data gets captured by another clock edge.

path endpoint

See [endpoint](#).

path group

A group of related paths, grouped either implicitly by the `create_clock` command or explicitly by the `group_path` command. By default, paths whose endpoints are clocked by the same clock are assigned to the same path group. Path groups affect the reporting of violations in PrimeTime. They also affect the relative amount of effort used by Design Compiler to optimize different paths.

path inspector

A window in the PrimeTime GUI used to display detailed information about a timing path such as the path schematic, path element tables, timing waveforms, path delay profiles, and text-format reports.

path startpoint

See [startpoint](#).

pessimistic

An analysis algorithm or technique with a known accuracy limitation, when the inaccuracy might indicate that a circuit has less timing slack than the real circuit. As a result, a violation might be reported that would not actually exist in the real circuit.

pin

An input or output of a cell instance used in a design. The timing constraints for pins are specified in the `.lib` logic library.

port

An input or output of a design. You specify timing constraints for ports with the `set_input_delay` and `set_output_delay` commands.

postlayout

After layout; the time at which detailed parasitic data becomes available for back-annotation on the design.

procedure (Tcl)

A user-defined command created by the `proc` command in PrimeTime. After a procedure is defined, it can be executed like a standard command. A procedure can take arguments and can use local and externally defined variables.

propagated noise

A noise bump on a net caused by noise on the input of the gate that is driving the net. For example, for an inverter whose input is zero and output is one, a positive bump on the input can result in a negative bump on the output.

pt_shell

The text-based, command-line form of PrimeTime that lets you enter commands, run scripts, and view results in text form. By contrast, the GUI (graphical user interface) form of PrimeTime also provides pull-down menus, dialog boxes, command buttons, and graphical presentation of analysis results.

quick timing model

A temporary timing model you create with PrimeTime commands for a block when there is no netlist and no other type of timing model available. You use `create_qtm_model` to create a new model, `create_qtm_port` and `create_qtm_delay_arc` to specify the model characteristics, and `save_qtm_model` to save the completed model.

RC

Resistor-capacitor, a simple parasitic model consisting of a single resistor and capacitor to represent net parasitics.

reconvergence pessimism

See [clock reconvergence pessimism](#).

recovery constraint

The minimum amount of time required between an asynchronous control signal (such as the asynchronous clear input of a flip-flop) going inactive and a subsequent active clock edge. This is like a setup check for the active clock edge. If the active clock edge occurs too soon after the asynchronous input goes inactive, the register data is uncertain because the clocked input data might not be valid.

register

A leaf-level instance of a flip-flop (controlled by an edge-sensitive clock input) or latch (controlled by a level-sensitive gate input).

related signal

The data signal treated as a clock in a data-to-data timing check. The other signal being checked is called the constrained signal.

removal constraint

The minimum amount of time required between a clock edge that occurs while an asynchronous input is active and the subsequent removal of the asserted asynchronous control signal. This is like a hold check for the removal of the asynchronous control signal. If the asynchronous input signal goes inactive too soon after a clock edge, the register data is uncertain because the clocked input data might be valid and conflict with the asynchronous control signal.

RSPF

Reduced Standard Parasitic Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Cadence Design Systems, Inc.

SBPF

Synopsys Binary Parasitic Format, a format used to store circuit parasitic information for back-annotation. Parasitic data in this format occupies less disk space and can be read much faster than the same data stored in SPEF format.

scalable polynomial delay model

A standard, polynomial-based format for specifying cell delays in .lib logic libraries.

script

A text file containing a sequence of pt_shell commands, which can be executed with the source command.

SDF

Standard Delay Format, a standard file format used to store circuit information for back-annotation, including delay information (such as pin-to-pin cell delays and net delays) and timing checks (such as setup, hold, recovery, and removal times).

PrimeTime can read and write SDF files with the read_sdf and write_sdf commands.

sensitize

To find or apply an input vector that causes a particular path to be logically active and to propagate data.

sequential logic

A logic network that contains elements that have memory or internal state, such as flip-flops, latches, registers, or RAM.

setup constraint

A timing constraint that specifies how much time is necessary for data to be available at the input of a device before the clock edge that clocks the data into the device. This constraint enforces a maximum delay on a timing path relative to a clock.

signal integrity

The immunity of a signal or net against crosstalk effects; the characteristic of a net that is not affected by transitions or noise on physically adjacent nets.

skew

The amount of shift or variation away from the nominal, expected time that a clock transition occurs; or the difference in the amount of shift between two different points in a clock network.

slack

The amount of time by which a violation is avoided. For example, if a signal must reach a cell input at no later than 8 ns and is determined to arrive at 5 ns, the slack is 3 ns. A slack of 0 means that the timing constraint is just barely satisfied. A negative slack indicates a timing violation.

slew

The amount of time it takes for a signal to change from low to high or from high to low; also known as transition time.

source latency

The amount of time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point.

SPEF

Standard Parasitic Exchange Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Open Verilog International (OVI).

SPICE

Simulation program with integrated circuit emphasis, a time-based simulation tool that analyzes circuit operation at the level of transistors, capacitors, and resistors. Many different forms of this simulator are available from different sources.

startpoint

The place in a design where a timing path starts. This is where data gets launched by a clock edge or where the data is expected to be available at a specific time. Every startpoint must be either a register clock pin or an input port.

static timing analysis

An efficient analysis that determines whether a circuit violates any timing requirements by considering path delays and timing constraints. Compared with gate-level simulation, static timing analysis is faster and more thorough, and does not require test vectors. However, it is not a replacement for simulation because it does not check the functionality of the circuit.

static noise analysis

An analysis performed by PrimeTime SI that determines worst-case noise effects so that those effects can be minimized or eliminated. This technique considers the cross-coupling capacitance between physically adjacent aggressor and victim nets and the steady-state (one or zero) load characteristics of the victim net.

synchronous

A timing relationship between two clocks or signals in which specific transitions occur at the same time or have a phase difference that stays fixed over time.

Tcl

Tool command language, a standard command scripting language on which the pt_shell interface is based.

Tcl procedure

See [procedure \(Tcl\)](#).

threaded multicore analysis

A flow that enables you to use multiple cores to improve performance by threading the timing update and optimizing handling in other major flow areas.

three-state

An output of a device (such as a bus driver) that can be in any of three logical states: low (0), high (1), or the high-impedance (Z) state.

time borrowing

See [borrowing](#).

timing arc

See [arc](#).

timing exception

An exception to the default (single-cycle) timing behavior assumed by PrimeTime. For PrimeTime to correctly analyze a circuit, you must specify each path that does not conform to the default behavior. Examples of timing exceptions include false paths, multicycle paths, and paths that require a specific minimum or maximum delay time different from the default calculated time.

timing fanin and fanout

See [fanin](#) and [fanout](#).

timing model

A circuit model that contains the timing characteristics of the block, but not necessarily any functional information. These models are often used in hierarchical timing analysis, when analyzing a very large chip design would be too time-consuming if done as a single, flat design.

transforming exceptions

See [exception transformation](#).

transition time

The amount of time it takes for a signal to change from low to high or from high to low; also known as slew.

transitive fanin and fanout

See [fanin](#) and [fanout](#).

transparent

The state of a level-sensitive latch while the gate input is asserted. During this time, signals pass through from input to output and the device operates just like a buffer.

true path

A path that is logically possible to operate; a path that can be sensitized and can propagate data.

uncertainty (clock uncertainty)

The amount of variation away from the nominal, ideal-clock times at which clock edges occur.

vector

A specific set of values applied to the inputs of a device for testing or analysis purposes, or the resulting set of values at the outputs of a device; or a sequence of such values used for sensitizing a path in the design for analysis purposes.

victim net

A net that has undesirable crosstalk effects due to parasitic cross-capacitance between it and another net (called the aggressor net).

voltage area

A physical portion of a chip that uses a different power supply voltage from other portions of the chip.

wire load model

A net resistance and capacitance model used for timing analysis before layout, in the absence of back-annotated delay values or parasitic information. PrimeTime estimates the wire load based on the fanout of each net and library-specified parameters.